СИСТЕМА
УПРАВЛЕНИЯ
БАЗАМИ
ДАННЫХ

ЛИНТЕР®

ЛИНТЕР БАСТИОН

ЛИНТЕР СТАНДАРТ

Справочник по SQL

НАУЧНО-ПРОИЗВОДСТВЕННОЕ ПРЕДПРИЯТИЕ

РЕЛЭКС

Товарные знаки

РЕЛЭКС™, ЛИНТЕР® являются товарными знаками, принадлежащими АО НПП «Реляционные экспертные системы» (далее по тексту – компания РЕЛЭКС). Прочие названия и обозначения продуктов в документе являются товарными знаками их производителей, продавцов или разработчиков.

Интеллектуальная собственность

Правообладателем продуктов ЛИНТЕР® является компания РЕЛЭКС (1990-2026). Все права защищены.

Данный документ является результатом интеллектуальной деятельности, права на который принадлежат компании РЕЛЭКС.

Все материалы данного документа, а также его части/разделы могут свободно размещаться на любых сетевых ресурсах при условии указания на них источника документа и активных ссылок на сайты компании РЕЛЭКС: relex.ru и linter.ru.

При использовании любого материала из данного документа несетевым/печатным изданием обязательно указание в этом издании источника материала и ссылок на сайты компании РЕЛЭКС: relex.ru и linter.ru.

Цитирование информации из данного документа в средствах массовой информации допускается при обязательном упоминании первоисточника информации и компании РЕЛЭКС.

Любое использование в коммерческих целях информации из данного документа, включая (но не ограничиваясь этим) воспроизведение, передачу, преобразование, сохранение в системе поиска информации, перевод на другой (в том числе компьютерный) язык в какой-либо форме, какими-либо средствами, электронными, механическими, магнитными, оптическими, химическими, ручными или иными, запрещено без предварительного письменного разрешения компании РЕЛЭКС.

О документе

Материал, содержащийся в данном документе, прошел доскональную проверку, но компания РЕЛЭКС не гарантирует, что документ не содержит ошибок и пропусков, поэтому оставляет за собой право в любое время вносить в документ исправления и изменения, пересматривать и обновлять содержащуюся в нем информацию.

Контактные данные

394006, Россия, г. Воронеж, ул. Бахметьева, 2Б.

Тел./факс: (473) 2-711-711, 2-778-333.

e-mail: info@linter.ru.

Техническая поддержка

С целью повышения качества программного продукта ЛИНТЕР и предоставляемых услуг в компании РЕЛЭКС действует автоматизированная система учёта и обработки пользовательских рекламаций. Обо всех обнаруженных недостатках и ошибках в программном продукте и/или документации на него просим сообщать нам в раздел Поддержка на сайте ЛИНТЕР.

Содержание

Предисловие 10
Назначение документа 10
Для кого предназначен документ 10
Необходимые предварительные знания 11
Основные понятия и определения 12
База данных 12
Субъекты БД 12
Пользователи 12
Объекты БД 12
Схемы 12
Таблицы 12
Представления 13
Столбцы 13
Индексы 13
Ограничивающие условия целостности 14
Привилегии 14
Роли 15
Синонимы 15
События 15
Последовательности 16
Блокировки 16
Транзакции 16
Хранимые процедуры 16
Временные процедуры 16
Триггеры 16
Кодовые страницы 16
Алиасы 17
Трансляции 17
Элементы языка 18
SQL-символы 18
Литералы 18
Лексемы 25
Имена 26
Типы данных 27
Спецификация значения 34
Спецификация таблицы 41
Спецификация столбца 42
Значимое выражение 43
Побитовые выражения 49
Побитовая операция AND 50
Логическое выражение 50
Предикаты 51
Предикат сравнения 52
Интервальный предикат 55
Предикат вхождения 58
Предикат уникальности 62
Предикат подобия 63
Предикат сопоставления 65
Предикат различимости 70
Предикат соответствия 72
Предикат неопределенного значения 76
Предикат неверного вещественного значения 78
Предикат выборки 79

Предикат существования ... 81
Предикат совмещения ... 82
Предикат внешнего соединения в стиле ORACLE ... 84
Агрегатные функции ... 86
Среднее арифметическое значений ... 87
Медиана значений (MEDIAN) ... 88
Количество записей выборки данных (COUNT) ... 90
Максимальное значение из множества (MAX) ... 91
Минимальное значение из множества (MIN) ... 91
Сумма множества значений (SUM) ... 92
Дисперсия множества числовых значений (VARIANCE) ... 93
Стандартное отклонение множества числовых значений (STDDEV) ... 94
Значение по умолчанию ... 94
Кванторные функции ... 95
Аналитические функции ... 96
Функции ранжирования для интервалов агрегирования ... 96
Ранжирование с учетом дубликатов записей ... 97
Ранжирование с исключением дубликатов записей (DENSE_RANK) ... 102
Последовательное ранжирование записей (ROW_NUMBER) ... 103
Получение первых записей интервалов агрегирования (FIRST_VALUE) ... 105
Получение последних записей интервалов агрегирования (LAST_VALUE) ... 108
Агрегатные функции для интервалов агрегирования ... 109
Кванторные функции для интервалов агрегирования ... 113
Идентификация сгруппированных данных (GROUPING) ... 115
Получение предшествующих данных (LAG) ... 117
Получение последующих данных (LEAD) ... 118
Объединение значений столбца ... 120
Спецификация типа данных ... 124
Спецификация значения по условию ... 126
Табличное выражение ... 131
FROM-спецификация ... 132
WHERE-спецификация ... 135
GROUP BY-спецификация ... 138
HAVING-спецификация ... 150
ORDER BY-спецификация ... 152
OVER-спецификация ... 155
WITHIN GROUP-спецификация ... 156
Соединение таблиц ... 158
Запрос выборки ... 173
select-запрос выборки ... 180
table-запрос выборки ... 185
values-запрос выборки ... 186
Иерархический запрос ... 187
Комбинированный запрос ... 194
Объединение запросов ... 197
Разность запросов ... 207
Пересечение запросов ... 208
Подзапрос ... 209
Тип выборки ... 210
Ограничение выборки ... 210
Управление оптимизацией выполнения запросов ... 213
Подсказка очередности слияния результатов вычисления предикатов ... 214
Подсказка о вычислении предиката последним ... 216
Подсказка о раскрытии представления ... 218
Подсказка о сортировке по заданному индексу ... 219

Подсказка о типе сортировки 220
Подсказки о кэшировании запросов 221
Подсказка для отмены итераторной стратегии 226
Субъекты БД 227
Пользователи 227
Создание пользователя 227
Удаление пользователя (DROP USER) 232
Изменение атрибутов пользователя (ALTER USER) 232
Изменение способа аутентификации 233
Изменение характеристик пароля 234
Добавление в группу 236
Блокирование/разблокирование пользователя 236
Количество одновременно открытых каналов 237
Ограничение количества неудачных соединений 238
Мандатный контроль доступа 239
Изменение графика работы пользователя 240
Доступность рабочих станций 242
Доступность внешних устройств 243
Кэширование результатов поисковых запросов 244
Объекты БД 246
Схемы 246
Создание схемы (CREATE SCHEMA) 246
Установка текущей схемы (SET SCHEMA) 248
Удаление схемы (DROP SCHEMA) 248
Роли 249
Создание роли (CREATE ROLE) 249
Удаление роли (DROP ROLE) 250
Назначение роли (GRANT ROLE) 250
Отмена роли (REVOKE ROLE) 250
Базовые таблицы 251
Создание таблицы (CREATE TABLE) 251
Кодировка символьных данных таблицы 254
Определение мандатного уровня доступа к таблице 256
Определение имени столбца 256
Определение типа данных столбца 256
Определение подставляемого значения столбца 258
Определение автоматически наращиваемого значения столбца 258
Определение значения по умолчанию столбцов таблицы 262
Определение значения из последовательности 264
Определение генерируемого значения 268
Определение фильтра для фразового поиска 271
Определение мандатного уровня доступа к столбцу 271
Определение ограничения NULL-значений 272
Определение ссылочной целостности столбца 273
Определение контролируемого значения столбца 281
Определение первичного ключа таблицы 287
Определение уникального ключа таблицы 288
Определение внешнего ключа таблицы 290
Определение ссылочной целостности таблицы 292
Спецификация параметров таблицы 302
Создание таблицы с загрузкой данных 305
Создание глобальной временной таблицы 308
Создание копии базовой таблицы 309
Удаление таблицы (DROP TABLE) 311
Модификация таблицы (ALTER TABLE) 312

Модификация столбцов таблицы ... 324
Временная модификация таблицы (ALTER TABLE ENABLE/DISABLE) ... 336
Восстановление таблицы (REBUILD TABLE) ... 338
Сжатие таблицы (PRESS TABLE) ... 339
Тестирование таблицы (TEST TABLE) ... 339
Предварительная загрузка таблицы (TEST TABLE PRELOAD) ... 341
Усечение таблицы (TRUNCATE TABLE) ... 342
Таблицы «в памяти» ... 344
Создание таблицы «в памяти» (CREATE TABLE IN-MEMORY) ... 344
Сохранение таблицы «в памяти» (SAVE TABLE) ... 345
Восстановление таблицы «в памяти» (RESTORE TABLE) ... 346
Модификация таблицы «в памяти» ... 347
Представления ... 348
Создание представления (CREATE VIEW) ... 348
Модификация представления (ALTER TABLE) ... 351
Удаление представления (DROP VIEW) ... 351
Привилегии ... 353
Определение привилегий доступа к ресурсам БД и операций с ними (GRANT) ... 353
Отмена привилегий доступа к ресурсам БД и операций с ними (REVOKE) ... 355
Индексы ... 356
Создание индекса (CREATE INDEX) ... 356
Удаление индекса (DROP INDEX) ... 360
Корректировка индекса (CORRECT INDEX) ... 361
Корректировка конвертера (CORRECT INDEX ROWID) ... 363
Корректировка битовой карты таблицы (CORRECT BITMAP) ... 364
Последовательности ... 365
Создание последовательности (CREATE SEQUENCE) ... 365
Удаление последовательности (DROP SEQUENCE) ... 369
Модификация последовательности (ALTER SEQUENCE) ... 370
Выбор значения последовательности (CURRVAL/NEXTVAL) ... 372
Генерация временной последовательности (SELECT LEVEL) ... 374
Триггеры ... 375
Создание триггера (CREATE TRIGGER) ... 375
Компиляция триггера (REBUILD TRIGGER) ... 381
Удаление триггера (DROP TRIGGER) ... 383
Изменение состояния триггера (ALTER TRIGGER) ... 383
Удаление текста триггера (ALTER TRIGGER DROP TEXT) ... 383
Глобальные переменные процедурного языка ... 384
Создание глобальной переменной простого типа (CREATE VARIABLE) ... 384
Удаление глобальной переменной (DROP VARIABLE) ... 385
События ... 386
Создание события (CREATE EVENT) ... 386
Удаление события (DROP EVENT) ... 389
Отключение события (ALTER EVENT DISABLE) ... 390
Включение события (ALTER EVENT ENABLE) ... 390
Оповещение о событии (SET EVENT) ... 391
Ожидание события (WAIT EVENT) ... 391
Опрос события (GET EVENT) ... 392
Сброс события (CLEAR EVENT) ... 393
Синонимы ... 393
Создание синонима (CREATE SYNONYM) ... 393
Удаление синонима (DROP SYNONYM) ... 395
Комментарии ... 396
Создание комментария (COMMENT ON) ... 396
Изменение комментария ... 399

Удаление комментария ... 399
Кодовые страницы ... 400
Создание кодировки (CREATE CHARACTER SET) ... 400
Кодировка системного словаря БД (SET DATABASE NAMES) ... 402
Кодировка соединения по умолчанию (SET NAMES) ... 403
Кодировка данных пользовательских таблиц (SET DATABASE DEFAULT CHARACTER SET) ... 404
Создание правила трансляции (CREATE TRANSLATION) ... 405
Удаление кодировки (DROP CHARACTER SET) ... 406
Удаление правила трансляции (DROP TRANSLATION) ... 406
Создание алиаса кодировки (CREATE ALIAS) ... 407
Удаление алиаса кодировки (DROP ALIAS) ... 408
Создание/изменение описания кодировки (ALTER CHARACTER SET) ... 408
Удаление описания кодировки (ALTER CHARACTER SET DROP) ... 409
Манипулирование данными ... 410
Добавление записи (INSERT) ... 410
Корректировка записи (UPDATE) ... 420
Удаление записи (DELETE) ... 426
Позиционная корректировка записи (UPDATE OF CURSOR) ... 429
Позиционное удаление записи (DELETE OF CURSOR) ... 431
Слияние данных (MERGE) ... 432
Пакетное добавление (START APPEND) ... 438
Начать пакетное добавление ... 438
Закончить пакетное добавление (END APPEND) ... 439
SQL-операторы с параметрами ... 440
Общие сведения ... 440
SQL-параметр ... 441
Параметрическое выражение ... 441
Управление работой СУБД ... 445
Блокирование доступа к данным ... 445
Блокирование таблицы (LOCK TABLE) ... 445
Разблокирование таблицы (UNLOCK TABLE) ... 445
Блокирование доступа к БД (ALTER USER LOCK) ... 446
Разблокирование доступа к БД (ALTER USER UNLOCK) ... 446
Локальные транзакции ... 447
Создание точки сохранения (SET SAVEPOINT) ... 447
Сохранение изменений (COMMIT) ... 448
Отмена изменений (ROLLBACK) ... 449
Управление функционированием ... 449
Включение проверки пароля при запуске ядра (STARTUP AUTHORIZATION) ... 449
Включение режима сертифицированной БД (SET CERTIFIED) ... 450
Включение/отключение режима суперпользователя для DBA (SET SUPERUSER) ... 451
Включение/отключение режима очистки оперативной памяти (SET MEMORY CLEAR) ... 451
Управление максимальным квантом обработки запросов пользователя (SET QUANT TIMEOUT) ... 452
Управление максимальным квантом обработки запросов в сессии (SET SESSION QUANT TIMEOUT) ... 453
Управление квантованием по времени (SET DATABASE QUANT) ... 454
Управление количеством квантов (SET DATABASE QUANTUM FOR) ... 454
Управление количеством выполняемых подряд квантов сессии (SET SESSION QUANTUM FOR) ... 456
Управление протоколированием (SET LOG) ... 457
Управление выводом процедур (SET PROCEDURE TRACE) ... 457

Управление трассировкой выполняемых запросов (SET TRACE) 458
Управление режимом ядра (SET TRUE COMMIT) 460
Размер рабочей области (SET WORKSPACE) 460
Ограничение длины записи (SET RECORD SIZE LIMIT) 461
Управление максимальным размером памяти каналов (SET CHANNEL MEMORY LIMIT) 462
Ограничение времени работы всех неактивных сессий СУБД (ALTER DATABASE SET SESSION TIME LIMIT) 462
Ограничение времени работы отдельной неактивной сессии СУБД (SET SESSION TIME LIMIT) 463
Приоритет пользователя (SET PRIORITY) 464
Управление логированием BLOB-данных (SET SESSION BLOB LOG) 465
Режим выполнения хранимых процедур (SET SESSION PROCEDURE AS DEFINER) 466
Уровни мандатного доступа к записям по умолчанию 467
Уровни мандатного доступа к объектам БД (SET SESSION SECURITY) 468
Режим проверки ограничения ссылочной целостности (SET CONSTRAINTS) 470
Архивирование БД 472
Начать оперативное архивирование (BACKUP DATABASE) 472
Остановить оперативное архивирование (BACKUP STOP) 473
Репликация данных 474
Объявление сервера репликации (CREATE NODE/SERVER) 474
Удаление сервера репликации (DROP NODE/SERVER) 475
Создание правила репликации (CREATE REPLICATION RULE) 475
Изменение правила репликации (ALTER REPLICATION RULE) 477
Удаление правила репликации (DROP REPLICATION RULE) 477
Синхронизация таблиц репликации (SYNCHRONIZE REPLICATION RULE) 478
Расширение SQL 479
Геометрические типы данных 479
Полнотекстовый поиск данных 479
Процедурный язык 479
Расширенные средства защиты данных 479
SQL-интерфейсы 480
Нижний уровень 480
Прикладной уровень 480
Встроенный SQL 480
ODBC 480
JDBC 480
Perl 480
PHP 480
Python 480
Qt 480
Ruby 480
TCL/TK 480
ADO.NET 481
Функции 482
Символьные функции 483
Конкатенация (CONCAT) 483
Перевод начальной буквы слова в заглавную (INITCAP) 484
Поиск подстроки (INSTR/POSITION) 485
Проверка вхождения подстроки по шаблону (REGEXP_LIKE) 487
Выборка подстроки по шаблону (REGEXP_SUBSTR) 495
Корректировка подстроки (INSERT/OVERLAY) 500
Замена всех подстрок (REPLACE) 502
Замена символов строки (TRANSLATE) 504

Определение длины строки (в символах) (LENGTH) 506
Определение длины строки (в байтах) (OCTET_LENGTH) 507
Перевод символов в нижний регистр (LOWER) 508
Перевод символов в верхний регистр (UPPER) 509
Дополнение строки слева (LPAD) 510
Дополнение строки справа (RPAD) 512
Левостороннее удаление символов (LTRIM) 512
Правостороннее удаление символов (RTRIM) 514
Удаление из строки крайних символов (TRIM) 515
Выделение подстроки (SUBSTR/SUBSTRING) 516
Выделение последних символов строки (RIGHT_SUBSTR) 519
Дублирование строки (REPEAT_STRING) 520
Обратный порядок символов строки (REVERSE) 521
Символьное представление байта (CHR) 522
Числовое представление символа (ASCII) 522
Символьное представление значения по умолчанию столбца (DEFTEXT) 523
Получить список ключевых слов СУБД ЛИНТЕР (LINTER_KEYWORDS) 525
Математические функции 525
Общая рекомендация 525
Вычисление абсолютного значения (ABS) 528
Округление до большего целого значения (CEIL) 528
Округление до меньшего целого значения (FLOOR) 529
Остаток от деления (MOD) 529
Тригонометрические функции (COS/SIN/TAN) 530
Обратные тригонометрические функции (ACOS/ASIN/ATAN/ATAN2) 530
Гиперболические функции (COSH/SINH/TANH) 531
Вычисление экспоненты (EXP) 531
Вычисление логарифмических значений (LN/LOG) 531
Вычисление степени (POWER) 532
Округление значения (ROUND) 532
Усечение представления значения (TRUNC) 534
Определение знака числа (SIGN) 536
Вычисление квадратного корня (SQRT) 536
Псевдослучайное значение (RAND) 536
Функции обработки списков 538
Определение максимального значения в списке (GREATEST) 538
Определение минимального значения в списке (LEAST) 539
Функции преобразования 540
Преобразование символьной шестнадцатеричной строки в строку байт (HEXTORAW/HEX) 540
Преобразование значения в шестнадцатеричное представление (RAWTOHEX) 541
Преобразование значимого выражения в символьное представление (TO_CHAR) 541
Преобразование символьного значения во внутреннее представление (TO_NUMBER) 548
Преобразование символьного представления типа «дата-время» во внутреннее (TO_DATE/TO_TIMESTAMP) 549
Преобразование значения в XML-формат (XML) 550
Преобразование значения в HTML-формат (HTML) 551
Преобразование из одной кодировки в другую (CONVERT) 551
Функции бесформатного преобразования (GETBYTE/GETBYTEB) 554
Значение заданного байта 554
Значение заданного слова (GETWORD/GETWORDB) 555
Значение заданного длинного слова (GETLONG) 556

Значение заданного количества бит (GETBITS) 557
Выбор подмножества символов (GETSTR) 558
Выбор подмножества байт (GETRAW) 559
Функции для работы с BLOB-данными 560
Определение числа слов BLOB-значения (COUNTBLOB) 561
Поиск слова (FINDBLOB) 563
Поиск слова в текстовых BLOB-данных (FINDRTF) 564
Местоположение искомых элементов текста (GETTEXTPOS) 565
Выборка текста (GETTEXT) 569
Дать порцию BLOB-значения (GETBLOB) 570
Дать порцию BLOB-значения в текстовом виде (GETBLOBSTR) 571
Определение длины BLOB-значения (LENBLOB) 573
Модификация подстроки BLOB-данных (SET/INSERT) 574
Модификация подстрок BLOB-данных (SET/REPLACE) 576
Функции для работы с типом DATE 577
Текущая дата в формате UNIX (UNIX_TIMESTAMP) 577
Сокращенное название дня даты (DAYNAME) 578
Сокращенное название месяца даты (MONTHNAME) 579
Выделение заданных элементов даты (DATESPLIT/EXTRACT) 579
Сдвиг даты на заданный интервал (MULTIME) 581
Вычисление интервала между двумя датами (DIVTIME) 583
Вычисление количества дней в дате (TO_DAYS/TIMEINT_TO_DAYS) 584
Вычисление даты по количеству дней (FROM_DAYS/TIMEINT_FROM_DAYS) 585
Последний день месяца (LAST_DAY) 586
Дата очередного дня недели (NEXT_DAY) 586
Помесячное изменение даты (ADD_MONTHS) 588
Операции над значениями типа DATE 589
Интервальное время 590
Разница дат в месяцах (MONTHS_BETWEEN) 593
Преобразование числового значения в интервал времени (NUMTODSINTERVAL) . 593
Преобразование строкового числового значения в интервал времени (TO_DSINTERVAL) 594
Работа с часовыми поясами 594
Преобразование времени по Гринвичу к локальному времени (TO_LOCALTIME) 594
Преобразование локального времени ко времени заданного часового пояса (TO_GMTIME) 599
Функции для работы с IP-адресами 602
Преобразование внешнего представления CIDR IP-адреса во внутреннее (CIDRTORAW) 603
Преобразование внутреннего представления CIDR IP-адреса во внешнее (RAWTOCIDR/IP_TEXT) 604
Определение длины префикса CIDR IP-адреса (CIDRLEN/IP_MASKLEN) 604
Установка новой длины префикса CIDR IP-адреса (IP_SET_MASKLEN) 605
Определение CIDR IP-адреса без длины префикса (CIDRPFX) 606
Определение CIDR IP-адреса для совместимости с PostgreSQL (IP_HOST) 607
Проверка адреса по адресной маске (CIDRMATCH) 607
Определение количества IP-адресов (CIDRCOUNT) 608
Выделение сетевого адреса со сбросом бит префикса (CIDRNET) 609
Выделение сетевого адреса (IP_NETWORK) 610
Выделение сетевого адреса с установкой бит префикса (CIDRBCAST) 610
Выделение сетевого адреса с установкой бит префикса для совместимости с PostgreSQL (IP_BROADCAST) 611
Выделение маски сети (CIDRMASK/IP_NETMASK) 612
Выделение сокращенного IP-адреса (IP_ABBREV) 612

Функции для работы с MAC-адресами ... 613
Преобразование внешнего представления MAC-адреса во внутреннее (MACTORAW) ... 613
Преобразование внутреннего представления MAC-адреса во внешнее (RAWTOMAC) ... 615
Обнуление последних трех байт MAC-адреса (MAC_TRUNC) ... 616
Прочие функции ... 616
Проверка разрешения доступа (ACCESS) ... 616
Получить имя БД (DBNAME) ... 618
Получить дату окончания лицензионного периода СУБД (EXPIRATION_DATE) ... 619
Игнорировать ошибку преобразования (NULLIFERROR) ... 619
Замена значения (DECODE) ... 621
Одновариантная замена NULL-значения реальным значением (NVL) ... 622
Двухвариантная замена NULL-значения реальным значением (NVL2) ... 624
Проверка на равенство двух значений (NULLIF) ... 625
Поиск первого реального значения (COALESCE) ... 626
Замена NaN-значения правильным значением (NANVL) ... 627
Генерация уникального идентификатора (SYS_GUID) ... 628
Информация о версии СУБД (LINTER_VERSION) ... 629
Информация о размере файла таблицы (LINTER_FILE_SIZE) ... 630
Информация о местонахождении файлов таблицы (LINTER_FILE_DEVICE) ... 631
Информация о состоянии файлов таблицы (LINTER_FILE_INFO) ... 632
Информация о состоянии страницы файла (LINTER_PAGE_INFO) ... 634
Информация о состоянии страницы файла (LINTER_PAGE_TIME) ... 636
Информация о фразовом индексе (LIN_INDEX_INFO) ... 637
Информация о длине внутреннего представления выражения (VSIZE) ... 639
Информация об ограничениях целостности (LINTER_DICT_INFO) ... 640
Информация о состоянии событий (GET_EVENTS_STATE) ... 642
Фонетический код значения (SOUNDEX) ... 643
Определение близости фонетического звучания (DIFFERENCE) ... 644
Приложение 1. Ключевые слова СУБД ЛИНТЕР ... 646
Приложение 2. Контекстно зависимые ключевые слова СУБД ЛИНТЕР ... 651
Указатель определяемых элементов ... 656
Указатель операторов ... 660
Указатель функций ... 662

Предисловие

Назначение документа

Язык SQL (SQL – Structured Query Language) является языком обработки и манипулирования данными для реляционных БД. Он основан на стандарте ANSI X3.135-2008 (SQL:2008) за исключением нереляционных возможностей, который определяет логические структуры данных и базовые операции для базы данных SQL, обеспечивает функциональные возможности для проектирования, доступа, поддержания, управления и защиты базы данных.

При реализации языка в СУБД ЛИНТЕР в него внесены некоторые элементы, не специфицированные в стандарте SQL:2008, а именно:

- интернационализация имен объектов БД (таблиц, столбцов и пр.);
- средства для работы в режиме реального времени;
- средства оперативного тестирования таблиц БД;
- средства оперативного архивирования объектов БД;
- средства поддержки кодовых страниц для представления системной и пользовательской информации;
- набор скалярных функций для поддержки ODBC-интерфейса;
- набор скалярных функций для совместимости SQL-сервера СУБД ЛИНТЕР с SQL-сервером СУБД Oracle.

Дополнительно к этому в язык SQL СУБД ЛИНТЕР внесены:

- команды управления комплексом средств защиты информации;
- команды организации полнотекстового поиска в БД;
- средства встраивания SQL для систем программирования C/C++;
- средства поддержки процедурного языка СУБД ЛИНТЕР;
- средства поддержки репликации (тиражирования) данных;
- средства поддержки геометрических типов данных.

Документ предназначен для СУБД ЛИНТЕР БАСТИОН 6.0 сборка 20.8, далее по тексту СУБД ЛИНТЕР.

Для кого предназначен документ

Документ предназначен для профессиональных пользователей базы данных и программистов, разрабатывающих информационные системы на основе СУБД ЛИНТЕР.

Дополнительные документы

- Администрирование комплекса средств защиты данных
- Архивирование и восстановление базы данных

- Архитектура СУБД
- Встроенный SQL
- Геометрические типы данных
- Запуск и останов СУБД ЛИНТЕР в среде ОС Linux
- Запуск и останов СУБД ЛИНТЕР в среде ОС Windows
- Интерфейс нижнего уровня
- Командный интерфейс
- Объектная библиотека прикладного интерфейса
- Полнотекстовый поиск в базе данных
- Прикладной интерфейс
- Процедурный язык
- Репликация данных
- Сетевые средства
- Создание и конфигурирование базы данных
- Справочник кодов завершения
- Тестирование базы данных
- JDBC-драйвер
- ODBC-драйвер
- Perl-интерфейсы
- PHP-интерфейсы
- Python-интерфейс
- Qt-интерфейс
- Ruby-интерфейс
- TCL/TK-интерфейс

Необходимые предварительные знания

Для использования языка SQL в СУБД ЛИНТЕР необходимо:

- знать основы реляционных баз данных;
- уметь работать в соответствующей операционной системе на уровне подготовленного пользователя.

Часть примеров SQL-конструкций использует данные таблиц `AUTO`, `PERSON` и `FINANCE`. Указанные таблицы находятся в демонстрационной БД СУБД ЛИНТЕР (подкаталог `db` установочного каталога СУБД ЛИНТЕР).

Основные понятия и определения

База данных

База данных (БД) – это совокупность данных, организованных в виде таблиц по определенным правилам, предусматривающим общие принципы описания, хранения и манипулирования данными.

БД ЛИНТЕР состоит из *системных* (с метаданными) и пользовательских таблиц. В системных таблицах содержатся описания как самих системных таблиц, так и пользовательских таблиц БД, пользователей БД, их привилегий и т.п. Совокупность системных таблиц составляет системную БД.

Создатель БД является владельцем системных таблиц и администратором БД с возможностью создавать новых пользователей БД и предоставлять им часть привилегий. Создатель БД обладает некоторыми привилегиями, отсутствующими у пользователя с привилегией DBA.

Субъекты БД

Субъекты БД – это пользователи.

Пользователи

Пользователь БД – зарегистрированный в БД субъект, имеющий доступ к БД в соответствии с назначенными ему привилегиями.

Каждый пользователь БД имеет уникальное имя (длиной до 66 символов) и пароль (длиной до 18 символов). Утерянный пароль восстановлению не подлежит. В этой ситуации любой администратор БД может разрешить пользователю создать новый пароль, но в случае утери пароля единственным создателем БД доступ к ней будет невозможен.

Объекты БД

Схемы

Схема – это именованная группа объектов БД, имеющая владельца.

Схемы задают пространство имен для именования объектов БД (таблиц, представлений, синонимов, индексов, последовательностей, триггеров и процедур). В БД могут существовать два объекта БД с одинаковыми именами, но с различными схемами.

Создавать объекты БД в схеме может только владелец схемы.

Таблицы

Базовая таблица – это упорядоченное множество из одного или более столбцов и нуля или более неупорядоченных строк. Каждый столбец имеет имя и тип данных. Строка – это последовательность значений столбцов. Строки одной таблицы имеют одинаковое число значений и содержат значение каждого столбца этой таблицы. Строка

– наименьшая единица данных, которая может быть добавлена в таблицу или удалена из нее.

Базовая таблица – это реально существующая в БД таблица, которая физически представлена, как минимум, файлом данных и файлом индексов.

Имя таблицы должно быть уникальным среди имен всех таблиц, представлений и синонимов в рамках схемы.

Представления

Представление – это «виртуальная таблица», которая непосредственно не существует в БД, но для пользователя выглядит так, как будто существует. В отличие от базовой таблицы, которая содержит данные только собственно самой таблицы, представление может «содержать» одновременно данные нескольких базовых таблиц, из которых оно порождено.

Поскольку представление рассматривается как виртуальная таблица, имя представления должно быть уникальным среди имен всех таблиц, представлений и синонимов в рамках схемы.

Столбцы

Столбец – множество значений, которое способно изменяться во времени. Значения одного столбца имеют одинаковый тип (см. раздел «Типы данных») и принадлежат одной таблице.

Значение столбца – это наименьшая единица данных, которая может быть выбрана из столбца или изменена.

Столбец имеет описание и порядковый номер внутри таблицы. Описание столбца содержит список его атрибутов и их значения:

- тип данных;
- длину данных;
- запрещены или разрешены в столбце NULL-значения;
- является или нет столбец автоинкрементным;
- является ли столбец первичным ключом;
- список допустимых значений;
- значение по умолчанию.

Индексы

Значения столбца базовой таблицы или некоторых агрегатных функций являются индексированными, если по их значениям создается *индекс*, то есть специальная структура данных, позволяющая СУБД выполнять ускоренный доступ к записям таблицы по значениям полей этой записи (по сравнению с доступом к неиндексированным значениям).

Индекс в СУБД ЛИНТЕР не является постоянным свойством столбца (функции). При необходимости он может быть создан и удален. Изначально все столбцы таблицы, кроме имеющих свойство первичного ключа, внешнего ключа или уникального ключа, являются неиндексированными.

Таблица может иметь *составные индексы*, то есть индексы, построенные на значениях нескольких столбцов таблицы.

Ограничивающие условия целостности

Ограничивающие условия целостности определяют правила, выполнение которых должна проверять СУБД ЛИНТЕР для поддержки логически целостного состояния БД.

Ограничивающие условия целостности контролируются после выполнения каждого запроса. Если при этом обнаружено, что условия нарушены, результаты выполнения запроса аннулируются, и пользователю передается код исключительного значения.

В СУБД ЛИНТЕР можно задавать следующие ограничения целостности:

- `NOT NULL` – столбец не допускает NULL-значений;
- `AUTOINC` – столбец допускает только целочисленные значения, большие всех значений, находящихся в этом столбце (за исключением случая AUTOINC с диапазонами). Если при добавлении строки в таблицу, имеющую столбец с данным ограничением целостности, значение этого столбца не указывается, то по умолчанию оно будет равно последнему занесенному значению плюс 1. При исчерпании диапазона нарастающих значений столбца циклический переход на начальное значение не выполняется, а фиксируется ошибка добавления строки в таблицу;
- `PRIMARY KEY` – помечает столбец или комбинацию столбцов в качестве первичного ключа таблицы. В таком столбце NULL-значения не допускаются. Каждое вводимое в столбец значение должно быть уникальным для одностолбцового первичного ключа и, возможно, повторяющимся в столбце, но создающим уникальную комбинацию в многостолбцовом первичном ключе. Ввод совпадающих значений первичных ключей запрещен;
- `UNIQUE` – помечает столбец или комбинацию столбцов в качестве уникального ключа таблицы. Данное ограничение гарантирует уникальность значений столбца, но допускает NULL-значения;
- `DEFAULT` – определяет значение по умолчанию, которое автоматически вставляется СУБД, если при вводе данных значение столбца не задано;
- `CHECK` – определяет диапазон допустимых значений столбца и/или строк таблицы, задавая условия, которым должны удовлетворять эти значения;
- `FOREIGN KEY` – определяет столбец как внешний ключ, то есть указывает на то, что значения данного столбца соответствуют значениям первичного или уникального ключа той же самой или иной таблицы БД;
- `REFERENCES` – связывает вместе первичные и внешние ключи. Значение внешнего ключа должно либо полностью совпадать с некоторым значением соответствующего первичного ключа, либо быть NULL-значением.

Привилегии

Пользователю назначается одна из трех категорий доступа к объектам БД:

1) CONNECT-категория. Предоставляет пользователю наименьшие права: доступ к БД с возможностью подавать SQL-запросы на манипулирование данными;
2) RESOURCE-категория. Предоставляет пользователю все права Connect-категории, а также право изменять схему БД (создавать/удалять/изменять структуру объектов БД) и передавать другим пользователям права на свои объекты;

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

3) DBA-категория. Предоставляет пользователю уровень администратора БД с максимальными правами, включающими права Resource-категории и право создавать новых пользователей БД.

Владельцем таблицы считается пользователь, создавший эту таблицу (для ее создания он должен иметь, как минимум, RESOURCE-категорию). Владелец получает на таблицу все возможные привилегии, вплоть до привилегии передачи (отмены) некоторых из них другим пользователям.

Кроме того, пользователю может быть предоставлена привилегия `BACKUP` на выполнение команды архивирования БД средствами СУБД (по умолчанию эту привилегию имеет только создатель БД).

Владелец может передать на свою таблицу следующие привилегии (или их совокупность):

- `SELECT` – на чтение данных;
- `INSERT` – на добавление данных;
- `UPDATE` – на модификацию данных;
- `DELETE` – на удаление данных;
- `ALTER` – на изменение параметров таблицы;
- `INDEX` – на построение/удаление индексов таблицы;
- `REFERENCES` – на создание внешних ключей, ссылающихся на таблицу;
- `ALL` – полный набор привилегий, т.е.

`SELECT+INSERT+UPDATE+DELETE+ALTER+INDEX+REFERENCES`.

Удалить объект БД может только его владелец или администратор БД.

Для вызова хранимой процедуры необходимо иметь привилегии `EXECUTE` или `EXECUTE AS OWNER`.

Кроме того, пользователю может быть предоставлена привилегия `BACKUP` на выполнение команды архивирования БД средствами СУБД (по умолчанию эту привилегию имеет только создатель БД).

Роли

Роль – именованный набор привилегий. Роль может быть назначена пользователю БД, тогда он получает все привилегии, присущие этой роли.

Синонимы

Синоним – альтернативное имя, присваиваемое некоторым объектам БД. Он используется для того, чтобы скрыть реальное имя объекта или реального владельца объекта.

События

Событие – механизм БД, предоставляющий возможность отслеживать их текущее состояние в режиме реального времени.

Последовательности

Последовательность – механизм БД, предоставляющий пользователю возможность генерации уникальных для всей БД значений из заданного диапазона.

Блокировки

Блокировка – механизм, предотвращающий одновременное изменение одних и тех же данных несколькими пользователями и просмотр данных в момент выполнения транзакции.

Транзакции

Транзакция – последовательность операций над БД, которая должна рассматриваться СУБД как единое неразрывное действие по изменению состояния БД.

Каждая транзакция завершается командой `COMMIT` или `ROLLBACK`. Если транзакция завершена командой `COMMIT`, то изменения, сделанные в БД этой транзакцией, фиксируются, то есть становятся доступными для всех действующих одновременно транзакций; в противном случае они отменяются (БД откатывается к предыдущему состоянию). Фиксированные изменения не могут быть отменены.

Хранимые процедуры

Хранимая процедура – хранимый объект БД, написанный на процедурном языке СУБД ЛИНТЕР и предназначенный для выполнения операций по обработке данных. Обращение к процедуре выполняется из пользовательских приложений по имени процедуры с указанием соответствующих параметров.

Работы с хранимыми процедурами описана в документе «Процедурный язык», раздел «Хранимые процедуры».

Временные процедуры

Временная процедура – объект БД, написанный на процедурном языке СУБД ЛИНТЕР и предназначенный для выполнения операций по обработке данных. В отличие от хранимых процедур, временные процедуры компилируются и выполняются за один шаг: скомпилированная временная процедура помещается в рабочую область ядра СУБД, откуда и извлекаются результаты её выполнения. Обращение к временной процедуре невозможно, допустимо только ее выполнение.

Работа с временными процедурами описана в документе «Процедурный язык», раздел «Временные процедуры».

Триггеры

Триггер – специальная хранимая процедура, которая автоматически активизируется при наступлении в БД события, на которое настроен триггер.

Кодовые страницы

Кодовая страница – именованный набор символов, используемых для хранения, обработки и отображения символьной информации, хранящейся в БД. Каждый

пользователь БД для обработки своих данных может использовать любую кодовую страницу, предварительно созданную и загруженную в БД.

Алиасы

Алиас – псевдоним (дополнительное имя) для кодовой страницы. Стандартные имена кодовых страниц ОС представлены, как правило, в виде аббревиатур или условных названий кодировок (например, `CP1251`, `KOI8-R`), что от неподготовленного пользователя требует дополнительных усилий для уточнения кодировки и правильного ее применения. Использование алиасов упрощает этот процесс. Так, например, кодовой странице `KOI8-R` можно назначить алиас «Кириллица», а `CP866` – алиас «`DOS`». Тогда в операторе создания таблицы для указания кодовой страницы, в которой должны храниться данные, можно будет использовать вместо `KOI8-R` алиас «`Кириллица`».

Трансляции

Трансляция – именованная операция, устанавливающая преобразование (перекодирование) одного множества символов в другое. Трансляция может задавать как однобайтовые, так и многобайтовые преобразования (в многобайтовой трансляции один символ будет представлен несколькими байтами).

Элементы языка

SQL-символы

Функция

Определение терминальных символов и элементов строк SQL-языка.

Спецификация

[1] <SQL-символ> : : =
<цифра>
| <шестнадцатеричная цифра>
| <буква>
| <специальный символ>
[2] <цифра> : : = 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
[3] <шестнадцатеричная цифра> : : =
<цифра> | A | B | C | D | E | F | a | b | c | d | e | f
[4] <буква> : : = <латинская буква> | <русская буква>
[5] <латинская буква> : : =
<строчная латинская буква> | <заглавная латинская буква>
[6] <русская буква> : : =
<строчная русская буква> | <заглавная русская буква>
[7] <строчная латинская буква> : : =
a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z
[8] <заглавная латинская буква> : : =
A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z
[9] <строчная русская буква> : : =
а | б | в | г | д | е | ё | ж | з | и | й | к | л | м | н | о | п | р | с | т | у | ф | х | ц | ч | ш | щ | ъ | ы | ь | э | ю | я
[10] <заглавная русская буква> : : =
А | Б | В | Г | Д | Е | Ё | Ж | З | И | Й | К | Л | М | Н | О | П | Р | С | Т | У | Ф | Х | Ц | Ч | Ш | Щ | Ъ | Ы | Ь | Э | Ю | Я
[11] <специальный символ> : : =
<пробел> | « | » | & | ' | (|) | * | + | , | - | . | / | : | ; | < | = | > | ? | [|] | _ | | |

Литералы

Функция

Определение постоянного не NULL-значения.

Спецификация

[1] <литерал> : : =
<символьный литерал>
| <байтовый литерал>
| <двоичный литерал>
| <UNICODE-литерал>
| <числовой литерал>
| <дата-время литерал>
| <логический литерал>
[2] <символьный литерал> : : = ' [<представление символа>...] '
[3] <представление символа> : : =
<цифра>

© Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

| <буква>
| <специальный символ>
| <одинарная кавычка>
[4] <одинарная кавычка>::= ''
[5] <байтовый литерал>::=
<шестнадцатеричный байтовый литерал>
<шестнадцатеричный числовой литерал>
[6] <шестнадцатеричный байтовый литерал>::=
HEX ('<представление байта>...')
| x'<шестнадцатеричная цифра>…'
[7] <шестнадцатеричный числовой литерал>::=
[[+] | -]0x<шестнадцатеричная цифра>…
[8] <представление байта>::=
<шестнадцатеричная цифра>[<шестнадцатеричная цифра>]
[9] <двоичный литерал>::=
<двоичный байтовый литерал>
<двоичный числовой литерал>
[10] <двоичный байтовый литерал>::=
[[+] | -] b'<двоичная цифра>…'
[11] <двоичный числовой литерал>::= [[+] | -] 0b<двоичная цифра>…
[12] <двоичная цифра>::= 0 | 1
[13] <UNICODE-литерал>::= N'<символьный литерал>'
[14] <числовой литерал>::=
<целочисленный литерал>
| <точный числовой литерал>
| <приближенный числовой литерал>
[15] <целочисленный литерал>::= [[+] | -] <беззнаковое целое>
[16] <точный числовой литерал>::=
[[+] | -] <беззнаковое целое>.<беззнаковое целое>
[17] <приближенный числовой литерал>::=
<мантисса> [E] <экспонента>
[18] <мантисса>::= <точный числовой литерал>
[19] <экспонента>::= [[+] | -] <беззнаковое целое>
[20] <беззнаковое целое>::= <цифра>[<цифра>…]
[21] <дата-время литерал>::= <представление даты>
[22] <представление даты>::=
<формат 1> | <формат 2> | <формат 3> | <формат 4> | <формат 5> | <формат 6>
[23] <формат 1>::= DD-MM-[YY]YY[:HH[:MI[:SS[.FF]]]]
[24] <формат 2>::= MM/DD/[YY]YY[:HH[:MI[:SS[.FF]]]]
[25] <формат 3>::= DD.MM.[YY]YY[:HH[:MI[:SS[.FF]]]]
[26] <формат 4>::= DD-MON-[YY]YY[:HH[:MI[:SS[.FF]]]]
[27] <формат 5>::= YYYY-MM-DD[:HH[:MI[:SS[.FF]]]]
[28] <формат 6>::= YYYYMMDD
[29] <логический литерал>::=
TRUE | true | 'TRUE' | 'true'| FALSE | false | 'FALSE' | 'false'

Синтаксические правила

1) <Символьный литерал> задает последовательность символов, начинающуюся и заканчивающуюся апострофом (одинарной кавычкой). Длина символьного литерала не более 4000 знаков.
2) <Представление символа> задает представление любого отображаемого символа, отличного от одинарной кавычки (' – апостроф).
3) Типом данных <символьного литерала> является CHAR. <Символьный литерал> имеет длину, равную числу содержащихся в нем символов.

```
select length('литерал');

|7 |
```

4) Каждая одинарная кавычка внутри <символьного литерала> представляется как <одинарная кавычка> (то есть парой символов одинарных кавычек).

```
select 'Трубы 1,5 дюйма: (1,5'''')';

|Трубы 1,5 дюйма: (1,5'')|
```

5) <Символьный литерал> допускает нулевую длину (то есть полное отсутствие символов).

```
select length('');

|0 |
```

6) Представление <символьного литерала> может совпадать с ключевым словом.

```
select 'table', 'create index';

|table |create index |
```

7) Типом данных <шестнадцатеричного байтового литерала> является BYTE.

8) <Байтовый литерал> задает последовательность байт, представленных в виде шестнадцатеричных чисел. Длина <байтового литерала> не более 8000 шестнадцатеричных цифр.

9) Шестнадцатеричная цифра представляет 4 бита.

Шестнадцатеричная цифра	Битовое представление
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

10) Буквенное представление цифр (A-F) задается буквами латинского алфавита на любом регистре (верхнем или нижнем).

```
select 0x3C, cast X'020000000000001A22124B2006000000' as byte(16);

|    60| 02 00 00 00 00 00 00 1A 22 12 4B 20 06 00 00 00|

hex('CFF0') – битовая маска '1100111111110000'

0x0A0D – символы «Перевод строки» (LF), «Возврат каретки» (CR)

0xB0B0B0B0 – представление графических ASCII-символов ▒▒▒▒
```

Примечание
Если из контекста ясно, что литерал должен иметь тип NCHAR или NCHAR VARYING, то его значение автоматически преобразуется к этому типу.

11) <Шестнадцатеричный байтовый литерал> имеет длину, равную числу содержащихся в нем байт.

```
select length(hex('00ffac0d'));

|4 |
```

12) <Шестнадцатеричный байтовый литерал> допускает нулевую длину (то есть полное отсутствие байт).

```
select length(hex(''));

|0 |
```

13) Каждый байт в <шестнадцатеричном байтовом литерале> представляется строго двумя шестнадцатеричными цифрами.

14) Если количество шестнадцатеричных цифр в литерале нечетное, по умолчанию недостающая цифра считается равной 0.

```
select cast hex('F') as byte(1), hex('f2');

|F0|F2|
```

15) <Шестнадцатеричный числовой литерал> задает допустимое целочисленное значение. Если количество шестнадцатеричных цифр в литерале нечетное, по умолчанию недостающая цифра считается равной 0.

```
select 0x45af3d;

|4566845 |
```

16) Максимальная длина <шестнадцатеричного байтового литерала> 4000 байт.

17) При использовании <шестнадцатеричного байтового литерала> в конструкции CAST результат преобразования зависит от длины литерала:

- если длина меньше или равна 8 байт, он преобразуется в тип данных INTEGER (обратный порядок байт в байтовом представлении INTEGER);
- если длина больше 8 байт, он преобразуется в байтовую строку.

```
select cast 0xFFF0 as integer, cast 0xFFF0ac456745dd45abcd as
 varbyte(5);

|65520 |FF0AC4567|
```

18) Типом данных <двоичного байтового литерала> является BYTE.

19) <Двоичный байтовый литерал> задает последовательность байт, представленных в виде двоичных чисел. Длина <двоичного литерала> не более 32000 двоичных цифр.

```
select b'1111111100000011000001';

|FF0182|
```

20) <Двоичный байтовый литерал> имеет длину, равную числу содержащихся в нем байт.

```
select length(b'10010011');

|1|
```

21) <Двоичный байтовый литерал> допускает нулевую длину (то есть полное отсутствие байт).

```
select length(b'');

|0 |
```

22) Каждый байт в <двоичном байтовом литерале> представляется восемью двоичными цифрами. Если количество двоичных цифр в литерале не кратно 8, по умолчанию недостающие цифры считаются равными 0.

```
select b'1', b'10000000';

|80 |80|
```

23) <Двоичный числовой литерал> задает допустимое целочисленное значение. Если количество двоичных цифр в литерале не кратно 8, по умолчанию недостающие цифры считаются равными 0.

```
select 0b0101111,cast 0b0101111 as byte(1);

|94 |5e|

select -0b0101111, cast -0b0101111 as byte(1),
  cast (256-94) as byte(1);

|-94 |a2|a2|
```

24) <Точный числовой литерал> может иметь тип данных DECIMAL (DEC, NUMERIC), BIGINT, INTEGER (INT) или SMALLINT.

25) <Точный числовой литерал> типа BIGINT, INTEGER (INT) или SMALLINT задает число со знаком или без знака, максимум 19 цифр, которые не включают десятичную точку.

26) Тип данных целочисленного литерала определяется величиной.

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

Диапазон значения	Тип литерала
от -32 767 до +32 767	SMALLINT
от -2 147 483 647 до +2 147 483 647	INT **Примечание** При необходимости может быть сконвертирован в SMALLINT. Если не помещается в эти границы – рассматривается как BIGINT
от -9 223 372 036 854 775 807 до +9 223 372 036 854 775 807	BIGINT

27) <Точный числовой литерал> имеет точность, равную количеству цифр в его представлении. Масштаб <точного числового литерала> определяется количеством цифр после реальной или подразумеваемой десятичной точки.

28) <Точный числовой литерал> типа DECIMAL задает знаковое или беззнаковое десятичное число с общим количеством цифр не более 30 и максимум 10 цифр после возможной десятичной точки. Точность литерала – общее число цифр в его представлении (включая лидирующие и завершающие нули). Масштаб – число цифр справа от десятичной точки (включая младшие нули).

29) <Точный числовой литерал> может не содержать десятичную точку, которая подразумевается за последней цифрой. В этом случае литерал представляет целое значение.

```
select 5, 5., 5.0, 0.5, .5;

|5 |5.0 |5.0 |0.5 |0.5 |
```

30) <Приближенный числовой литерал> всегда рассматривается как DOUBLE, хотя при необходимости может быть сконвертирован в REAL.

31) Точность <приближенного числового литерала> типа REAL – 6 десятичных цифр, точность <приближенного числового литерала> типа DOUBLE – 15 десятичных цифр.

32) <Приближенный числовой литерал> задает вещественное десятичное число, представленное в виде мантиссы и характеристики (стандартное математическое представление). Первое число (мантисса) может включать знак и десятичную точку; второе число (характеристика) – только знак (без десятичной точки). Значением литерала является произведение характеристики и значения мантиссы в степени 10.

```
select -333111e-3, .555E+3;

|-333.111 |555 |
```

33) <Дата-время литерал> символьный литерал в одном из допустимых форматов <представления даты>. Для явного преобразования его в значение DATE необходимо использовать конструкцию CAST, например:

```
cast '28.04.1950' as date.
```

Неявное преобразование <дата-время литералов> к типу DATE выполняется при вставке их значений в столбцы типа DATE или при сравнении их со значениями типа DATE, например:

```
select $$$s13, rowtime from $$$sysrl where rowtime > '19.10.2011';
```

34) В <дата-время литерале> DD представляет день месяца (число от 1 до 31), MM – номер месяца (число от 1 до 12), MON – первые три буквы названия месяца, [YY] – номер века, YY – год века, HH – количество часов (число от 0 до 24), MI – количество минут (число от 0 до 59), SS – количество секунд (число от 0 до 59), FF – количество тиков (число от 0 до 99).

```
'13-11-1992'
'11/13/92:7:30'
'27.3.2003:15:45'

'28-апр-50'
'1875-01-12'
'1-sep-2003:24:03:35:88'
```

35) В <дата-время литерале> допускается задавать параметры одной цифрой.

```
select '8-5-39:2:5:';

|8-5-39:2:5: |
```

36) Если показания времени опущены, то дата считается введенной на момент времени 0 HH 0 MI 0 SS 0 FF (0 час. 0 мин. 0 сек. 0 тиков). Если не указаны цифры, обозначающие век, а YY<38, то считается XXI век Н.Э. (т.е. [YY]=20), иначе – XX век Н.Э. (т.е. [YY]=19).

```
select to_char('18-05-19','dd.mm.yyyy'),
       to_char('18-05-39','dd.mm.yyyy');

|18.05.2019 |18.05.1939 |
```

37) Следующие форматы <дата-время литерала> являются форматами по умолчанию, то есть распознаются автоматически:

- DD.MM.[YY]YY[:[HH:[MI:[SS.[FF]]]]];
- MM/DD/[YY]YY[:[HH:[MI:[SS.[FF]]]]];
- DD-MON-[YY]YY[:[HH:[MI:[SS.[FF]]]]];
- YYYY-MM-DD[:[HH:[MI:[SS.[FF]]]]].

```
create or replace table tab
  (d1 date default '25.01.2013:15:22:32.11');
create or replace table tab
  (d1 date default '01/25/2013:15:22:32.11');
create or replace table tab
  (d1 date default '25-AUG-2013:15:22:32.11');
create or replace table tab
  (d1 date default '2003-01-25:15:22:32.11');
```

38) <Логический литерал> можно задавать в одинарных кавычках: 'TRUE', 'true', 'FALSE', 'false' и т.п.

```
create or replace table tb (b boolean default 'TRUE');
insert into tb default values;
insert into tb values ('true');
insert into tb values ('false');
```

```
select rowid, b from tb where b = 'True';
select rowid, b from tb where b = 'False';
```

Лексемы

Функция

Определение лексических единиц SQL-языка.

Спецификация

[1] <лексема>::=
<литерал>
| <идентификатор>
| <ключевое слово>
| <разделитель>
[2] <идентификатор>::= <стандартный идентификатор>
[3] <стандартный идентификатор>::=
{<латинская буква> | $ } [{<цифра> | <латинская буква> | _ | $}…]
[4] <ключевое слово>::=
неопределяемый элемент синтаксической конструкции
[5] <разделитель>::=
, | (|) | < | > | . | : | = | * | + | - | / | <> | >= | <= | <комментарий>
[6] <комментарий>::=
{<однострочный комментарий> | <многострочный комментарий>}
[7] <однострочный комментарий>::= -- [<символ> …]
[8] <многострочный комментарий>::= /* [<символ> …] */
[9] <символ>::= любой символ или графический знак

Синтаксические правила

1) <Идентификатор> должен содержать не более 66-ти символов.
2) <Идентификатор> может быть представлен двумя способами:
   - без двойных кавычек. В этом случае символами <идентификатора> могут быть только цифры, латинские буквы, знаки подчеркивания (_) и доллара ($). <Идентификатор> может начинаться с символа $ или _ и не должен совпадать с ключевыми словами СУБД ЛИНТЕР. Все такие <идентификаторы> приводятся к верхнему регистру.

Примечание
Во избежание возможных коллизий не рекомендуется использовать имена, начинающиеся с символа $, т.к. их присваивает СУБД ЛИНТЕР создаваемым ею объектам БД.

Одинаковые идентификаторы:

```
BANK, Bank
```

   - заключен в двойные кавычки. В этом случае могут использоваться любые символы. <Идентификатор> может совпадать с ключевыми словами СУБД ЛИНТЕР, и преобразование регистра представления знаков <идентификатора> не выполняется.

Разные идентификаторы:

```
"BANK", "Bank", "банк", "Банк"
```

Допустимые идентификаторы:

```
"Table", SYSTEM."Склад", "Табельный номер"
```

3) <Идентификаторы>, не упомянутые как ключевые в стандарте SQL (см. приложение 1), считаются ключевыми только в том случае, если они встречаются в определенном контексте (см. приложение 2).

   Таким образом, контекстно зависимые ключевые слова распознаются только в том контексте, где они могут встретиться, в других случаях они считаются идентификаторами.

Имена

Функция

Определение имен.

Спецификация

[1] <имя пользователя>::= <идентификатор>
[2] <имя схемы>::= <идентификатор>
[3] <имя псевдонима>::= <идентификатор>
[4] <имя хранимой процедуры>::= <идентификатор>
[5] <имя триггера>::= <идентификатор>
[6] <имя курсора>::= <идентификатор>
[7] <имя SQL-параметра>::= <идентификатор>
[8] <имя кодировки>::= <идентификатор>
[9] <имя трансляции>::= <идентификатор>
[10] <имя алиаса>::= <идентификатор>
[11] <имя точки сохранения>::= <идентификатор>
[12] <имя роли>::= <идентификатор>
[13] <имя индекса>::= <идентификатор>
[14] <имя фильтра>::= <идентификатор>
[15] <имя синонима>::= <идентификатор>
[16] <имя события>::= <идентификатор>
[17] <имя последовательности>::= <идентификатор>
[18] <имя группы>::= <идентификатор>
[19] <имя правила>::= <идентификатор>
[20] <имя узла>::=
<идентификатор> (длина идентификатора не более 8 символов)
[21] <имя устройства>::= <идентификатор>

Синтаксические правила

1) Длина <имени таблицы> не более 66 символов.
2) Длина <имени представления> не более 66 символов.
3) Длина <имени столбца> не более 66 символов.
4) Если <идентификатор схемы> опущен, то по умолчанию используется схема пользователя, от имени которого подан запрос.
5) Два полных имени таблицы равны только тогда, когда совпадают их <имя схемы> и <имя таблицы>.
6) Два полных имени представления равны только тогда, когда совпадают их <имя схемы> и <имя представления>.
7) Ссылка на <псевдоним таблицы> допускается во всех вложенных подзапросах.

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

Типы данных

Функция

Определение типов данных.

Спецификация

[1] <тип данных>::=
<строковый тип>
| <байтовый тип>
| <UNICODE тип>
| <точный числовой тип>
| <приближенный числовой тип>
| <дата-время тип>
| <логический тип>
| <BLOB тип>
| <внешний файл>

[2] <строковый тип>::=
<строковый тип фиксированной длины>
| <строковый тип переменной длины>

[3] <строковый тип фиксированной длины>::=
{CHAR | CHARACTER} [(<длина>)]

[4] <строковый тип переменной длины>::=
{CHAR VARYING | CHARACTER VARYING | VARCHAR} (<длина>)

[5] <байтовый тип>::=
<байтовый тип фиксированной длины>
| <байтовый тип переменной длины>

[6] <байтовый тип фиксированной длины>::= BYTE | RAW [(<длина>)]

[7] <байтовый тип переменной длины>::=
{BYTE VARYING | VARBYTE} (<длина>)

[8] <UNICODE тип>::=
<UNICODE тип фиксированной длины>
| <UNICODE тип переменной длины>

[9] <UNICODE тип фиксированной длины>::=
{NATIONAL CHARACTER | NATIONAL CHAR | NCHAR} [(<длина>)]

[10] <UNICODE тип переменной длины>::=
{NATIONAL CHARACTER VARYING
| NATIONAL CHAR VARYING
| NCHAR VARYING
| NVARCHAR} (<длина>)

[11] <точный числовой тип>::=
{DECIMAL | DEC | NUMERIC | NUMBER} [(<точность> [,<масштаб>])]
| BIGINT
| {INTEGER | INT}
| SMALLINT

[12] <приближенный числовой тип>::=
{REAL
| {DOUBLE [PRECISION] | FLOAT}
| FLOAT (<точность>)}

[13] <дата-время тип>::= DATE

[14] <логический тип>::= BOOLEAN

[15] <BLOB тип>::= BLOB | LONG RAW

[16] <внешний файл>::= {EXTFILE | EF} [ROOT '<каталог>']

[17] <длина>::= <беззнаковое целое>

[18] <точность>::= <беззнаковое целое>

[19] <масштаб>::= <беззнаковое целое>
[20] <каталог>::= <символьный литерал>

Синтаксические правила

1) CHAR является синонимом CHARACTER.
2) VARCHAR является синонимом CHAR VARYING, CHARACTER VARYING.
3) VARBYTE является синонимом BYTE VARYING.
4) NCHAR является синонимом NATIONAL CHAR, NATIONAL CHARACTER.
5) NCHAR VARYING является синонимом NATIONAL CHAR VARYING, NATIONAL CHARACTER VARYING, NVARCHAR.
6) DEC, DECIMAL, NUMERIC и NUMBER являются синонимами.
7) INT является синонимом INTEGER.
8) DOUBLE является синонимом DOUBLE PRECISION.
9) Значение <длины> и <точности> должно быть больше 0.
10) Если <длина> не задана, по умолчанию принимается значение 1.
11) Если <масштаб> не задан, по умолчанию принимается значение 0.

Общие правила

1) Тип данных CHAR задает строку символов фиксированной длины (максимальная длина строки – 4000 байтов). Если реальное количество символов в строке меньше <длины>, то при записи в БД строка дополняется до заданной <длины> пробелами справа и при выборке возвращается с дополненными пробелами.

Примечание
Если ядро СУБД запущено с ключом /COMPATIBILITY=STANDARD, то функция length концевые нули учитывает, иначе игнорирует.

```
create table tab1(c char(50));
insert into tab1 values ('1234567890');
select c, length(c) from tab1;

|1234567890                                          |50 |
```

2) Тип данных VARCHAR задает строку символов переменной длины (максимальная длина строки – 4000 байтов). Если реальное количество символов в строке меньше <длины>, то строка при записи в БД или при выборке из БД не дополняется до заданной <длины> пробелами. Фактическая длина значений типа VARCHAR хранится в БД.

Примечание
Пробелы для типа VARCHAR и не дополняются и не усекаются, всегда хранится реальная длина с концевыми пробелами, если они были заданы при занесении значения, и без них, если они не были заданы.

```
create or replace table tab1(vc varchar(50));
insert into tab1 values ('123   ');
! три пробела в конце значения
select vc, length(vc) from tab1;
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
 VC
 --
|123                                                  |           6|
```

Примечание
В типах данных CHAR, VARCHAR длина и количество символов в строке эквивалентны только для однобайтовых кодировок. Например, если для столбца типа CHAR(N) или VARCHAR(N) задана многобайтовая кодировка (например, UTF-8), то в этот столбец всегда можно занести значение, состоящее не более чем из N байт в этой кодировке, однако не всегда можно занести значение, состоящее не более чем из N символов. Поэтому, если пользователь работает в кодировке UTF-8, но использует только основной набор символов и кириллицу, ему лучше выбрать для соответствующих столбцов одну из однобайтовых кириллических кодировок (CP866, CP1251 или KOI8-R). Тогда в поле CHAR(N) он всегда сможет занести кириллическую строку до N символов включительно.

3) Любой строковый тип данных может иметь значение длины 0.
4) Столбец как базовой таблицы, так и подзапроса не может иметь значение длины 0.
5) При выборке константы в SELECT-запросе столбец наследует тип выбираемого выражения. В случае пустой строки этот тип должен быть скорректирован, чтобы длина не была нулевой.
6) Тип данных BYTE задает строку байт фиксированной длины (максимальная длина байтовой строки – 4000 байтов). Если реальное количество байт в строке меньше <длины>, то строка при записи в БД или при выборке из БД дополняется до заданной <длины> двоичными нулями справа. Длина значений типа BYTE в БД не хранится.

Примечание
Если ядро СУБД запущено с ключом /COMPATIBILITY=STANDARD, то функция length концевые нули учитывает, иначе игнорирует.

```
create table tab1(b byte(20));
insert into tab1 values (hex('12ffca4527'));
select b, length(b) from tab1;

|12FFCA4527000000000000000000000000000000 |20 |

select trim(rawtohex(b), '0'), length(b) from tab1;

|12FFCA4527 |20 |
```

7) Тип данных VARBYTE задает строку байт переменной длины (максимальная длина байтовой строки – 4000 байтов). Если реальное количество байт в строке меньше <длины>, то строка не дополняется до заданной <длины> двоичными нулями.

Примечание
Независимо от режима запуска ядра СУБД (с ключом или без ключа /COMPATIBILITY=STANDARD) концевые нули всегда усекаются.

```
create table tab1(vb varbyte(20));
insert into tab1 values (hex('12ffca4527'));
```

```
select vb, length(vb) from tab1;

|12FFCA4527 |5 |
```

8) Строка символов/байт состоит из последовательности символов (байт) с кодами от 0 до 255.
9) Для символьных строк поддерживается преобразование к соответствующей кодировке (русская, латинская, верхний/нижний регистр), байтовые строки независимы от используемой кодовой страницы.

```
select lower('НПП "Релэкс"'), upper('НПП "Релэкс"');

|нпп "релэкс" |НПП "РЕЛЭКС" |
```

10) Длина строки, не содержащей символов/байт, равна `0`.
11) Строка символов/байт может иметь неопределенное (NULL) значение.

```
create table tab1(vb byte(20), c char (10));
insert into tab1 values (hex('12ffca4527'), null);
select count(*) from tab1 where c is null;

|1 |
```

12) Все строки символов сравниваются между собой в соответствии с весами, установленными для данной кодировки стандартами `UNICODE` (при этом короткая строка дополняется до длинной пробелами/нулями).
13) Результатом сравнения любой строки символов/байт со строкой с неопределенным (NULL) значением всегда будет `FALSE`.
14) Строки байт сравниваются в порядке кодов.
15) Строка символов/байт и число не сопоставимы между собой.

```
create table tab1(c char (10));
insert into tab1 values ('125');
```

Неправильная конструкция:

```
select * from tab1 where c > 100;
```

Правильные конструкции:

```
select * from tab1 where to_number(c) > 100;
select * from tab1 where c > cast 100 as char;
select * from tab1 where c > to_char(100);
```

16) Типы данных `NCHAR`, `NCHAR VARYING` задают представление данных в формате `UNICODE`, в котором для представления символа используется шестнадцать бит – два байта данных, обеспечивая, таким образом, уникальные коды для 65536 символов. Поскольку `UNICODE` может представить такое количество различных символов, то он обеспечивает стандартизированные коды для символов, используемых в большинстве языков мира, и, кроме того, может представлять разнообразные дополнительные графические символы (математические, фонетические, торговые и пр.).
17) Тип данных `NCHAR` задает UNICODE-строку символов фиксированной длины (максимальная длина Unicode-строки – 4000 байтов, 2000 Unicode-символа). Если реальное количество символов в строке меньше <длины>, то при записи в БД строка дополняется до заданной <длины> пробелами справа. Длина значений типа `NCHAR`

в БД не хранится. При выборке из БД дополнение пробелами до заданной длины не выполняется.

```
create table tab1(n nchar (10));
insert into tab1 values (hex('125745fa768877ad'));
select n, length(n) from tab1;

|####  |4 |
```

18) Тип данных NCHAR VARYING задает UNICODE-строку символов переменной длины (максимальная длина UNICODE-строки – 4000 байтов, 2000 UNICODE-символа). Если реальное количество символов в строке меньше <длины>, то при записи в БД или при выборке из БД строка не дополняется до заданной <длины> пробелами. Фактическая длина значений типа NCHAR VARCHAR хранится в БД.

```
create table tab1(vn nchar varying(10));
insert into tab1 values (hex('125745fa768877ad'));
select vn, length(vn) from tab1;

|####  |4 |
```

19) Для символьных данных, представленных в UNICODE, поддерживаются преобразования к верхнему/нижнему регистру.
20) При выполнении операции сравнения данные типа VARCHAR, VARBYTE, NCHAR VARYING дополняются, при необходимости, пробелами (двоичными нулями) для выравнивания длин сравниваемых операндов.
21) При сравнении данных типа CHAR, VARCHAR, NCHAR, NCHAR VARYING, представленных в разных кодировках, выполняется преобразование к одной кодировке.
22) Тип данных DECIMAL задает число с фиксированной точкой (целое или дробное) с максимальным масштабом (количество цифр справа от десятичной точки), равным 10, и точностью (максимальное число значащих цифр), равной 30.

```
create table tab1(d1 decimal, d2 numeric);
insert into tab1 values (1, 1.);
insert into tab1 values (.076, null);
insert into tab1 values (99999999999999999999.9999999999,null);
insert into tab1 values (-654.0, +67.004);
select * from tab1;

|1.0                                          |1.0          |
|0.076                                        |NULL         |
|99999999999999999999.9999999999              |NULL         |
|-654.0                                       |67.004       |
```

23) Если <точность> и <масштаб> не заданы, по умолчанию принимается: <точность> – 30, <масштаб> – 10.
24) Если задана только точность, то <масштаб> равен 0.
25) Цифры, которые выходят за размер масштаба, все равно хранятся в значении, но не выдаются при выдаче значения (округляются). То есть в случае масштаба 0 в БД будут храниться дробные числа, но при выводе они будут округляться до целого.

```
create table t_d(d decimal(4,2));
```

```
! Значения укладываются в точность и масштаб
insert into t_d values (11.11);
insert into t_d values (-22.22);

! Превышено число цифр до запятой – ошибка
insert into t_d values (111.11);
insert into t_d values (-222.22);

! Превышено число цифр после запятой – усечение при выводе
insert into t_d values (33.333);
insert into t_d values (-44.444);

select * from t_d;

| 11.11|
|-22.22|
| 33.33|
|-44.44|
```

Примечание
Тип данных BIGINT задает целое число с точностью 19 цифр и нулевым количеством цифр справа от точки – целые числа в диапазоне от -9 223 372 036 854 775 807 до +9 223 372 036 854 775 807.

```
create table tab1(bi bigint);
```

26) Тип данных INT задает целое число с точностью 10 цифр и нулевым количеством цифр справа от точки – целые числа в диапазоне от -2 147 483 647 до +2 147 483 647.

```
create table tab1(i1 int, i2 integer);
```

27) Тип данных SMALLINT задает целое число с точностью 5 цифр и нулевым количеством цифр справа от точки – целые числа в диапазоне от -32 767 до +32 767.

```
create table tab1(si smallint);
```

28) Тип данных REAL задает тип данных приближенного числа с точностью, равной 6.

```
create table tab1(r real);
insert into tab1 values (1.);
insert into tab1 values (-67.e+6);
insert into tab1 values (+00002.E-12);

select * from tab1;

|1         |
|-6.7e+07  |
|2e-12     |
```

29) Тип данных DOUBLE задает тип данных приближенного числа с точностью, равной 15.

```
create table tab1(d1 double, d2 double precision);
```

```
insert into tab1 values (1., null);
insert into tab1 values (-67.e+6, 877655.0000000078341);
insert into tab1 values (+02.E-34, 5643);

select * from tab1;

|1         |NULL              |
|-67000000 |877655.000000008  |
|2e-34     |5643              |
```

30) Максимальное значение по модулю для типа REAL 3.402823466e+38, для типа DOUBLE 1.7976931348623158e+308.

31) Для значений типов REAL и DOUBLE не рекомендуется использовать точное сравнение.
Не рекомендуется:
столбец = константа
Рекомендуется:
столбец BETWEEN (константа - точность) AND (константа + точность)

32) В значениях типа DATE дата совмещена со временем. Время в дате считается с точностью до тиков (тик – 1/100 часть секунды).

33) В СУБД ЛИНТЕР формат хранения значений типа DATE позволяет представлять даты в диапазоне от нулевой даты (00.00.0000) до 31.12.9999.

34) Значения типа DATE допускают операции сравнения между собой и арифметические операции прибавления (вычитания) к полной дате интервала даты (часов, минут, секунд, дней, месяцев или лет).

35) Форматы представления типа данных приведены в пункте <литералы>.

```
create table tab1 (dt date);
insert into tab1 values ('28.04.03');
insert into tab1 values ('28.04.03:12:56:45.99');
insert into tab1 values (to_date('10-sep-50'));
insert into tab1 values (to_date('12.18.1878', 'mm.dd. yyyy'));

select * from tab1;

|28.04.2003:00:00:00.00  |
|28.04.2003:12:56:45.99  |
|10.09.1950:00:00:00.00  |
|18.12.1878:00:00:00.00  |
```

36) Логический тип данных имеет два значения: TRUE (истина) и FALSE (ложь).

```
create table tab1 (l boolean);
insert into tab1 values (true);
insert into tab1 values ('FALSE');
insert into tab1 values (null);
select * from tab1;

|TRUE  |
|FALSE |
```

```
|NULL   |
```

37) Тип данных `BLOB` задает неструктурированные данные объемом до 2 Гбайт, предназначенные для хранения текстовой, графической и мультимедийной информации (объемные фрагменты текста, музыка, анимация, видеоизображения, графика и т.п.). Для столбца типа `BLOB` NULL-значение недопустимо (допускается только BLOB-значение нулевой длины).

```
create table tab1 ("Графика" blob, "Музыка" blob, VIDEO blob);
```

38) Тип данных `EXTFILE` – символьная строка `char(511)`, содержащая ссылку на локальный файл (то есть файл, расположенный на локальном или сетевом диске и доступный файловой системе компьютера, на котором функционирует СУБД ЛИНТЕР). Значение типа данных `EXTFILE` нельзя записать и/или прочитать явно. Для этого должны использоваться средства полнотекстового доступа к данным (см. документ «Полнотекстовый поиск в базе данных», приложение «Расширения SQL»).

```
create table tab1 ("Слова" extfile, "Музыка" extfile root 'd:
\Program Files\Music\shanson');
insert into tab1("Музыка", "Слова") values (ef('music.doc'),
 ef('text.doc'));
```

Спецификация значения

Функция

Определение значений параметров или предопределенных переменных.

Спецификация

[1] <спецификация значения>::=
<литерал>
| USER
| LINTER_USER_ID
| EFFECTIVE_USER
| EFFECTIVE_USER_ID
| LINTER_SYSTEM_USER
| SYSDATE
| {LOCALTIMESTAMP| CURRENT_TIMESTAMP}
| NOW
| {LOCALTIME |CURRENT_TIME}
| ROWID
| ROWTIME
| DBROWTIME
| ROWNUM
| LAST_ROWID
| LAST_AUTOINC
| LINTER_NAME_LENGTH
| TRIGGER_INFO_SIZE
| AUD_OBJ_NAME_LEN
| PROC_INFO_SIZE
| PROC_PAR_NAME_LEN
| SESSIONID
| USER_TIME
| SYSTEM_TIME
| CURRENT_DATE
| CURRENT_SCHEMA

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

Синтаксические правила

1) <Спецификация значения> задает значение, которое не содержится среди столбцов списка таблиц SQL-запроса.

```
select user, count(*) from AUTO;

| SYSTEM| 1000|
```

```
select rownum, sysdate;

|1 |31.03.2003:11:33:53.00 |
```

```
select 146;

|146|
```

2) Типом данных USER, EFFECTIVE_USER, LINTER_SYSTEM_USER является CHAR(66).
3) Типом данных SYSDATE, CURRENT_TIMESTAMP, NOW, ROWTIME, LOCALTIME, LOCALTIMESTAMP, CURRENT_DATE, CURRENT_TIME является DATE.
4) Типом данных LINTER_USER_ID, EFFECTIVE_USER_ID, LAST_ROWID, LAST_AUTOINC, ROWNUM, ROWID, LINTER_NAME_LENGTH, TRIGGER_INFO_SIZE, AUD_OBJ_NAME_LEN, PROC_INFO_SIZE, PROC_PAR_NAME_LEN является INT.
5) Типом данных DBROWTIME, SESSIONID является BIGINT.
6) Значения NOW и SYSDATE являются синонимами.

```
select sysdate, now;

|14.12.2007:09:14:15.00|14.12.2007:09:14:15.00|
```

7) Значения CURRENT_TIMESTAMP и LOCALTIMESTAMP являются синонимами.

```
select CURRENT_TIMESTAMP
  union all
select LOCALTIMESTAMP;

|28.08.2012:15:39:39.51|
|28.08.2012:15:39:39.51|
```

8) Значения CURRENT_TIME и LOCALTIME являются синонимами.

```
select CURRENT_TIME
  union all
select LOCALTIME;

|00.00.0000:15:49:46.18|
|00.00.0000:15:49:46.18|
```

Общие правила

1) Значение USER равно <имени пользователя> (символьному идентификатору) пользователя БД в текущем сеансе работы (в данном соединении с БД) и закрепленное за родительским каналом.

```
! Является ли текущий пользователь владельцем таблиц AUTO.
select case $$$s34 when user then 'Да' else 'Нет' end
  from $$$sysrl, $$$usr
 where $$$s12=$$$s31 and $$$s13='AUTO';

|Да |
```

2) Запрос

```
select user;
```

поданный из хранимой процедуры от имени владельца процедуры, возвращает не имя владельца процедуры, а имя пользователя, который запустил ее на выполнение.

3) Значение `LINTER_USER_ID` равно числовому идентификатору пользователя БД в текущем сеансе работы (в данном соединении с БД) и закреплено за родительским каналом (столбец `$$$S31` в системной таблице `$$$USR`).

```
select $$$s31 from $$$usr where $$$s34=(select user);
```

4) Значение `EFFECTIVE_USER` возвращает `<имя пользователя>` (символьный идентификатор) от имени которого выполнятся запрос.

Примечание
Поддерживается со сборки 6.0.17.92.

5) Запрос

```
select EFFECTIVE_USER;
```

поданный из хранимой процедуры, выполняемой от имени владельца, возвращает имя владельца процедуры, от имени которого выполняются все запросы в данной процедуре.

6) Значение `EFFECTIVE_USER_ID` возвращает числовой идентификатор пользователя БД от имени которого выполнятся запрос.

Примечание
Поддерживается со сборки 6.0.17.92.

7) Запрос

```
select EFFECTIVE_USER_ID;
```

поданный из хранимой процедуры, выполняемой от имени владельца, возвращает числовой идентификатор владельца процедуры, от имени которого выполняются все запросы в данной процедуре.

8) Значение `LINTER_SYSTEM_USER` соответствует `<имени создателя БД>` (идентификатору).

9) Получить явное значение `LINTER_SYSTEM_USER` с помощью SQL-запроса невозможно, то есть запросы типа

```
select LINTER_SYSTEM_USER;
```

не обрабатываются.

Извлечь это имя можно только с помощью обращения к системной таблице пользователей БД (rowid=1 – это системный идентификатор создателя БД):

```
select $$$S34 from LINTER_SYSTEM_USER.$$$USR where rowid=1;
```

Запрос к системной таблице с явным указанием имени ее создателя:

```
select count(*) from "SYSTEM"."$$$ATTRI";
select count(*) from SYSTEM.$$$attri;
```

Запрос к системной таблице с использованием псевдопеременной:

```
select count(*) from LINTER_SYSTEM_USER.$$$ATTRI;
```

10) Значение SYSDATE равно текущим дате и времени по Гринвичу (нулевому часовому поясу).

```
select 'На ' ||
  to_char(sysdate,'dd.mm.yyyy:hh:mi:ss') ||
  ' число строк в таблице AUTO равно ' ||
  to_char(count(*),'9999')
  from auto;

|На 23.04.2003:17:08:05 число строк в таблице AUTO равно 1000 |
```

11) Значения CURRENT_TIMESTAMP и LOCALTIMESTAMP равны текущей локальной дате, установленной на ЛИНТЕР-сервере. Они эквивалентны вызову функции to_localtime(sysdate).

```
select sysdate
  union all
select CURRENT_TIMESTAMP
  union all
select LOCALTIMESTAMP
  union all
select to_localtime(sysdate);

|28.08.2012:11:45:12.28|
|28.08.2012:15:45:12.28|
|28.08.2012:15:45:12.28|
|28.08.2012:15:45:12.28|
```

12) Значения CURRENT_TIME и LOCALTIME равны текущему локальному времени, установленному на ЛИНТЕР-сервере (дата имеет нулевое значение).

```
select sysdate
  union all
select CURRENT_TIME
  union all
select LOCALTIME
  union all
select to_localtime(sysdate);

|28.08.2012:11:56:35.62|
|00.00.0000:15:56:35.62|
|00.00.0000:15:56:35.62|
```

```
|28.08.2012:15:56:35.62|
```

13) Значение CURRENT_DATE равно текущей локальной дате, установленной на ЛИНТЕР-сервере (время имеет нулевое значение).

```
select sysdate
  union all
select CURRENT_TIME
  union all
select CURRENT_DATE;

|28.08.2012:12:00:17.10|
|00.00.0000:16:00:17.10|
|28.08.2012:00:00:00.00|
```

14) Значение CURRENT_SCHEMA предоставляет текущую схему пользователя.

```
select current_schema;

|SYSTEM                                                          |
```

15) ROWID возвращает системный номер записи (уникальное значение). Максимальное значение – 0x01FFFFFF.

16) ROWID уникально определяет запись в таблице в каждый момент времени (то есть в таблице не может быть одновременно двух записей с одинаковым ROWID). Но если удалить запись с некоторым ROWID, то этот ROWID может быть впоследствии использован повторно СУБД ЛИНТЕР для новой записи. ROWID записи сохраняется неизменным на всем протяжении ее существования в БД и не меняется при операциях UPDATE и UPDATE CURRENT над этой записью.

```
create table SAMPLE (c char);
insert into SAMPLE values('a');
insert into SAMPLE values('b');
insert into SAMPLE values('c');
insert into SAMPLE values('d');
select rowid, c from SAMPLE;

|1|a|
|2|b|
|3|c|
|4|d|

delete from SAMPLE where rowid=2;
update SAMPLE set c='F' where rowid=4;
select rowid, c from SAMPLE;

|1|a|
|3|c|
|4|F|
```

17) ROWTIME возвращает дату/время последнего изменения строки в таблице. Это поле доступно только для чтения, принудительно установить собственное время внесения строки в таблицу невозможно.

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

18) DBROWTIME возвращает время последнего изменения строки в виде значения BIGINT, которое представляет время в виде количества интервалов времени, прошедших с 01.01.1990:00:00:00.00 (один интервал времени равен 1/65536 части секунды). В отличие от ROWTIME, которое возвращает округляемое значение времени в виде DATE (то есть оно может не быть уникальным), псевдостолбец DBROWTIME всегда предоставляет уникальное значение времени.

```
select dbrowtime,rowtime from auto;
select min(dbrowtime), max(dbrowtime) from auto;
```

19) Значение ROWTIME изменяется при операциях UPDATE, UPDATE CURRENT над записью, при добавлении и удалении BLOB, а для записей системных таблиц – также и при операциях, специфических для этих таблиц. Значение ROWTIME хранится в БД вместе с ROWID и содержимым записи. В СУБД ЛИНТЕР оно всегда запоминается и выдается по Гринвичу (т.к. используется в операциях журнала и не должно меняться при переносе БД в другой часовой пояс).

```
! Удалить записи с истекшим сроком хранения (за предыдущий год)
delete from tst
  where to_number(to_char(rowtime,'yyyy')) <=
        to_number(to_char(sysdate,'yyyy'))-1;
```

20) ROWNUM возвращает порядковый номер записи в выборке данных. Тип значения – INT. ROWNUM следует указывать только в SELECT-списке (как в основном запросе, так и в подзапросах). ROWNUM вычисляется позже проверки условий и сортировки результата.

```
select rownum, rowid from tst1
  union
select rownum, rowid from tst2;

|1|100  |
|2|101  |
|3|105  |
|4|1000 |
|5|1001 |
```

21) LAST_ROWID возвращает значение последнего ROWID, который был обработан в текущем сеансе при выполнении запросов INSERT, UPDATE, DELETE. Если от одного канала открыто несколько курсоров, то значение LAST_ROWID обновляется одновременно для основного канала и всех этих курсоров. Если же приложение открыло несколько независимых каналов, то значения LAST_ROWID для всех них будут независимыми.

```
/* начало сеанса */
select rowid, last_rowid from test;

|1|0|
|2|0|

insert into test (i,blb) values(100,NULL);
select rowid, last_rowid from test;

|1|3|
|2|3|
```

```
|3|3|
```

```
update bank set name ='АКБ Промбанк' where rowid=204;
select rowid, last_rowid from test;
```

```
|1|204|
|2|204|
|3|204|
```

22) LAST_AUTOINC возвращает последнее из значений AUTOINC-столбцов, которое было добавлено в какую-либо таблицу в текущем сеансе. Аналогично LAST_ROWID: если от одного канала открыто несколько курсоров, то значение LAST_AUTOINC обновляется одновременно для основного канала и всех этих курсоров. Если же приложение открыло несколько независимых каналов, то значения LAST_AUTOINC для всех них будут независимыми.
23) LAST_AUTOINC возвращает значение типа INT. Если необходимо вернуть последнее вставленное в AUTOINC значение как BIGINT, следует использовать конструкцию CAST LAST_AUTOINC AS BIGINT.
24) Значение LINTER_NAME_LENGTH возвращает максимально возможную длину идентификатора объекта БД.

```
select LINTER_NAME_LENGTH;
```

```
|66 |
```

25) Значение TRIGGER_INFO_SIZE возвращает максимально возможную длину описания триггера (столбец $$$INFO системной таблицы $$$TRIG).

```
select TRIGGER_INFO_SIZE;
```

```
|320 |
```

26) Значение AUD_OBJ_NAME_LEN возвращает максимально возможную длину имени объекта контроля (Audit).

```
select AUD_OBJ_NAME_LEN;
```

```
|134 |
```

27) Значение PROC_INFO_SIZE возвращает максимально возможную длину описания хранимой процедуры (столбец $$$INFO системной таблицы $$$PROC).

```
select proc_info_size;
```

```
|32 |
```

28) Значение PROC_PAR_NAME_LEN возвращает максимально возможную длину имен параметров хранимой процедуры.

```
select PROC_PAR_NAME_LEN
```

```
|66 |
```

29) PROC_PAR_NAME_LEN возвращает длину поля NAME в системной таблице $$$PROC.
30) Значение SESSIONID является идентификатором текущей сессии пользователя БД. Идентификатор сессии создается ядром СУБД при выполнении пользователем соединения с БД (открытия канала) и остается неизменным для всех подканалов

и курсоров, порождаемых при выполнении пользовательских SQL-запросов, триггеров и хранимых процедур по этому соединению.

```
select channel, last_request, "SESSIONID"
  from linter_system_user.$$$chan a
  where a."SESSIONID" = SESSIONID;

|channel |Last_request |Sessionid      |
|3       |204          |34333001939289 |
|4       |SELECT       |34333001939289 |
 …
```

31) `USER_TIME` возвращает время (в миллисекундах) работы ядра СУБД ЛИНТЕР. Значение берется из «виртуальной» таблицы `$$$SYSINFO`.

```
select user_time from $$$sysinfo;
```

32) `SYSTEM_TIME` возвращает время (в миллисекундах), затраченное ядром СУБД ЛИНТЕР на операции ввода-вывода. Значение берется из «виртуальной» таблицы `$$$SYSINFO`.

```
select system_time from $$$sysinfo;
```

Спецификация таблицы

Функция

Определение ссылки на таблицу или представление.

Спецификация

[1] <спецификация таблицы>::= [<имя схемы>.]<описатель таблицы>
[2] <описатель таблицы>::=
<имя таблицы> [[AS] <псевдоним таблицы>]
[3] <имя таблицы>::= <идентификатор>
[4] <псевдоним таблицы>::= <идентификатор>

Синтаксические правила

1) <Имя таблицы> должно быть уникальным среди имен таблиц данного пользователя БД.

Это разные таблицы:

```
SYSTEM.AUTO
SYS.AUTO
"Систем".AUTO
```

2) <Имя таблицы> может ссылаться на базовую таблицу или представление.

3) <Псевдоним таблицы> используется лишь в контексте SQL-оператора, и поэтому должен быть уникальным только в данном SQL-операторе. <Псевдоним таблицы> используется, как правило, для альтернативного обозначения таблиц с длинными именами или для связывания в одном SQL-запросе одной и той же таблицы.

Эти конструкции эквивалентны:

```
select count(a.id) from "Штатное расписание" as a
```

```
 where a."Отдел"=3
   and a."Должность"='Инженер';

select count("Штатное расписание".id)
  from "Штатное расписание"
 where "Штатное расписание"."Отдел"=3
   and "Штатное расписание"."Должность"='Инженер';
```

4) Псевдоним используется для связывания одной и той же таблицы и в качестве комментария к SQL-запросу:

```
select t2.id "На эти документы есть ссылки",
       t2.name,
       t2.status "Статус найденных документов",
       t1.id "Эти документы ссылаются на другие документы",
       t1.s
  from DOC t1, DOC t2
 where t1.status like '%вв%'
   and instr(t1.s, t2.id)<>0
 order by 1, 3 DESC;
```

Спецификация столбца

Функция

Определение ссылки на столбец.

Спецификация

[1] <спецификация столбца>::=
[<имя схемы>.][<имя таблицы> | <псевдоним таблицы>]
<описатель столбца> [, <описатель столбца> [, ...]]
[2] <описатель столбца>::=
<имя столбца> [[AS] <псевдоним столбца>]
[3] <имя столбца>::= <идентификатор>
[4] <псевдоним столбца>::= <идентификатор>

Синтаксические правила

1) <Спецификация столбца> указывает на именованный столбец. Смысл указателя на столбец зависит от контекста.

```
select personid,make from auto;
select SYSTEM.SAMPLE."Наименование" from SAMPLE;
select a.name from "Справочник банков" as a;
```

2) <Имя таблицы> может ссылаться на базовую таблицу или представление.

3) Если <спецификация столбца> включает <имя таблицы>, то <спецификация столбца> должна использоваться внутри области видимости указанных в нем имен таблиц (представлений) или их синонимов. Если найдено более одного такого <имени таблицы>, то берется область видимости с наибольшей локализацией.

```
select tab1.col1, tab2.col1 from tab1,tab2
  where tab1.id=tab2.id and tab2.id <> 3;
```


© Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

4) Если <спецификация столбца> не включает имени таблицы (представления) или их синонимов, то <спецификация столбца> должна быть в области видимости одной или более таблиц (представлений), у которых есть такой столбец, причем в соответствующей области видимости <спецификация столбца> должна быть уникальной.

```
select personid, make from auto
  where length(make) = (select max(length(make)) from auto);
```

5) <Псевдоним столбца> может совпадать с именем встроенной в SQL СУБД ЛИНТЕР функции.

```
select make as length from auto where rowid < 3;
```

Общие правила

1) Значением <спецификации столбца> является значение столбца <имя столбца> в данной строке таблицы <имя таблицы> или псевдонима <псевдоним таблицы>.

Значимое выражение

Функция

Определение значения.

Спецификация

[1] <значимое выражение>::=
<числовое выражение>
| <строковое выражение>
| <дата-время выражение>
| <логическое выражение>

[2] <числовое выражение>::=
<член выражения> <арифметический оператор>
{+ | -} <член выражения>

[3] <арифметический оператор>::= {+ | - | * | / | %}

[4] <член выражения>::=
<элемент выражения>
| <член выражения> {* | /} <элемент выражения>

[5] <элемент выражения>::= [+ | -] <значение>

[6] <строковое выражение>::=
<символьное выражение> | <байтовое выражение>

[7] <символьное выражение>::=
<член выражения> | <символьная конкатенация>

[8] <символьная конкатенация>::=
<символьное выражение> {|| | +}<значение>

[9] <байтовое выражение>::=
<член выражения> | <байтовая конкатенация>

[10] <байтовая конкатенация>::=
<символьное выражение> {|| | +}<значение>

[11] <дата-время выражение>::= <значение>

[12] <значение>::=
<спецификация значения>
| {<имя столбца> | <псевдоним столбца>}
| <агрегатная функция>
| <значение последовательности>
| <подзапрос>

| <спецификация типа>
| <спецификация значения по условию>
| <SQL-параметр>
| (<значимое выражение>)

Синтаксические правила

1) Первым символом лексемы, следующей за унарным знаком операции, не может быть знак плюс или минус.
2) Если тип <значения> – символьная/байтовая строка, то <значимое выражение> может включать только операцию конкатенации. Тип результата – также символьная/байтовая строка.

   Конструкция

```
select 'Фирма: ' ||  make || ' модель: ' || model from auto;

|Фирма: FORD модель: MERCURY COMET GT V8|
 …
```

   эквивалентна:

```
select 'Фирма: ' +  make + ' модель: ' + model from auto;
```

3) Использование в <значимом выражении> типа данных BLOB не допускается.
4) При использовании в <значимом выражении> типа данных DATE допускаются операции добавления (вычитания) интервалов даты.

```
select sysdate + to_date('01','mm');
select sysdate,
  (sysdate + to_date('01','mm')) - to_date('05','yy');

|24.04.2003:14:26:51.00 |24.05.1998:14:26:51.00 |
```

5) Если у обоих операндов точный числовой тип, то и результат будет иметь точный числовой тип, определяемый следующим образом:
   - если один из операндов имеет тип DECIMAL, то тип результата – DECIMAL;
   - если оба операнда целые, то результат будет иметь целый тип;
   - если один из операндов имеет тип INT, то тип результата – INT;
   - если один из операндов имеет тип BIGINT, то тип результата – BIGINT;
   - если оба операнда типа SMALLINT, тип результата – SMALLINT.
6) Если <значимое выражение> является <числовым выражением>, то в операциях сравнения и присвоения допускается использовать строковый литерал без оператора преобразования.

```
create or replace table tst
  (si smallint,i int, bi bigint, d double, nm numeric);
! Эти запросы эквивалентны:
insert into tst (si,i,bi,d,nm)
  values (12, 3658, 5688854, 45.e+3, 56.099);
insert into tst (si,i,bi,d,nm)
  values ('12', '3658', '5688854', '45.e+3', '56.099');
```


© Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
! Эти запросы эквивалентны:
select count(*) from auto where year='70';
select count(*) from auto where year=70;
select count(*) from auto where year= cast '70' as int;
```

7) В случае арифметических действий над аргументами типа DECIMAL типом результата берется DECIMAL(30,10).
8) Если тип одного из операндов приближенный, то тип результата тоже будет приближенным. Типом данных результата будет DOUBLE, если хотя бы один из операндов имеет тип DOUBLE.
9) Если <значение> равно NULL, то результатом <значимого выражения> тоже будет NULL.

```
create table tab1 (i1 int, i2 smallint, r real);
insert into tab1 values (1, 5, 56.8);
insert into tab1 values (1, NULL, 56.8);
select * from tab1;

|1|5   |56.7999992|
|1|NULL|56.7999992|

select (i1+i2)/r from tab1;

|0.1056338 |
|NULL      |
```

10) Если операция не указана, то результат <значимого выражения> берется из <значения>.
11) Когда <значимое выражение> применяется к строке таблицы, то указание столбца этой таблицы есть его значение в этой строке.
12) Двухместные арифметические знаки «+», «-», «*», «/» обозначают сложение, вычитание, умножение, деление соответственно. Если делитель равен 0, будет зафиксирована исключительная ситуация.
13) Унарный плюс не меняет операнд. Унарный минус меняет знак ненулевого числа на противоположный.
14) Если тип результата – точный числовой, то возможны следующие варианты:
    - если операция – не деление, и результат ее точно не представляется значением результирующего типа, то будет зафиксирована исключительная ситуация;
    - если операция – деление, и результат ее представляется значением результирующего типа с потерей одной или более значащих цифр, то будет зафиксирована исключительная ситуация.
15) Выражения внутри круглых скобок вычисляются первыми, и, если порядок вычисления не задан круглыми скобками, унарные операции выполняются перед умножением/делением, умножение/деление – перед сложением/вычитанием, операции одного уровня выполняются слева направо.

```
create table tab1 (i1 int, i2 int, i3 int, i4 int);
insert into tab1 values (1,2,3,4);
select * from tab1;

|1 |2 |3 |4 |
```

```
select (i1 * i2) + (i3 * i4) from tab1;

|14|
select i1 * i2 + i3 * i4 from tab1;

|14|
select i1 * (i2 + i3) * i4 from tab1;

|20|
select ((length(make) + year) * (length(model) - sqrt(4)))/100
  from auto;
```

16) Глубина стека для обработки выражений 50.

17) При обработке числовой константы, если не удалось ее преобразовать к типу DECIMAL, делается попытка преобразовать ее к типу DOUBLE.

Общие правила

1) Если используются арифметические операторы, то результат <значимого выражения> получается путем применения операторов к значениям операндов.
2) Если какой-либо из операндов равен NULL, результат NULL.
3) Арифметические операторы применимы только к числовым операндам.
4) При невозможности вычислить <значимое выражение> (например, деление на нуль или переполнение числа) фиксируется исключительная ситуация.
5) Если арифметическая ошибка выявляется при выполнении агрегатной функции, то фиксируется исключительная ситуация.
6) Если эти алгоритмы обработки являются неприемлемыми при обработке данных, необходимо принять дополнительные меры, такие как:
   - включить проверку на нуль в операциях деления и установить желаемое (по смыслу) значение результата в таких случаях;

```
create table tab1 (i1 dec, i2 dec not null);
insert into tab1 values (1,2);
insert into tab1 values (1,0);
select * from tab1;

|1.0 |2.0 |
|1.0 |0.0 |


select case i2 when 0 then 'Деление на нуль'
               else cast i1/i2 as char
       end
  from tab1;

|0.5|
```

Деление на нуль

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

или

```
select case i2 when 0 then null
               else i1/i2
       end
  from tab1;
```

- добавить дополнительные предикаты для управления NULL-значениями (подобно CHECK или атрибут NOT NULL при определении свойств столбца таблицы), например:

```
check (c1*c2 is not NULL and c1*c2>5000);
```

Примеры

Значимое выражение – <спецификация значения>:

```
select sysdate;
select * from person where name=user;
select rownum, count(*) from person group by name;
```

Значимое выражение – <имя столбца>:

```
select distinct make from auto;
```

Значимое выражение – <агрегатная функция>:

```
select default( make) from auto;
select count(*) from person where name='ADKINSON';
```

Значимое выражение – <значение последовательности>:

```
select ("my_seq".nextval +1000)/34;
select * from auto
 where personid = (select "my_seq".currval);
```

Значимое выражение – (<значимое выражение>):

```
select (default( make)) from auto;
select (sysdate);
```

Значимое выражение – <подзапрос>:

```
select model from auto
where auto.personid=
  (select personid from person
    where auto.personid=person.personid and name='ANDERSON');

|OLDSMOBILE 98|
|PLUS 8       |
 …
```

Значимое выражение – <спецификация типа>:

```
select count(*) from auto
```

```
 where cast make as char(1) in ( 'A', 'B', 'C')
 group by cast make as char(1);

|98   |
|10   |
|175  |
```

Значимое выражение – <спецификация значения по условию>:

```
select case year when 70 then 'Год выпуска 1970'
                 else 'Год выпуска 1971'
       end,
       count(*)
  from auto group by year;

|Год выпуска 1970 |465 |
|Год выпуска 1971 |535 |
```

Значимое выражение – <числовое выражение>:

```
select make, mod(length(make), 3), year+1900
  from auto fetch first 2;

|FORD   |1 |1971 |
|ALPINE |0 |1970 |
```

Значимое выражение – <строковое выражение>:

```
select
  case year when 70 then 'Год выпуска 1970'
            else 'Год выпуска 1971' end
  || ' : ' || to_char( count(*), '999')
  from auto group by year;

|Год выпуска 1970 : 465 |
|Год выпуска 1971 : 535 |
```

Значимое выражение – <дата-время выражение>:

```
select sysdate + to_date('17', 'dd');
```

Значимое выражение – <логическое выражение>:

```
select count(*) from auto
 where (length(make)-10)>0 or  (year-70)=0
 group by length(make)-10 fetch first 3;

|3  |
|14 |
|74 |
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

Побитовые выражения

Спецификация

[1] <битовое выражение>::=
<числовое выражение> <побитовая операция>
<числовое выражение>
[2] <битовое выражение>::=
<побитовое «И»> | <побитовое «ИЛИ»>
| <побитовое «НЕ»> | <побитовое «ИСКЛЮЧАЮЩЕЕ ИЛИ»>
[3] <побитовое «И»>::= &
[4] <побитовое «ИЛИ»>::= |
[5] <побитовое «НЕ»>::= ~
[6] <побитовое «ИСКЛЮЧАЮЩЕЕ ИЛИ»>::= ^

Синтаксические правила

1) Допустимые типы <числового выражения> – BYTE, SMALLINT, INT и BIGINT.
2) Результаты выполнения операции <побитовое «И»>:

<побитовое «И»>	0	1
0	0	0
1	0	1

3) Результаты выполнения операции <побитовое «ИЛИ»>:

<побитовое «ИЛИ»>	0	1
0	0	1
1	1	1

4) Результаты выполнения операции <побитовое «НЕ»>:

<числовое выражение>	<побитовое «НЕ»>
0	1
1	0

5) Результаты выполнения операции <побитовое «ИСКЛЮЧАЮЩЕЕ ИЛИ»>:

<побитовое «ИСКЛЮЧАЮЩЕЕ ИЛИ»>	0	1
0	0	1
1	1	0

Пример

```
create or replace table test_tab (i int);

insert into test_tab values (7);
insert into test_tab values (0);
insert into test_tab values (-7);

select i, i|0x11 as "OR", i&0x11 as "AND",
       ~i as "NOT", i^0x11 as "XOR" from test_tab;
```

```
 I|OR|AND|NOT|XOR
-----------------
 7|23|  1| -8| 22
 0|17|  0| -1| 17
-7|-7| 17|  6|-24
```

Побитовая операция AND

Функция

Получить значение побитовой операции AND двух операндов.

Спецификация

[1] <синтаксис>::=
BITAND(<числовое выражение 1>, <числовое выражение 2>)

Синтаксические правила

1) <Числовое выражение 1> и <числовое выражение 2> должны иметь типы данных SMALLINT, INTEGER или BIGINT.

Возвращаемое значение

Результат побитовой операции AND над указанными операндами.

Примечание
Функцию BITAND(x,y) можно заменить выражением x&y.

Примеры

```
SELECT BITAND(5,3);
! Результат: 1

SELECT BITAND(15,7);
! Результат: 7

SELECT BITAND(5,2);
! Результат: 0

SELECT BITAND(5,0);
! Результат: 0

SELECT BITAND(6,2);
! Результат: 2
```

Логическое выражение

Функция

Определение логического выражения.

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

Спецификация

[1] <логическое выражение>::=
<логический терм> | <логическое выражение> OR <логический терм>
[2] <логический терм>::=
<логический множитель> | <логический терм> AND <логический множитель>
[3] <логический множитель>::=
[NOT] <первичное логическое выражение>
IS [NOT] {TRUE | FALSE}
[4] <первичное логическое выражение>::=
<предикат> | (<логическое выражение>)

Общие правила

1) Результатом <логического выражения> будет результат применения логических операций к тем условиям, которые получены при применении заданного <предиката> к данной строке таблицы или к данной группе в сгруппированной таблице. Если логические операции не заданы, то результатом <логического выражения> будет результат заданного <предиката>.

```
select make from auto
 where not bodytype ='SEDAN'
   and (cylnders>6 or weight between 3000 and 4000)
    or (color like 'B%' and not year=70);
```

2) Результатом NOT (TRUE) является значение FALSE, результатом NOT (FALSE) – значение TRUE, результатом NOT (UNKNOWN) – значение UNKNOWN. Операции AND и OR определяются из таблиц истинности, приведенных ниже.

Примечание
Логическое значение UNKNOWN в данной версии СУБД ЛИНТЕР не поддерживается.

3) Выражения внутри скобок вычисляются первыми. Если порядок вычислений не задан, приоритет операций следующий: NOT, AND, OR. Операции одинакового приоритета выполняются слева направо.
4) Результаты выполнения операции AND:

AND	TRUE	FALSE	UNKNOWN
TRUE	TRUE	FALSE	UNKNOWN
FALSE	FALSE	FALSE	FALSE
UNKNOWN	NULL	FALSE	UNKNOWN

5) Результаты выполнения операции OR:

OR	TRUE	FALSE	UNKNOWN
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	UNKNOWN
UNKNOWN	TRUE	UNKNOWN	UNKNOWN

Предикаты

Функция

Определение условия, которое должно быть проверено на соответствие значению истина (TRUE), ложь (FALSE) или не определено (NULL).

Спецификация

[1] <предикат> : : =
<предикат сравнения>
| <интервальный предикат>
| <предикат вхождения>
| <предикат уникальности>
| <предикат подобия>
| <предикат сопоставления>
| <предикат различимости>
| <предикат соответствия>
| <предикат неопределенного значения>
| <предикат неверного вещественного значения>
| <предикат выборки>
| <предикат существования>
| <предикат совмещения>
| <предикат внешнего соединения в стиле ORACLE>

Общие правила

1) Результат предиката получается посредством его применения к данной записи выборки данных.

Предикат сравнения

Функция

Определение сравнений двух значений.

Спецификация

[1] <предикат сравнения> : : =
<сравниваемое значение1>
<оператор сравнения>
<сравниваемое значение2>
[2] <сравниваемое значение1> : : =
<значимое выражение>
| <список значений>
| <подзапрос>
| <скалярная процедура>
[3] <скалярная процедура> : : = идентификатор
[4] <сравниваемое значение2> : : =
[(]<список значений>[)]
| <скалярная процедура>
[5] <список значений> : : = <значимое выражение> [, …]
[6] <оператор сравнения> : : =
<равно>
| <не равно>
| <меньше>
| <больше>
| <больше чем или равно>
| <меньше чем или равно>
[7] <равно> : : = =
[8] <не равно> : : = <>
[9] <меньше> : : = <
[10] <больше> : : = >
[11] <больше чем или равно> : : = >=
[12] <меньше чем или равно> : : = <=

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

Синтаксические правила

1) Типы данных <сравниваемого значения1> и <сравниваемого значения2> должны быть совместимы.
2) Результат <подзапроса> должен возвращать одну запись (или одно значение, в зависимости от WHERE-условия), в противном случае будет зафиксирована исключительная ситуация.

```
select dept_no
  from dept
  where
    (boss_surname, boss_name) = (select boss_surname, boss_name
      from dept
        where dept_no=5);
select name, make
  from auto, person
 where auto.personid=person.personid
   and salary >= (select avg(salary) from person);
```

3) Если <сравниваемое значение1> или <сравниваемое значение2> имеет явное NULL-значение, то результат сравнения «на равенство» будет всегда неизвестен (UNKNOWN, то есть логическое NULL-значение), и такие записи не попадут в выборку данных.

Сравните:

```
select count(*) from auto where NULL = personid;

|    0|
select count(*) from auto where personid=null;

|    0|
select count(*) from auto where personid is null;

|    0|
select count(col1) from sample where col2 is not null;
```

4) Хотя сравнение на равенство <значимых выражений> не определено, когда оба имеют NULL-значения, тем не менее, в контекстах GROUP BY, ORDER BY и DISTINCT все NULL-значения считаются равными.
5) Параметрами <скалярной процедуры> может быть SELECT-запрос, возвращающий значение требуемого типа.
6) Если у <скалярной процедуры> нет параметров, то необходимо указать пустой список параметров.

```
create or replace procedure sp_test() result int
code
  return 2; //
end;
select * from auto where personid = sp_test();
```

7) <Список значений> может быть списком константных значений и/или имен столбцов.

```
select count(*)
  from auto
 where (make, model) = ('GENERAL MOTORS', 'CHEVROLET IMPALA');

|   17|
```

Общие правила

1) Числовые значения сравниваются по алгебраическим правилам.
2) Сравнение типов данных NCHAR (VARCHAR) выполняется по весам символов, а не в порядке возрастания их кодов.

```
select count(*)
  from auto
 where (year+1900, color) > (1971, 'BLACK');

|          392|
```

3) Если значения типа CHAR, BYTE, NCHAR имеют разную длину, то сравнение производится посредством временного дополнения короткого значения справа до размера длинного значения. Символами дополнения являются пробел для символьного значения и двоичный нуль – для байтового.
4) Если значения типа VARCHAR, VARBYTE, NCHAR VARYING имеют разную длину, то сравнение выполняется по длине значения.
5) Два значения равны, если все символы (байты) в одних и тех же позициях значения равны. Если два значения не равны, отношение между ними определяется сравнением первой пары неравных символов.
6) Скалярное выражение типа BOOLEAN, при необходимости, автоматически преобразуется в предикат.

   Запрос вида:

```
SELECT * FROM T WHERE B;
```

   идентичен запросу:

```
SELECT * FROM T WHERE B = 'TRUE';
```

7) Для столбцов с типом данных BLOB запрещено:
   - сравнение значений;
   - присвоение значения одного столбца из БД другому столбцу.
8) Для столбцов с типом данных EXTFILE запрещено сравнение значений.
9) Для столбцов типа EXTFILE разрешено сравнение результата функции.

```
create or replace table tst (i int, extf extfile);
insert into tst(i,extf) values (1, extfile('c:/config.sys'));
insert into tst(i,extf) values (2, extfile('d:/Linter.doc'));
insert into tst(i,extf) values (3, extfile('d:/proc.doc'));
insert into tst(i,extf) values (2, extfile('e:/linter.htm'));
insert into tst(i,extf) values (2, extfile('e:/linter.htm'));
--
select count(i)
  from tst
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
 where trim(filename(extf)) = 'e:\linter.htm';

| 2|
```

Примечание
Функция `filename(0)` возвращает имя внешнего файла (см. документ «Полнотекстовый поиск в базе данных»), пункт «Путь к внешнему файлу».

```
select i
  from tst
 where trim(filename(extf)) like '%doc%';

| 2|
| 3|
```

10) Предикат сравнения (кроме `IS NULL` и `IS NOT NULL`) всегда возвращает NULL-значение, если хотя бы один из операндов равен NULL. В результате такие записи не попадают в выборку.
11) Следует различать `<предикат сравнения>` для списка значений с `<предикатом дубликата значения>`. В `<предикате дубликата значения>` NULL-значения сравниваемых значений считаются равными, а в `<предикате сравнения>` – нет (т.к. в этом случае предполагается арифметическое сравнение значений).

Сравните результаты:

```
select count(*) from auto
 where (1,2,nullif(0,0))=(1,2,nullif(0,0));

|    0|
```

```
select count(*) from auto
  where (1, 2, nullif(0,0)) is not distinct
  from (1, 2, nullif(0,0));

|    1000|
```

Интервальный предикат

Функция

Определение сравнения вида «условие-диапазон».

Спецификация

[1] <интервальный предикат>::=
<значимое выражение> [NOT] BETWEEN [ASYMMETRIC | SYMMETRIC]
<нижняя граница> AND <верхняя граница>
[2] <нижняя граница>::=
{<значимое выражение> | <интервальный запрос>}
[3] <верхняя граница>::=
{<значимое выражение> | <интервальный запрос>}
[4] <интервальный запрос>::=
<select-запрос выборки>

| (<table-запрос выборки>)
| (<values-запрос выборки>)

Синтаксические правила

1) Типы всех <значимых выражений> должны быть совместимыми.

```
Salary*(Age -30) BETWEEN 0 AND 52*3500*10
```

2) Для значений типов CHAR и VARCHAR проверяется совпадение кодировок, при несовпадении преобразование типов не выполняется.

3) <Интервальный запрос> не должен возвращать множественное значение.

```
select "Наименование"
  from "Наличие"
 where "Цена" BETWEEN (select min(sum1) from "Прейскурант")
                  AND (select avg(sum1) from "Прейскурант");

create or replace table tst(i int, c char(5));
insert into tst values (1, '11111');
insert into tst values (2, '22222');
insert into tst values (3, '33333');
insert into tst values (4, '44444');
insert into tst values (5, '55555');

select count(*) from tst
 where i between (values(2)) and (values(4));

|         3|
```

```
create or replace table tst1(i int);
insert into tst1 (values (2);
insert into tst1 (values (4);

select count(*) from tst
 where i between (table tst1 where rowid=1)
             and (table tst1 where rowid=2);

|         3|
```

4) Выражение:

```
<значимое выражение> BETWEEN <нижняя граница> AND <верхняя
 граница>
эквивалентно:
<значимое выражение> >= <нижняя граница> AND
<значимое выражение> <= <верхняя граница>
```

Эквивалентные конструкции:

```
select count(*) from auto where cylnders between 10 and 12;

select count(*) from auto where cylnders>=10 and cylnders<=12;
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

5) Выражение:

```
<значимое выражение> NOT BETWEEN <нижняя граница>
AND <верхняя граница>
эквивалентно:
NOT (<значимое выражение > BETWEEN <нижняя граница>
AND <верхняя граница>)
```

Эквивалентные конструкции:

```
select count(*) from auto where cylnders not between 10 and 12;

select count(*) from auto where not cylnders between 10 and 12;
```

6) Опция SYMMETRIC задаёт свойство симметричности интервальных границ, то есть значение <интервального предиката> не должно зависеть от порядка задания границ интервала, например:

значение предиката будет истинным в обоих случаях:

```
! between symmetric 1 and 3
! between symmetric 3 and 1

create or replace table test( i1 int, i2 int, i3 int );
insert into test values ( 1, 2, 3 );
insert into test values ( 2, 3, 1 );
insert into test values ( 3, 1, 2 );
insert into test values ( 1, 3, 0 );

select * from test where i1 between symmetric 4 and 2;

 I1      I2      I3
 --      --      --
|      2|      3|      1|
|      3|      1|      2|

select * from test
  where i1 < 2 and i1 between symmetric i2 and i3;

 I1      I2      I3
 --      --      --
|      1|      3|      0|
```

7) Опция ASYMMETRIC задаёт свойство асимметричности интервальных границ, то есть значение <интервального предиката> зависит от порядка задания границ интервала, например:

значение предиката `between asymmetric 1 and 3` – истинно,

а предиката `between asymmetric 3 and 1` – ложно

```
select count(*) from auto where 2 between asymmetric 1 and 3;
```

```
|   1000|
select count(*) from auto where 2 between asymmetric 3 and 1;

|      0|
select count(*) from auto where 2 between symmetric 3 and 1;

|   1000|
```

8) По умолчанию (если опции симметричности не заданы) используется ASYMMETRIC.

Предикат вхождения

Функция

Определение условия вхождения.

Спецификация

[1] <предикат вхождения>::=
[([<множество1>])] [NOT] IN ([<множество2>])
[2] <множество1>::=
<значимое выражение> [, …] | <запрос сравнения>
[3] <множество2>::=
[<список вхождения> [, …]] | <запрос вхождения>
[4] <список вхождения>::=
<спецификация значения> | <значимое выражение>
[5] <запрос сравнения>::=
<select-запрос выборки>
| (<table-запрос выборки>)
| (<values-запрос выборки>)
[6] <запрос вхождения>::=
<select-запрос выборки>
| (<table-запрос выборки>)
| (<values-запрос выборки>)

Синтаксические правила

1) <Множество1> не должно быть пустым.

```
SELECT make, count(*)
  FROM Auto
 WHERE (color) IN ('BLACK','BLUE','GREEN')
 GROUP BY make;

SELECT make, count(*)
  FROM Auto
 WHERE (make, color) IN (values('FORD','GREEN'))
 GROUP BY make;

|FORD  |   5|
create or replace table tst (search_make char(20),search_color
 char(10));
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
insert into tst (search_make, search_color) values ('FORD',
 'RED');

select * from auto where (make,color) in (table tst);
```

2) Если <множество1> состоит только из одного <значимого выражения>, ограничивающие его круглые скобки не обязательны.

```
select model from auto where color in ('BLACK', 'WHITE')
  and cylnders in (6,8,12);
```

3) <Множество2> может быть пустым.

```
SELECT count(*) FROM Auto WHERE (color) not IN ();

|            1000|
```

4) Типы данных элементов <множества1> и <множества2> должны быть совместимы.

```
SELECT Name FROM Person WHERE PersonID IN (SELECT PersonID FROM
 Auto);
```

5) Если <множество1> содержит только один элемент, то количество элементов <множества 2> может быть произвольным.

6) Если <множество2> не пусто, то <подзапрос сравнения> должен возвращать одну запись выборки данных, при этом количество столбцов записи выборки данных этого подзапроса должно совпадать с количеством элементов <множества 2>.

```
select count(*)
  from auto
  where (select $$$S31
    from $$$USR
      where $$$s34 like 'SYS_') in (select $$$S31 from $$$USR);

|      0|
```

Недопустимый запрос:

```
SELECT make, count(*)
  FROM Auto
WHERE (color) IN (values('RED','GREEN')) GROUP BY make;
```

7) Тип данных <множества1> должен соответствовать типам данных <множества2>. Сравнение одного NULL-значения с другим NULL-значением выдает FALSE, поэтому такие записи в выборку данных не попадают.

```
select count(*) from auto where (make, year) in ( 'FORD', 70);
create table tab1(i int autoinc, c char(10), dt date);
insert into tab1 (c, dt) values( 'abcdef', '01.01.2003');
insert into tab1 (c, dt) values( 'bcdefa', '02.01.2003');
insert into tab1 (c, dt) values( 'bcdefab', null);

select * from tab1;

|1|abcdef|01.01.2003:00:00:00.00|
|2|bcdefa|02.01.2003:00:00:00.00|
```

```
|3|cdefab|NULL                  |
```

```
select *
  from tab1
  where (i,c, dt) in (select i,c, dt from tab1 where c like'_cde
%');

|2 |bcdefa |02.01.2003:00:00:00.00 |
! записи с NULL-значениями в выборку данных не включаются

select * from tab1
 where (i,c, dt) in (select i,c, dt from tab1);

|1|abcdef |01.01.2003:00:00:00.00 |
|2|bcdefab|02.01.2003:00:00:00.00 |
```

8) <Подзапрос вхождения> может возвращать любое число записей выборки данных, при этом количество столбцов записи выборки данных этого подзапроса должно совпадать с количеством элементов <множества1>.

```
select count(*)
  from auto
 where (1, 2, personid) in (select 1, 2, 7
                             union
                            select 1, 2, 240
                             union
                            select 1, 2, 500);

|               3|
```

```
SELECT make, count(*)
  FROM Auto
  WHERE (color) IN (values('RED'), ('GREEN'))
  GROUP BY make;
```

9) <Список вхождения> может содержать одновременно <спецификацию значения> и <значимое выражение>.

```
select personid, depndnts
  from person
  where
  cast personid as real in (9e0, cast (depndnts+2) as real - 1e0)
  and personid < 11;

|2|         1|
|9|         2|
```

Общие правила

1) Пусть <множество1>={x_1}, <множество2>={$y_1, y_2, \ldots y_n$}. Значение предиката равно TRUE в том и только в том случае, когда x_1 равно одному из значений {$y_1, y_2, \ldots y_n$}, то есть ($x_1=y_i$).

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

2) Пусть <множество1>={x_1,x_2, ...x_n}, <множество2>={y_1,y_2,…y_n}. Значение предиката равно TRUE, когда:
   - размерность <множества1> и <множества2> одинакова;
   - x_i равно строго y_i, то есть все элементы <множества1> содержатся в <множестве2>, причём в том же порядке, в каком они расположены в <множестве1> (упорядоченное пересечение множеств не пусто).

Сравните:

```
select count(*)
  from auto
 where (1, 2, personid) in (select 1, 2, 7
                             union
                            select 1, 2, 240
                             union
                            select 1, 2, 500);

|               3|
```

```
select count(*)
  from auto
 where (1, 2, personid) in (select 1, 2, 7
                             union
                            select 1, 240, 2
                             union
                            select 500, 1, 2);

|               1|
```

3) Если единственный элемент <множества1> не совпадает с каким-либо элементом <множества2> (x_1 не равен y_i), значение предиката равно FALSE (пересечение множеств пусто).
4) Если группа элементов <множества1> не совпадает с группой элементов <множества2>, то есть один или несколько x_i не равны соответствующим y_i, значение предиката равно FALSE (пересечение множеств пусто).
5) Если <множество2> пусто, значение предиката всегда равно FALSE.

```
select count(*) from auto where model in();

|            0|
```

```
select count(*) from auto where model not in();

|            1000|
```

6) <Запрос вхождения> фактически выполняется отдельно для каждого значения-кандидата основного запроса, и значения, которые он сформирует, будут составлять список значений <множества2> для этой записи.

   Примеры:
   а)

```
select count(*) from person
```

```
 where (30, personid) in (select age, personid from person);
```

Подзапрос (select age, personid from person) выполняется столько раз, сколько записей в таблице person (в данном примере – 1000 раз).

б)

```
select count(*) from person where (30) in (select age from
 person);

|        1000|
```

Предикат (30) in (select age from person) всегда возвращает `TRUE` (поэтому в `count(*)` попадают все записи таблицы `PERSON`), т.к. подзапрос (select age from person) предиката не связан с основным запросом какими-либо условиями.

в) Подсчитать количество офис-менеджеров старше 30 лет, владеющих автомобилями от концерна `FORD`:

```
select count(*)
  from auto
 where auto.personid in (select distinct personid
                           from person
                          where age > 30
                            and job='OFFICE MANAGER'
                            and auto.make='FORD');

|        3|
```

7) Конструкция:

```
<значимое выражение> NOT IN …
```

имеет тот же результат, что и конструкция `NOT <значимое выражение> IN …`.

Эти конструкции эквивалентны:

```
select name, count(*)
  from person where age not in (30,40,50,60) group by name;
select name, count(*)
  from person where not age in (30,40,50,60) group by name;
```

Предикат уникальности

Функция

Определение условия отсутствия дубликатов в результате запроса.

Спецификация

[1] <предикат уникальности> : : = [NOT] UNIQUE <запрос сравнения>

Общие правила

1) Предикат возвращает значение `TRUE` в том и только в том случае, когда в таблице-результате `<запроса выборки>` отсутствуют какие-либо две записи, одна из

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

которых является дубликатом другой. В противном случае значение предиката – FALSE.

```
SELECT count(*) FROM Auto
  WHERE NOT UNIQUE (values('RED'), ('GREEN'));

|          0|
SELECT count(*) FROM Auto
  WHERE NOT UNIQUE (values('RED'), ('RED'));

|       1000|
```

Пример

Найти номера отделов, в которых есть однофамильцы:

```
select distinct "Номер отдела"
  from "Структура организации"
  where not unique (select "Фамилия"
    from "Штатное расписание"
      where "Штатное расписание"."Номер отдела" =
            "Структура организации"."Номер отдела");
```

Предикат подобия

Функция

Определение сравнения на соответствие шаблону.

Спецификация

[1] <предикат подобия>::=
<символьное выражение> [NOT] LIKE <шаблон> [ESCAPE <ESC-символ>]

[2] <шаблон>::=
<символьное выражение> | <символьный литерал> [{|| | +} {<символьный литерал> | <параметр>} …]

[3] <ESC-символ>::= <символьный литерал> | <UNICODE-литерал>

Синтаксические правила

1) Конструкция:

```
<символьное выражение> NOT LIKE <шаблон>
```

эквивалентна:

```
NOT (<символьное выражение> LIKE <шаблон>).
```

2) Если <ESC-символ> не указан, то символ подчеркивания (_) в <шаблоне> представляет собой указатель на произвольный символ в <символьном выражении>, знак процента (%) в <шаблоне> представляет указатель на подстроку (возможно, пустую), а каждый символ в <шаблоне>, отличный от знаков подчеркивания и процента, представляет сам себя.

```
select distinct make
  from auto where upper(make) LIKE ('%MOTOR%');
```

```
|AMERICAN MOTORS |
|GENERAL MOTORS  |
```

```
select distinct bodytype
  from auto where upper(bodytype) LIKE ('_E%');
```

```
|SEDAN           |
|SEDAN HARDTOP   |
```

3) <Символьное выражение> может быть <UNICODE-типом> или приводиться к нему.

4) <Символьное выражение> может быть символьным константным выражением, символьным параметром, значимым символьным выражением или их комбинацией.

   Запрос с константным выражением:

```
select distinct make, bodytype from auto where bodytype like
 ('SE'||'DAN'||'%');
```

   Запрос с параметром:

```
select distinct make, bodytype from auto where bodytype like
 ('SE'||'DAN'||?);
```

   Запрос с символьной функцией:

```
select * from auto where serialno LIKE 'L9791' +
 substr(serialno,6);
```

5) <ESC-символ> используется для отмены действия специальных символов _ и %.

6) В обычном режиме символ обратного слэша «\» по умолчанию является ESC-символом, в режиме /COMPATIBILITY=STANDARD ESC-символа по умолчанию нет.

7) Если <ESC-символ> задан, то комбинация <ESC-символ>_, <ESC-символ>%, <ESC-символ><ESC-символ> в <шаблоне> указывает на то, что специальные символы _ , %, <ESC-символ> должны представлять самих себя.

   Найти все записи, в которых название организации начинается с АОЗТ_:

```
select count(1) from "Организации"
 where upper("Название") like 'АОЗТ/_%' escape '/';
```

8) Строка шаблона должна гарантировать ее разбивку на следующие последовательности:

   - одиночные не <ESC-символы>;
   - <ESC> + '%';
   - <ESC> + '_';
   - <ESC> + <ESC>.

   То есть после <ESC-символа>, которому не предшествует <ESC-символ>, не может стоять символ конца строки или обычный символ, а могут только '%', '_' и <ESC-символ>.

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

9) Если <символьное выражение> имеет NULL-значение, то результатом предиката будет ложь (FALSE).

10) Конструкция <символьное выражение> LIKE <шаблон> истинна (TRUE), если <символьное выражение> соответствует шаблону, и ложна (FALSE) – в противном случае.

```
SELECT "Название" FROM "Организации"
 WHERE "Название" LIKE('%O_%') or "Название" LIKE('%T_%');
```

11) Если типом данных <шаблона> является UNICODE, то <символьное выражение> также по возможности приводится к этому типу.

12) Если в качестве <ESC-символа> задана UNICODE-константа, то второй байт у этой константы должен быть нулевым.

13) При трансляции запроса с предикатом LIKE выполняется проверка аргумента на правильность использования <ESC-символа>.

```
create or replace table ttt3
  (c char(10) check (c like 'a~qaa' escape '~'));
create or replace table ttt3
  (c char(10) check (c like 'a~_aa' escape '~'));
create or replace table ttt3
  (c char(10) check (c like 'a~%aa' escape '~'));
create or replace table ttt3
  (c char(10) check (c like 'a~~aa' escape '~'));

insert into ttt3 values ('aaa');
!914: запись не удовлетворяет условию CHECK
insert into ttt3 values ('a~qaa');
!914: запись не удовлетворяет условию CHECK
insert into ttt3 values ('aqaa');
!914: запись не удовлетворяет условию CHECK
insert into ttt3 values (' ');
!914: запись не удовлетворяет условию CHECK
insert into ttt3 values ('a~aa');
!ok
insert into ttt3 values (NULL);
!ok
drop table ttt3;
```

Предикат сопоставления

Функция

Определение соответствия шаблону регулярного выражения.

Спецификация

[1] <предикат сопоставления>::=
<символьное выражение> [NOT] SIMILAR TO <шаблон> [ESCAPE <ESC-символ>]
[2] <шаблон>::= <регулярное выражение>
[3] <ESC-символ>::= одиночный ASCII | UNICODE-символ
[4] <регулярное выражение>::=
<элемент выражения>

| <регулярное выражение> <вертикальная черта> <элемент выражения>
[5] <элемент выражения>::=
<коэффициент регулярности>
| <элемент выражения> <коэффициент регулярности>
[6] <коэффициент регулярности>::=
<первичное регулярное выражение>
| <первичное регулярное выражение> *
| <первичное регулярное выражение> +
| <первичное регулярное выражение> ?
| <первичное регулярное выражение> <повторение>
[7] <повторение>::= { <нижняя граница> [, [<верхняя граница>]] }
[8] <нижняя граница>::= <беззнаковое целое>
[9] <верхняя граница>::= <беззнаковое целое>
[10] <первичное регулярное выражение>::=
<описатель символа>
| %
| <набор символов>
| (<регулярное выражение>)
[11] <описатель символа>::=
<экранированный символ> | <неэкранированный символ>
[12] <экранированный символ>::= <ESC-символ> <специальный символ>
[13] <специальный символ>::= любой из символов []()|^-+*%_?{}
[14] <неэкранированный символ>::=
любой символ, отличный от <специального символа> и не совпадающий с <ESC-символом> (если он определен)
[15] <набор символов>::=

_
| [<символ сопоставления> ...]
| [^<символ сопоставления> ...]
| [<включаемый символ сопоставления> ...^<исключаемый символ сопоставления> ...]
[16] <включаемый символ сопоставления>::= <символ сопоставления>
[17] <исключаемый символ сопоставления>::= <символ сопоставления>
[18] <символ сопоставления>::=
<описатель символа>
| <описатель символа> - <описатель символа>
| [:<идентификатор набора символов>:]
[19] <идентификатор набора символов>::= <идентификатор>
[20] <вертикальная черта>::= |

Синтаксические правила

1) <Символьное выражение> и <регулярное выражение> должны иметь строковый тип данных или приводиться к нему.
2) Значением <ESC-символа> должен быть односимвольный литерал, кроме: '[', ']', '(', ')', '|', '^', '-', '+', '*', '_', '%' , '?', '{', '}'.
3) По умолчанию в качестве <ESC-символа> используется символ '\'.
4) Недопустимо использование <ESC-символа> в следующих случаях:
   - <ESC-символ> равен «:», и при этом используются конструкции вида [:alpha:];
   - в функции SUBSTRING, когда количество символов двойной кавычки «"», «прикрытых» <ESC-символом>, меньше двух, например:

```
select substring('This is not line22' similar 'This is
 \"[[:ALPHA:]]+[[:DIGIT:]]+' escape '\');
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

где нет второй последовательности из закрытой слэшем двойной кавычки (\").

5) <Регулярное выражение> задается, как правило, литералом соответствующего типа.
6) Создаваемое по приведенным правилам регулярное выражение представляет собой символьное значение, содержащее все символы, которые требуется явно сопоставлять с символами значения-источника.
7) При вычислении регулярного выражения образуются все возможные символьные значения, не содержащие специальных символов и соответствующие исходному шаблону.
8) Любой символ обозначает себя самого, если это не метасимвол.
9) Метасимволы шаблона обозначают группу других метасимволов.
10) Метасимвол _ заменяет один символ.

```
select count(*) from auto where make similar to 'B_W';

|         10|
```

11) Метасимвол % заменяет произвольное количество символов.

```
select count(*) from auto where make similar to 'F%D';

|        118|
```

12) Альтернативные последовательности разделяются символом <вертикальная черта> ('|'). Внутри квадратных скобок это обычный символ.

```
select count(*) from auto where color similar to 'B%|GR_';

|        416|
```

13) Конструкция <повторение> задает количество повторений <первичного регулярного выражения>.
14) Показатель числа повторений можно также задать с помощью выражения в фигурных скобках. Например:
    - {m} – повторение ровно m раз;
    - {m,} – повторение не менее m раз;
    - {m,n} – повторение не менее m раз и не более n раз.

    Значения n и m не могут быть больше 65536.

```
select count(*) from auto where make similar to 'FER{2,3}_RI';

|         30|
select count(*) from auto where make similar to 'FER{2}_RI';

|         30|
select count(*) from auto where make similar to 'FER{2,}_RI';

|         30|
```

15) Квадратные скобки представляют собой один из вариантов определения набора символов – через перечисление символов. При вычислении регулярного выражения в каждом из генерируемых символьных значений конструкция в квадратных скобках заменяется одним из символов соответствующего набора. Определяемый набор символов может задаваться нижней и верхней границей диапазона допустимых символов.

    Например, в регулярном выражении 'string [3-8]' конструкция в квадратных скобках представляет собой любой одиночный символ, изображающий цифры от 3 до 8 включительно.

16) При задании диапазона значений можно использовать любые символы, но требуется, чтобы значение кода символа <нижней границы> диапазона было не больше значения кода символа <верхней границы>.

17) Сравнивается вес символов.

18) Набор символов можно определять с помощью <идентификатора набора символов>:
    - ALPHA – любой символ алфавита;
    - UPPER – любой символ верхнего регистра;
    - LOWER – любой символ нижнего регистра;
    - ALNUM – любой алфавитно-цифровой символ;
    - SPACE – символ пробела;
    - WHITESPACE – любой из символов с кодами U+0009 (Horizontal Tabulation), U+000A (Line Feed), U+000B (Vertical Tabulation), U+000C (Form Feed), U+000D (Carriage Return), U+0085 (Next Line));
    - DIGIT – любая одиночная десятичная цифра.

```
select distinct make from auto where
  make similar to '_[[:alpha:]]R{2,}A[[:alpha:]]{2}';

|FERRARI             |
```

```
select count(*) from auto where model similar to '%[[:digit:]]';

|         280|
```

19) <Символ сопоставления> можно задавать в виде диапазона символов, например, a-z, 0-9 – цифра и т.д.

20) Метасимвол ^ обозначает начало исключаемого из сопоставления значения (набор символов).

    Например, регулярное выражение '_S[^t]*ing%' генерирует все символьные значения, где вторым символом является "S", за которым (не обязательно непосредственно) следует символьное значение "ing", но между "S" и "ing" отсутствуют вхождения символа "t".

```
create or replace table test(id int, ch char(15));
insert into test values(1, 'String');
insert into test values(2, 'String main');
insert into test values(3, 'Swing');
insert into test values(4, 'Sorting');
```

```
insert into test values(5, 'String main');
insert into test values(6, 'Spring');
insert into test values(7, 'Spring ability');

select id, ch from test
  where ch similar to 'S[^t]*ing%';

 ID          CH
 --          --
|          3|Swing          |
|          6|Spring         |
|          7|Spring ability |
```

21) После символа ^ можно указать несколько наборов символов, которые не должны входить в определяемый набор, например: '_S[a-x^to-p]*ing%' – в данном случае за символом "S" могут стоять символы из диапазона "a-x", но при этом в определяемый набор не должен входить не только символ "t", но и символы из диапазона "o-p".

```
select count(*) from auto where 'ain56eing5'
  similar to '[[:alpha:]65^fgh]+ing*5';

|     1000|
```

22) Круглые скобки () используются для группировки элементов в один логический элемент. В общем случае в круглых скобках могут находиться произвольные регулярные выражения.

```
select count(*) from auto where '034'
  similar to '0(1|2|3)4|5(6|7)8';

|      1000|
```

23) <ESC-символ>, поставленный перед любым метасимволом, отменяет специальную интерпретацию этого метасимвола.

```
select count(*) from auto where 'HARDWARE_123'
  similar to '(HARD|SOFT)WARE%\_[:DIGIT:]+' escape '\';

|       100|
```

24) Если в шаблоне присутствует конструкция в квадратных скобках с использованием двоеточия, то <ESC-символом> не может быть также символ :.

```
select count(*) from auto where '}}}' similar to '\}\}\}';
```

25) Метасимволы имеют модификаторы (пишутся после метасимвола):

- * – элемент регулярного выражения, непосредственно предшествующий символу '*', может появляться ноль или более раз. '*' эквивалентна {0,}.

  Выражение 'string [56]*' генерирует символьные значения 'string', 'string 5', 'string 6', 'string 55', 'string 66', 'string 56', 'string 65', 'string 555' и т.д.
- + – элемент регулярного выражения, непосредственно предшествующий символу '+', может появляться ноль или более раз. '+' эквивалентен {1,}.

```
select count(*) from auto
```

```
  where 'ain56eing5' similar to '[[:alnum:]]+ing*5';

|    1000|
```

- ? – элемент регулярного выражения, непосредственно предшествующий символу '?', может появляться ноль или 1 раз. '?' эквивалентен {0,1}.

26) Проверка совпадения выполняется столько раз, сколько возможно, не учитывая результат действия последующих метасимволов. Для управления количеством совпадений необходимо использовать символ '?'.

Таким образом:

- *? – 0 и более раз;
- +? – 1 и более раз;
- ?? – 0 или 1 раз;
- {n}? – точно n раз;
- {n,}? – не меньше n раз;
- {n,m}? – больше или равно n и меньше m раз.

Общие правила

1) Значением предиката является `TRUE` в том и только в том случае, когда среди всех символьных значений, генерируемых по `<шаблону>` регулярного выражения, найдётся символьное значение, совпадающее с `<символьным выражением>`.

Предикат различимости

Функция

Определение предиката различимости двух записей.

Спецификация

[1] <предикат различимости> : : =
(<проверяемая запись>) <тип проверки> (<сличаемая запись>)
[2] <тип проверки> : : = { IS [NOT] DISTINCT FROM }
[3] <проверяемая запись> : : =
<значимое выражение> [,…] | <проверяемый запрос>
[4] <сличаемая запись> : : =
<значимое выражение> [,…] | <сличаемый запрос>
[5] <проверяемый запрос> : : =
<select-запрос выборки>
| <table-запрос выборки>
| <values-запрос выборки>
[6] <сличаемый запрос> : : =
<select-запрос выборки>
| <table-запрос выборки>
| <values-запрос выборки>

Синтаксические правила

1) `<Проверяемая запись>` и `<сличаемая запись>` должны быть одинаковой степени и иметь совместимые типы данных.

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
select model
  from auto
 where (1, 2, 3) is not distinct from (0+1,4-2,sqrt(9)) limit 2;

|MERCURY COMET GT V8 |
|A-310               |
```

Общие правила

1) <Проверяемая запись> с именами столбцов c1,c2,...,cn и <сличаемая запись> с именами столбцов d1,d2,...,dn считаются записями-дубликатами, если для каждого i (i=1,2,...,n):
   - ci и di не содержат NULL-значения, а их реальные значения равны;
   - ci и di содержат NULL-значения.

```
create or replace table auto_old (make char(20), color char(10));
insert into auto_old(make, color) values ('FORD', 'RED');
insert into auto_old(make, color) values ('FORD', 'GREEN');

select count(model) from auto as a, auto_old as o
 where (a.make,a.color) is not distinct from (o.make, o.color);

|         10|
```

```
create or replace table auto_old (personid int, make char(20),
 color char(10));
insert into auto_old(personid, make, color) values (1, 'FORD',
 'RED');
insert into auto_old(personid, make, color) values (134, 'FORD',
 'GREEN');

select count(model)
  from auto as a, auto_old as o
 where (a.make, a.color) is distinct from (o.make, o.color)
   and a.personid=o.personid;

|          2|
```

2) Предикат возвращает значение TRUE в том и только в том случае, когда <проверяемая запись> и <сличаемая запись> не являются дубликатами. В противном случае значением предиката является FALSE.
3) Если предикат DISTINCT содержит модификатор NOT, то предикат (<проверяемая запись>) IS NOT DISTINCT FROM (<сличаемая запись>) эквивалентен NOT (<проверяемая запись>) IS DISTINCT FROM (<сличаемая запись>).

   Эти конструкции эквивалентны:

```
select count(model) from auto
 where (make,color) is not distinct from values('FORD', 'RED');
```

```
select count(model) from auto
 where not (make,color) is distinct from values('FORD', 'RED');

|          5|
```

Пример

Найти отделы, руководители которых не являются тёзками Петрова Петра.

```
create or replace table dept(dept_no int,
  boss_surname char(10), boss_name char(10));
insert into dept values (1,'ИВАНОВ','ИВАН');
insert into dept values (2,'ПЕТРОВ',NULL);
insert into dept values (3, NULL, NULL);
insert into dept values (4,'ПУПКИН','ВАСИЛИЙ');
insert into dept values (5,'ПЕТРОВ','ПЕТР');
insert into dept values (6, NULL, NULL);
```

Результаты нижеследующих запросов идентичны:

```
select dept_no
  from dept
 where ('ПЕТРОВ','ПЕТР') is distinct
  from (boss_surname,boss_name);

select dept_no
  from dept
 where (boss_surname, boss_name) is distinct from
 (select boss_surname, boss_name
    from dept
   where boss_surname='ПЕТРОВ'
     and boss_name='ПЕТР');

 -------
|          1|
|          2|
|          3|
|          4|
|          6|
```

Предикат соответствия

Функция

Определение условия соответствия проверяемой записи с результатом табличного подзапроса.

Спецификация

[1] <предикат соответствия>::=
(<проверяемая запись>) [NOT] MATCH <тип совпадения> (<сличаемые записи>)

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

[2] <проверяемая запись> : : =
{<имя столбца> | <значимое выражение>}[, …]
[3] <сличаемые записи> : : =
<select-запрос выборки>
| <table-запрос выборки>
| <values-запрос выборки>
[4] <тип совпадения> : : = [UNIQUE] [SIMPLE | PARTIAL | FULL]

Синтаксические правила

1) Количество элементов в <проверяемой записи> должно совпадать с количеством элементов в <сличаемых записях>.
2) Типы данных элементов в <проверяемой записи> должны быть совместимы с типами соответствующих элементов <сличаемых записей>.
3) Сравнение пар соответствующих элементов производится аналогично <предикату сравнения>.
4) Опция UNIQUE запрещает использовать для проверки совпадения дубликаты <сличаемых записей>.
5) Опции SIMPLE (обычное совпадение), PARTIAL (частичное) и FULL (полное) задают условия совпадения записей, имеющих NULL-значения.

Общие правила

1) <Предикат соответствия> возвращает TRUE, если <проверяемая запись> совпадает по заданному критерию с одной из <сличаемых записей>, в противном случае – FALSE (таблица 1).

Таблица 1. Возвращаемые значения <предиката соответствия>

Тип совпадения	Проверяемая запись	Опция UNIQUE	Результат
SIMPLE или не задан	Содержит только NULL-значения	Задана	TRUE
		Не задана	TRUE
	Содержит реальные и NULL-значения	Задана	TRUE
		Не задана	TRUE
	Содержит реальные значения	Задана	TRUE, если среди <сличаемых записей> есть уникальная запись, совпадающая с <проверяемой записью>, иначе FALSE
		Не задана	TRUE, если среди <сличаемых записей> есть, возможно, не уникальная запись, совпадающая с <проверяемой записью>, иначе FALSE
PARTIAL	Содержит только NULL-значения	Задана	TRUE
		Не задана	TRUE

Тип совпадения	Проверяемая запись	Опция UNIQUE	Результат
	Содержит реальные и NULL-значения	Задана	TRUE, если среди <сличаемых записей> есть уникальная запись, реальные значения которой совпадают с соответствующими реальными значениями <проверяемой записи>, иначе FALSE
		Не задана	TRUE, если среди <сличаемых записей> есть, возможно, не уникальная запись, реальные значения которой совпадают с соответствующими реальными значениями <проверяемой записи>, иначе FALSE
FULL	Содержит только NULL-значения	Задана	TRUE
		Не задана	TRUE
	Содержит реальные и NULL-значения	Задана	FALSE
		Не задана	FALSE
	Содержит реальные значения	Задана	TRUE, если среди <сличаемых записей> есть уникальная запись, полностью совпадающая с <проверяемой записью>, иначе FALSE
		Не задана	TRUE, если среди <сличаемых записей> есть, возможно, не уникальная запись, полностью совпадающая с <проверяемой записью>, иначе FALSE

Примеры

1)

```
create or replace table t1 (i1 int, i2 int, i3 int);
create or replace table t2 (i1 int, i2 int, i3 int);
```

```
insert into t1 (i1,i2,i3) values (1,2,3);
insert into t1 (i1,i2,i3) values (1,2,4);
insert into t1 (i1,i2,i3) values (1,1,3);
insert into t1 (i1,i2,i3) values (1,2,3);
insert into t1 (i1,i2,i3) values (1,2,5);

insert into t2 (i1,i2,i3) values (1,2,3);
```

2) Эти 3 запроса идентичны:

```
select count(*) from t1 where (1,2,3) match (select * from t2);
select count(*) from t1 where (1,2,3) match (values (1,2,3));
select count(*) from t1 where (1,2,3) match (table t2);

|       5|
```

3)

```
select count(*) from t1
 where (t1.i1,t1.i2,t1.i3) match (select * from t2);

|       2|
```

4)

```
select count(*) from t1
 where (t1.i1,t1.i2,t1.i3) not match (select * from t2);

|       3|
```

5)

```
select count(*) from t1
 where (t1.i1,t1.i2,t1.i3) match full (select * from t2);

|       2|
```

6) Эти 3 запроса идентичны:

```
select count(*) from t1
 where (t1.i1,t1.i2,t1.i3)
    match partial (select * from t2);

select count(*) from t1
 where (t1.i1,t1.i2,t1.i3)
    match partial (values (cos(0),2,sqrt(9)));

select count(*) from t1
 where (t1.i1,t1.i2,t1.i3)
    match partial (table t2);

|       2|
```

7)

```
select count(*) from t1
 where (t1.i1,t1.i2,t1.i3) match unique (select * from t2);

|       2|
```

8)

```
select count(*) from t1
 where (1,NULLIF(0,0),3) match FULL (select * from t2);

|       0|
```

9)

```
select count(*) from t1
 where (1,NULLIF(0,0),3) match simple (select * from t2);

|       5|
```

10)

```
select count(*) from t1
 where (1,NULLIF(0,0),3) match partial (select * from t2);

|       5|
```

11)

```
create or replace table emp (emp_no int, dept_no int,
  emp_surname char(50), emp_name char(50));
insert into emp values (1,1,'ИВАНОВ','ИВАН');
insert into emp values (2,NULL,'ПЕТРОВ','ПЕТР');
insert into emp values (3,1,'ПУПКИН','ВАСИЛИЙ');
insert into emp values (4,1,'ИВАНОВ','СЕРГЕЙ');
insert into emp values (5,2,'СИДОРОВ','АНТОН');
```

Запрос вернет данные о служащих, которые:

- либо не приписаны к какому-нибудь отделу;
- либо в своем отделе имеют однофамильцев.

```
select emp_surname, dept_no
  from emp
 where (dept_no, emp_surname) match unique simple
    (select emp1.dept_no, emp1.emp_surname
       from emp emp1
      where emp1.emp_no <> emp.emp_no);
```

Предикат неопределенного значения

Функция

Определение проверки на NULL-значение.

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

Спецификация

[1] <предикат неопределенного значения>::=
<значимое выражение> IS [NOT] NULL

Общие правила

1) <Значимое выражение> не должно быть <запросом выборки>.
2) Если <предикат неопределенного значения> является <предикатом внешнего соединения в стиле ORACLE>, то:
   - конструкция '<значимое выражение> IS NOT NULL' игнорируется (если оператор внешнего соединения (+) опустить, то предикат IS NOT NULL выполнится, и результат запроса, написанного ниже, не будет содержать записей).

```
create or replace table test
( a BIGINT not null,
  b INTEGER null,
  c BIGINT null
);
insert into test (a,b,c) values (1,1,1);
insert into test (a,b,c) values (2,null,2);
insert into test (a,b,c) values (3,1,null);
insert into test (a,b,c) values (4,null,null);
insert into test (a,b,c) values (5,1,5);
```

Запрос:

```
select *
  from test t1, test t2
 where t1.b=1
   and t1.a=t2.c(+)
   and t2.a(+) is not null
   and t2.c(+)=2;
```

эквивалентен:

```
select *
  from test t1, test t2
 where t1.b=1
   and t1.a=t2.c(+)
   and t2.c(+)=2;
```

- в конструкции '<значимое выражение> IS NULL' оператор внешнего соединения игнорируется (то есть если оператор внешнего соединения (+) опустить, то предикат IS NULL выполнится, и результаты запросов, написанных ниже, будут эквивалентны).

Запрос:

```
select *
  from test t1, test t2
 where t1.b=1
   and t1.a=t2.c(+)
```

```
   and t2.a(+) is null
   and t2.c(+)=2;
```

эквивалентен:

```
select *
  from test t1, test t2
 where t1.b=1
   and t1.a=t2.c(+)
   and t2.a is null
   and t2.c(+)=2;
```

3) Конструкция '<значимое выражение> IS NULL' имеет одно из двух значений: истина (TRUE) или ложь (FALSE).
4) Конструкция '<значимое выражение> IS NULL' истинна только тогда, когда <значимое выражение> имеет NULL-значение.
5) Конструкция '<значимое выражение> IS NOT NULL' эквивалентна конструкции 'NOT (<значимое выражение> IS NULL)'.
6) <Значимое выражение> может быть задано <SQL-параметром>, который должен содержать спецификацию типа данных параметра.

```
select count(*) from auto where  ? (int) is not null;
5

|  1000|

select count(*) from auto where  ? (int) is not null;
NULL

|        0|
```

Пример

```
select * from person where age is not null;
```

Предикат неверного вещественного значения

Функция

Определение проверки на NaN-значение.

Спецификация

[1] <предикат неверного вещественного значения>::=
<значимое выражение> IS [NOT] NaN

Общие правила

1) Значение NaN ("Not a number" – не число) применимо для вещественных чисел (REAL, DOUBLE) и обозначает результат некорректной арифметической операции (типа деления ноля на ноль или бесконечность).

```
0*∞=NaN
0/0=NaN
∞/∞=NaN
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
+X/0=NaN
-+X/0=NaN
(x – положительное вещественное число)
```

2) При сортировке NaN-значения считаются наибольшими из всех определенных вещественных чисел (NULL-значение не является определенным числом).
3) Любые операции над NaN-значением в качестве результата дают другое NaN-значение.
4) <Значимое выражение> не должно быть <запросом выборки>.
5) Конструкция '<значимое выражение> IS NaN' имеет одно из двух значений: истина (TRUE) или ложь (FALSE).
6) Конструкция '<значимое выражение> IS NaN' истинна только тогда, когда <значимое выражение> имеет NaN-значение.
7) Конструкция '<значимое выражение> IS NOT NaN' эквивалентна 'NOT (<значимое выражение> IS NaN)'.
8) Утилиты СУБД ЛИНТЕР NaN-значения выводят на экран в виде символьных значений следующего вида:
   - +INF: положительная бесконечность;
   - -INF: отрицательная бесконечность;
   - IND: неопределенность;
   - NaN: не число.

Пример

```
create or replace table test(d double);
insert into test select
  cast(HEXTORAW('000000000000F0FF0000000000000000')) as double;
insert into test values(1.1);
insert into test values(2.2);
insert into test values(3.3);
insert into test;
insert into test select
  cast(HEXTORAW('000000000000F07F0000000000000000')) as double;
insert into test select
  cast(HEXTORAW('000000000000F8FF0000000000000000')) as double;
create index d on test;
select * from test order by d;
select * from test where d is NaN;
select * from test where d is NOT NaN;
```

Предикат выборки

Функция

Определение условия выборки.

Спецификация

[1] <предикат выборки>::=
<значимое выражение> <предикат сравнения> <условие> <запрос выборки>

[2] <условие>::= ALL | {SOME | ANY}

Синтаксические правила

1) Типы данных <значимого выражения> и <запроса выборки> должны быть совместимы.
2) ANY является синонимом SOME.

Общие правила

1) <Условие> ANY берет все записи выборки данных, возвращаемые <запросом выборки>, и оценивает их как верные, если хотя бы один из них удовлетворяет <операции сравнения>.

```
SELECT Name FROM Person
 WHERE PersonID = ANY(SELECT PersonID FROM Auto);

select distinct c1 from tab1, tab2 where c1<>c2;
```

это то же самое, что

```
select c1 from tab1 where c1<> any (select c2 from tab2);
```

2) <Условие> ALL берет все записи выборки данных, возвращаемые <запросом выборки>, и оценивает их как верные, если все они удовлетворяют <предикат сравнения>.

```
SELECT Name FROM Person
 WHERE Salary <= ALL (SELECT Salary FROM Person);
```

3) Результат <предиката выборки> выводится путем применения <предиката сравнения> к каждой записи выборки данных из <запроса выборки>. При этом возможны следующие ситуации:

- если <запрос выборки> пуст, или <предикат сравнения> является истинным для каждой записи выборки данных из <запроса выборки>, то результат <значимое выражение> <предикат сравнения> ALL(<запрос выборки>) имеет значение «истина» (TRUE);
- если <предикат сравнения> имеет значение «ложь» (FALSE) хотя бы для одной записи выборки данных из <запроса выборки>, то результат <значимое выражение> <предикат сравнения> ALL(<запрос выборки>) имеет значение «ложь» (FALSE);

Получить список лиц, чья зарплата больше, чем у всех остальных:

```
SELECT Name FROM Person
 WHERE Salary >= ALL (SELECT Salary FROM Person);
```

- если <предикат сравнения> имеет значение «истина» (TRUE) хотя бы для одной записи выборки данных из <запроса выборки>, то результат <значимое выражение> <предикат сравнения> {SOME | ANY} <запрос выборки> имеет значение «истина» (TRUE);

Удалить из table_1 те строки, номера которых встречаются в table_2:

```
delete from table_1 where number = ANY (select some_numbers from
 table_2)
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

- если результат <запроса выборки> пуст или результатом неявно заданного <предиката сравнения> для каждой записи выборки данных из <запроса выборки> является «ложь» (FALSE), то результатом <значимое выражение> <предикат сравнения> {SOME | ANY} <запрос выборки> будет «ложь» (FALSE);
- если результатом <предиката выборки> не являются ни TRUE, ни FALSE, значение его не определено (NULL-значение).

Примечание

Кванторные предикаты с помощью отрицания легко преобразовываются один в другой: ALL к ANY и обратно. С другой стороны, нет такого запроса, сформулированного с их использованием, который нельзя было бы в равной степени хорошо, и даже лучше, сформулировать, используя EXISTS, IN-предикат и агрегатные функции, например:

```
select c1 from tab1 where c1<> all (select c2 from tab2);
```

эквивалентно

```
select c1 from tab1 where c1 not in (select c2 from tab2);
```

или

```
select c1 from tab1 where not exists (select c2 from tab2 where
 c1=c2);
```

Примеры

1)

```
create or replace table
  "Справочник автопроизводителей" (make char(20));
insert into "Справочник автопроизводителей"  values('FORD');
insert into "Справочник автопроизводителей"  values('VAZ');
insert into "Справочник автопроизводителей"  values('FIAT');
```

2)

```
SELECT count(make)  FROM auto
 WHERE make = ANY(table "Справочник автопроизводителей" );

|        143|
```

3)

```
SELECT count(make) FROM auto WHERE make = ANY(values ('FORD'));
```

Предикат существования

Функция

Определение проверки на пустое множество значений.

Спецификация

[1] <предикат существования>::=
[NOT] EXISTS (<проверяемый запрос>)

[2] <проверяемый запрос>::=
<select-запрос выборки>
| <table-запрос выборки>
| <values-запрос выборки>

Общие правила

1) Результат <предиката существования> имеет значение «истина» (TRUE) только в том случае, если результат <проверяемого запроса> не пуст, в противном случае возвращается значение «ложь» (FALSE).

Примечание
Обычно предикат EXISTS используется в сочетании с внешними ссылками.

Примеры

1)

```
select name from person
 where exists(select * from auto
   where personid=person.personid);
```

2)

```
select name from person
 where exists(table auto
   where personid=person.personid);
```

3)

```
select name from person
 where exists (values (100));
```

4) В таблице со столбцом типа int нужно узнать первое пропущенное по порядку значение, то есть если в таблице есть строки: 1 5 2 7 8 10, запрос должен выдать в качестве ответа 3:

```
select min(a1.id+1) as answer
  from tab as a1
 where not exists (select 1 from tab as a2 where a1.id +1 =
 a2.id );
```

Предикат совмещения

Функция

Определение проверки совпадения во времени двух дат.

Спецификация

[1] <предикат совмещения>::=
(<проверяемая запись>) [NOT] OVERLAPS (<сличаемая запись>)
[2] <проверяемая запись>::=
<значимое выражение>
| <select-запрос выборки>
| <table-запрос выборки>
| <values-запрос выборки>
[3] <сличаемая запись>::=
<значимое выражение>
| <select-запрос выборки>
| <table-запрос выборки>
| <values-запрос выборки>

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

Синтаксические правила

1) Количество <значимых выражений> в <проверяемой записи> и <сличаемой записи> должно быть равно 2.
2) Тип данных первого столбца <проверяемой записи> и <сличаемой записи> должен быть совместимым с типом даты-времени.
3) Тип данных второго столбца <проверяемой записи> и <сличаемой записи> должен быть типом даты-времени.

Общие правила

1) <Предикат совмещения> возвращает TRUE, если интервал между датами <проверяемой записи> попадает в интервал между датами в <сличаемой записи>. В противном случае возвращается FALSE.
2) Если одна из дат в <проверяемой записи> или <сличаемой записи> имеет NULL-значение, результат предиката всегда FALSE.

Примеры

1) Найти номера проектов, которые выполнялись в период с 15 января 2007 г. по 15 февраля 2007 г.:

```
create or replace table pro (pro_no int, pro_name char(100),
  pro_sdate date, pro_edate date);
insert into pro
  values (1, 'project_1', '01.02.2006', '01.12.2006');
insert into pro
  values (2, 'project_2', '01.10.2006', '01.03.2007');
insert into pro
  values (3, 'project_3', '01.02.2007', '01.05.2007');
```

Эти три запроса идентичны:

```
select pro_no from pro
 where (pro_sdate, pro_edate)
  overlaps ('15.01.2007', '15.02.2007');

select pro_no from pro
 where (pro_sdate, pro_edate)
  overlaps (values ('15.01.2007', '15.02.2007'));

select pro_no from pro
 where (pro_sdate, pro_edate)
  overlaps (select '15.01.2007', '15.02.2007');

 pro_no
 ------
|          2|
|          3|
```

2)

```
select id from date_list dl1
 where (date1,date2) not overlaps
  (select date1,date2 from date_list dl2 where dl1.id=dl2.id);
```

Предикат внешнего соединения в стиле ORACLE

Функция

Определение внешнего соединения в стиле СУБД ORACLE.

Спецификация

[1] <предикат внешнего соединения в стиле ORACLE>::=
<соединение слева>
| <соединение справа>
| <двустороннее соединение>

[2] <соединение слева>::=
<значимое выражение> = <значимое выражение> (+)

[3] <соединение справа>::=
<значимое выражение> (+) = <значимое выражение>

[4] <двустороннее соединение>::=
<значимое выражение> (+) = <значимое выражение> (+)

Синтаксические правила

1) При отсутствии скобок соединение таблиц выполняется слева направо.
2) Посредством внешнего соединения нельзя соединять одну и ту же таблицу (с оператором `(+)`) более чем с одной таблицей в одном предложении `SELECT` (будет выдано сообщение `1079 – «Неверная операция JOIN»`). С другой стороны, несколько разных таблиц (с оператором `(+)`) могут быть соединены с одной таблицей.

   Допустимая конструкция:

```
…where tab1(+)=tab and tab2(+)=tab
```

   Недопустимая конструкция:

```
…where tab1=tab(+) and tab2=tab(+)…
```

3) Две таблицы не могут быть соединены друг с другом одновременно и через простое, и через внешнее соединения, то есть конструкция вида:

```
SELECT * FROM TAB1 T1, TAB1 T2
 WHERE T1.I = T2.I(+) AND T1.J = T2.J;
```

   является недопустимой.
4) Рекомендуется не использовать в одном условии и внешнее соединение в стиле Oracle, и вызов хранимой процедуры (это может вызвать конфликт в механизме обработки запросов СУБД ЛИНТЕР).
5) Столбец с (+) может стоять под скалярной функцией или в выражении, но он обязательно должен быть один в своей части предиката.

```
create table departments(d_id int, d_name char(20));
insert into departments values (1, 'Sales');
insert into departments values (2, 'IT-technologies');
```


© Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
insert into departments values (3, 'Finance');
insert into departments values (4, 'Management');
insert into departments values (5, 'Design');

create table persons(p_id int, p_name char (20), d_id int);
insert into persons values (1, 'John',  3);
insert into persons values (2, 'Mary',  2);
insert into persons values (3, 'Kate',  4);
insert into persons values (4, 'Jack',  2);
insert into persons values (5, 'Peter', 7);
insert into persons values (6, 'Ann',   5);

SELECT p.p_name, d.d_name
  FROM persons p, departments d
 WHERE p.d_id = d.d_id(+) + 1;
```

Общие правила

1) Внешнее соединение расширяет результат простого соединения: к строкам, возвращаемым простым соединением, добавляются строки из одной таблицы, которые не совпадают со строками другой таблицы.
2) Результатом <значимого выражения>, относящегося к конструкции `(+)`, должно быть значение (возможно, модифицированное) столбца таблицы. Другое <значимое выражение> внешнего соединения может быть произвольным.

```
SELECT distinct auto.personid
  FROM auto, person
 WHERE auto.make (+) like 'FO%'
   AND color='BLACK';

|  20|
|  53|
| 102|
…
create or replace table tab1 (i int);
insert into tab1 values(1);
insert into tab1 values(2);
insert into tab1 values(3);
insert into tab1 values(4);
create or replace table tab2 (i int);
insert into tab2 values(2);
insert into tab2 values(4);
insert into tab2 values(5);
insert into tab2 values(7);

SELECT tab1.i, tab2.i
  FROM tab1, tab2
 WHERE tab1.i(+) = length(to_char(tab2.i, '99'))
```

```
   AND tab2.i > length(to_char(tab2.i, '99'));

|1 |2 |
|1 |4 |
|1 |5 |
|1 |7 |
```

3) Если таблица, помеченная оператором (+), не содержит ни одной строки, совпадающей со строкой другой таблицы, то в результирующей строке генерируются пустые значения для столбцов этой таблицы. Простое соединение не возвращает таких строк.

```
SELECT tab1.i, tab2.i
  FROM tab1, tab2
 WHERE tab1.i(+) = length(to_char(tab2.i,'99')) + 5;

|NULL |2 |
|NULL |4 |
|NULL |5 |
|NULL |7 |
```

Внешнее соединение по константному выражению (для соблюдения синтаксиса используется конструкция вычитания значений одного и того же столбца):

```
create or replace table tab1 (i int, j int);
create or replace table tab2 (ref_tab1 int, k int);
insert into tab1 values (1,1);
insert into tab2 values (1,2);
insert into tab2 values (1,3);

select tab1.i, tab2.ref_tab1, tab2. k
  from tab1, tab2
 where tab1.i = 1
   and tab1.i = tab2.ref_tab1(+)
   and 0 + tab1.i - tab1.i = tab2.k(+);

| 1 | NULL| NULL|
```

Агрегатные функции

Функция

Определяет значение, получаемое применением функции к набору данных.

Спецификация

[1] <агрегатная функция>::= COUNT (*) | <функция>
[2] <функция>::=
<тип функции> ([<классификатор>] <значимое выражение>)
[3] <тип функции>::=
{AVG| MEDIAN| MAX| MIN| SUM| COUNT| VARIANCE| STDDEV| DEFAULT| EVERY| ANY| SOME}

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

[4] <классификатор>::= DISTINCT | ALL

Синтаксические правила

1) Агрегатные функции применяются к таблице (представлению) или группе в сгруппированной таблице (представлении).
2) <Значимое выражение> в <функции> может быть именем столбца или выражением.
3) <Значимое выражение> в <функции> не должно содержать подзапрос или другую <функцию>.
4) Если <классификатор> пропущен, по умолчанию принимается ALL.

```
create or replace table tab1 (i int, j real);
insert into tab1 values (1, 1.3);
insert into tab1 values (3, 2.55);
insert into tab1 values (5, 5.45);
insert into tab1 values (3, 2.55);
insert into tab1 values (NULL, NULL);

! Эти запросы эквивалентны:
select avg(all i), round(avg(all j), 2) from tab1;
select avg(i), round(avg(j), 2) from tab1;

|3.0          |2.96          |
```

5) Операция DISTINCT запрещена для столбца типа BLOB.

Среднее арифметическое значений

Функция AVG возвращает среднее арифметическое значение набора числовых значений.

Синтаксические правила

1) <Значимое выражение> в аргументе должно быть числовым типом данных или приводиться к нему.

Общие правила

1) Тип данных результата:
   - DECIMAL, если тип аргументов INT, SMALLINT, BIGINT, DECIMAL;
   - DOUBLE, если тип аргументов REAL, DOUBLE.
2) Функция применяется к набору значений, полученных из аргумента путем исключения NULL-значений, то есть записи с NULL-значением не учитываются при подсчете записей.

```
! среднее значение данных таблицы tst
|1    |
|2    |
|3    |
|NULL |
|4    |
```

```
! будет
select avg(i) from "tst";

|2.5 |
```

Примечание
Для учета записей с NULL-значением им надо явно присвоить значение. Например,

```
select avg(case i when NULL then 0 else i end) from tab1;
```

или

```
select avg(nvl(i,0)) from tab1;
```

3) Если указан DISTINCT, дубликаты значений исключаются из подсчета.
4) Если функция применяется к пустому набору значений, результат NULL.

Показать пятерку моделей автомобилей, которые предпочитают лица с самой высокой средней зарплатой:

```
select auto.model, round(avg(person.salary)) as avg_mod
  from auto,person
 where auto.personid = person.personid
 group by auto.model
 order by avg_mod desc limit 5;

 MODEL                AVG_MOD
 -----                -------
|CAPRI RS 2600       |                67400|
|DINO SPIDER         |                58182|
|JAVELIN AMX V8      |                57233|
|XJ 6 4.2            |                56829|
|850 SPORT SPIDER    |                56250|
```

Медиана значений (MEDIAN)

Функция MEDIAN возвращает медиану набора данных. Медиана – это значение, которое делит набор данных на две равные половины.

Примечание
Поддерживается со сборки 6.0.17.92.

Синтаксические правила

1) <Значимое выражение> в аргументе должно быть числовым типом данных или приводиться к нему.

Общие правила

1) Тип данных результата:
   - DECIMAL, если тип аргументов INT, SMALLINT, BIGINT, DECIMAL;

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

- DOUBLE, если тип аргументов REAL, DOUBLE.

2) Функция применяется к набору значений, полученных из аргумента путем исключения NULL-значений, то есть записи с NULL-значением не учитываются при подсчете записей.

```
create or replace table test(i int);
insert into test values(NULL),(2),(2),(2),(3);
select median(distinct i) from test;

|2.5 |
```

3) В одном подзапросе с GROUP BY можно использовать только одну функцию MEDIAN. При нарушении указанного требования будет выдан код завершения 99 «Операция не реализована».

```
create or replace table test(s int,i int, j char(10));
insert into test
  values (1, 400, 'd'), (1, 1, 'a'), (2, 10, 'g'),
         (2, 1, 'b'),   (1, 25, 'e');
select s, median(i) from test group by s;

|1 |25.0 |
|2 |5.5  |
```

Примечание

Для учета записей с NULL-значением им надо явно присвоить значение. Например,

```
select median(case i when NULL then 0 else i end) from tab1;
```

или

```
select median (nvl(i,0)) from tab1;
```

4) Если указан DISTINCT, дубликаты значений исключаются из подсчета.

5) Если функция применяется к пустому набору значений, результат NULL.

6) Для определения медианы находится такое наибольшее значение, которое не превышает минимум половину всех значений выборки данных.

7) Если набор данных содержит нечетное количество значений, медиана равняется одному значению, иначе медиана равняется усредненному значению двух средних значений.

Пример

Определить среднюю зарплату сотрудников отдела и медианное значение (медиана показывает, что одна половина сотрудников отдела получает зарплату меньше медианного значения, то есть меньше 23500.0 руб., а другая половина – больше 23500.0 руб., в то время как в среднем они получают по 29250.0 руб.).

```
create or replace table payment(manager char(20), salary numeric);
insert into payment values('Иванов', 20000);
insert into payment values('Петров', 22000);
insert into payment values('Сидоров', 25000);
insert into payment values('Кац', 50000);
```

```
select avg(salary) as "AVG",
       median(salary) as "MEDIAN"
  from payment;

 AVG       MEDIAN
 ---       ------
|  29250.0|  23500.0|
```

Количество записей выборки данных (COUNT)

Функция COUNT возвращает количество записей выборки данных.

Общие правила

1) Тип возвращаемого значения `INT`. Результат не может быть `NULL`.
2) Аргумент `*` функции задает подсчет всех записей.
3) Если задан `DISTINCT`, при подсчете исключаются дубликаты значений. Результат – число различных не NULL-значений в выборке данных.

```
create table tab1 (i int, c char(10));
insert into tab1 values (1, '1');
insert into tab1 values (1,null);
insert into tab1 values (1, '12');
insert into tab1 values (null, '1');
select * from tab1;

 I      C
 ------ ------
| 1    | 1    |
| 1    | NULL |
| 1    | 12   |
| NULL | 1    |

select count(i), count(c), count(*) from tab1;

|3 |3 |4 |

select count(distinct i), count(distinct c), count(*) from tab1;

|1 |2 |4 |
```

4) Конструкции `COUNT(<значимое выражение>)` или `COUNT(ALL <значимое выражение>)` выдают количество всех значений без учета NULL-значений. Результат – число всех значений (включая дубликаты).

```
! Определить число работников, имеющих оклад меньше половины от
 максимального:
select count("категория_раб")
  from "штат_расписание"
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
  where "оклад" < (select 0.5*max("оклад")
  from "штат_расписание");
```

Максимальное значение из множества (MAX)

Функция MAX возвращает наибольшее значение из заданного множества.

Синтаксические правила

1) <Значимое выражение> может иметь любой тип данных, кроме BLOB.
2) Спецификация DISTINCT синтаксически допустима, но не оказывает никакого действия на выбор максимального значения.

Общие правила

1) При подсчете максимального значения NULL-значения игнорируются, однако если все множество значений состоит только из NULL-значений, возвращается NULL.
2) Типом данных результата функции является тип данных <значимого выражения> (кроме типа данных EXTFILE, для которого выдается символьный описатель столбца).

   Например, описатель EXTFILE-столбца содержит спецификацию внешнего файла:

```
create or replace table
  tab_extfile (id integer, ext extfile root 'c:\ext');
insert into tab_extfile values(1, extfile('Abba.mp3'));
insert into tab_extfile values(2, extfile('Beatls.mp3'));

SELECT max(ext) FROM tab_extfile;

|Beatls.mp3   |
```

Примеры

```
! Найти максимальные оклады среди разных групп работников:
select max("оклад") from "штат_расписание"
  group by "категория_раб";

! Найти разрыв между средним и максимальным окладами:
select abs(max("оклад")-avg("оклад")) from "штат_расписание";

! Найти все категории работников, у которых максимальный оклад
! не больше 300 руб.:
select "категория_раб"
  from "штат_расписание"
 group by "категория_раб"
having max("оклад") < 300;
```

Минимальное значение из множества (MIN)

Функция MIN возвращает наименьшее значение из заданного множества.

Синтаксические правила

1) <Значимое выражение> может иметь любой тип данных, кроме BLOB.
2) Спецификация DISTINCT синтаксически допустима, но не оказывает никакого действия на выбор минимального значения.

Общие правила

1) При подсчете минимального значения NULL-значения игнорируются, однако если все множество значений состоит только из NULL-значений, возвращается NULL.
2) Типом данных результата функции является тип данных <значимого выражения> (кроме типа данных EXTFILE, для которого выдается символьный описатель столбца).

Примеры

```
select min(make), min(model) from auto;

|ALPINE |124 SPORT COUPE |

select model from auto group by model
  having min(length(model)) < 4;

|600 |
|SM  |
```

Сумма множества значений (SUM)

Функция SUM возвращает сумму множества числовых значений.

Синтаксические правила

1) <Значимое выражение> в аргументе должно быть числовым типом данных или приводиться к нему.

Общие правила

1) Подсчитанная сумма должна быть в пределах типа возвращаемого значения, иначе фиксируется ошибка переполнения.
2) При подсчете NULL-значения игнорируются.
3) Если задана опция DISTINCT, дубликаты из подсчета исключаются.
4) Тип данных результата:
   - DECIMAL, если тип аргументов INT, SMALLINT, BIGINT, DECIMAL;
   - BIGINT, если ядро СУБД запущено с ключом /COMPATIBILITY=STANDARD и тип аргументов INT, SMALLINT, BIGINT;
   - DOUBLE, если тип аргументов REAL, DOUBLE.

```
select count(*), sum(5) from auto;

|1000 |5000.0 |
```

5) Если тип данных аргумента DECIMAL со значениями точности pi и масштабом si, то точность результата есть max(pi) и масштаб max(si).

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
create table tab1(n decimal);
insert into tab1 values (1.5);
insert into tab1 values (12345.12345);
select * from tab1;

| 1.5         |
| 12345.12345 |

select sum(n) from tab1;

|12346.62345 |
```

Дисперсия множества числовых значений (VARIANCE)

Функция VARIANCE возвращает дисперсию множества числовых значений.

Синтаксические правила

1) <Значимое выражение> в аргументе должно быть числовым типом данных или приводиться к нему.

Общие правила

1) Дисперсия вычисляется по формуле, представленной на рисунке 1.

$$D=\frac{\sum_{i=1}^{n}\left(x(i)\right)^2-\frac{1}{n}\cdot\left(\sum_{i=1}^{n}x(i)\right)^2}{n-1}$$

Рисунок 1. Формула вычисления дисперсии

где *x (i)* – один из элементов множества *x*, *n* – число элементов, а ∑ – сумма, берется по всему множеству *x*.

2) При n=0 возвращается NULL-значение, при n=1 дисперсия считается равной 0.
3) Тип данных результата:
   - DECIMAL, если тип аргументов INT, SMALLINT, BIGINT, DECIMAL;
   - DOUBLE, если тип аргументов REAL, DOUBLE.
4) Функция применяется к набору значений, полученных из аргумента путем исключения NULL-значений, то есть записи с NULL-значением не учитываются при подсчете записей.
5) Если задана опция DISTINCT, дубликаты из подсчета исключаются.

```
select variance(all age), variance(distinct age)
  from person;
```

```
| 137.92212600 | 136.66666666 |
```

Стандартное отклонение множества числовых значений (STDDEV)

Функция STDDEV возвращает стандартное отклонение множества числовых значений.

Синтаксические правила

1) <Значимое выражение> в аргументе должно быть числовым типом данных или приводиться к нему.

Общие правила

1) Стандартное отклонение вычисляется как квадратный корень из дисперсии, определяемой функцией VARIANCE.
2) Функция применяется к набору значений, полученных из аргумента путем исключения NULL-значений, то есть записи с NULL-значением не учитываются при подсчете записей.
3) Если задана опция DISTINCT, дубликаты из подсчета исключаются.
4) Тип возвращаемого значения – DOUBLE.

```
select stddev(all age), stddev(distinct age)
  from person;

| 11.7440251195135 | 11.6904519445001 |
```

Значение по умолчанию

Функция DEFAULT возвращает типизированное значение по умолчанию заданного столбца.

Общие правила

1) Выдается текущее значение по умолчанию заданного столбца. Тип возвращаемого значения – константа.
2) Запрещено использовать для столбцов с типами данных BLOB и EXTFILE.
3) Если значение по умолчанию не задано или недопустимо для данного столбца, выдается NULL.
4) Для столбцов с атрибутом GENERATED BY DEFAULT AS или DEFAULT <значимое выражение> возвращается NULL-значение. Это связано с тем, что <значимое выражение> в данном случае вычисляется непосредственно в момент добавления новой записи (то есть заранее неизвестно). Получить текстовый вид <значимого выражения> по умолчанию можно с помощью функции LINTER_DICT_INFO, например:

```
alter table q1 alter column i set default (j + 9);
select LINTER_DICT_INFO(2,$$$S11,2)
  from LINTER_SYSTEM_USER.$$$SYSRL
 where $$$S13='Q1';

|("J"+9)    |
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

5) Если выражение, от которого берется функция DEFAULT, является столбцом представления, источником для которого служит не столбец базовой таблицы, а выражение, то результатом будет NULL-значение.
6) Типом возвращаемого значения является тип аргумента функции (см. функцию DEFTEXT, которая также возвращает значение по умолчанию, но в символьном представлении).

```
select default(make), default(personid) from auto;

|NULL |NULL |

create or replace table
  tab1 (i int default 100, b varbyte(10) default hex('616161'));
select default(i), default(b) from tab1;

|100 |FFFFFF |

select default(i), cast default(b) as nchar(10) from tab1;

|100 |яяя |
```

Кванторные функции

Функции EVERY, ANY, SOME выполняют проверку истинности набора логических значений.

Синтаксические правила

1) <Значимое выражение> в аргументе должно быть логическим типом данных или приводиться к нему.
2) Функция SOME является синонимом функции ANY.

Общие правила

1) Функция EVERY возвращает значение TRUE, если выражение проверки истинно для всех значений <значимого выражения>, иначе – FALSE.
2) Функция ANY (SOME) возвращает значение FALSE, если выражение проверки ложно для всех значений <значимого выражения>, иначе – TRUE.
3) NULL-значения при вычислении значения функций пропускаются (не учитываются).

Примеры

```
create or replace table test(i int, b boolean);
insert into test values (1, true);
insert into test values (1, true);
insert into test values (2, true);
insert into test values (2, false);
insert into test values (3, false);
insert into test values (4, true);
insert into test values (4, NULL);
insert into test values (5, NULL);
```

```
insert into test values (5, true);
insert into test values (6, NULL);

select every(b) from test where i = 1;

|    T|

select every(case when i > 5 then true else false end)
  from test where i = 6;

|    T|

select any(b) from test where i = 1;

|    T|

select some(case when i > 6 then true else false end)
  from test where i = 6;

|    F|
```

Аналитические функции

Аналитические функции в явном виде можно использовать только в <запросе выборки>, а ссылки на них (по номеру или по имени) – в <ORDER BY-спецификации>.

Функция

Определяет значение, получаемое путем применения функции к интервалу (разделу) набора данных.

Спецификация

[1] <аналитическая функция>::=
{RANK | DENSE_RANK | ROW_NUMBER | AVG | COUNT | MEDIAN | MIN | MAX | STDDEV | VARIANCE | SUM | EVERY | ANY | SOME | FIRST_VALUE | LAST_VALUE | GROUPING | LAG | LEAD | LISTAGG}
(<значимое выражение>)
<OVER-спецификация>|<WITHIN GROUP-спецификация>

Функции ранжирования для интервалов агрегирования

Для иллюстрации работы функций ранжирования используется таблица `rank_tst`:

```
create or replace table rank_tst(i int);
insert into rank_tst (i) values (100);
insert into rank_tst (i) values (200);
insert into rank_tst (i) values (200);
insert into rank_tst (i) values (200);
insert into rank_tst (i) values (500);
insert into rank_tst (i) values (500);
```

```
insert into rank_tst (i) values (1000);
```

Ранжирование с учетом дубликатов записей

Функция

Определяет ранг записей раздела выборки данных с учетом дубликатов записей.

Спецификация

[1] <ранжирование с учетом дубликатов> ::=
RANK ([<значимое выражение>]) <OVER-спецификация>

Синтаксические правила

1) <Значимое выражение> должно иметь целочисленное значение в диапазоне [1-3] или приводиться к нему. При этом:

   RANK(1) = RANK()

   RANK(2) = DENSE_RANK()

   RANK(3) = ROW_NUMBER()

   Для всех остальных значений <значимого выражения> функция RANK возвращает NULL.

2) <Значимые выражения> в <OVER-спецификации> задают столбцы выборки, по которым выполняется разбивка выборки на разделы.

3) <Имена столбцов> в <ORDER BY-спецификации> внутри <OVER-спецификации> задают имена ранжируемых столбцов.

Общие правила

1) Ранжирование записей – это присвоение им определенных значений (рангов).

2) Рангом текущей записи с учетом дубликатов является количество уже ранжированных записей (без учета текущей записи), плюс единица, то есть значение ранга характеризует количество всех предшествующих или последующих записей другого ранга (более высокого или низкого – в зависимости от заданного упорядочивания) раздела выборки по ранжируемому значению, например:

Значение ранжируемого столбца	Ранг записи в выборке
100	1
200	2
200	2
200	2
500	5
500	5
1000	7

   В приведенном примере значение ранга 1 свидетельствует о том, что у данной записи в выборке нет предшествующих записей; значение ранга 2 – у текущей записи есть одна предшествующая запись более высокого ранга; значение ранга 5 – у текущей записи есть 4 предшествующих записи более высокого ранга и т.д.

3) Дубликатами считаются записи, имеющие одинаковые значения в ранжируемых столбцах.
4) Функция RANK использует логическое группирование. Это значит, что когда две или более записи в разделе имеют одинаковое значение в ранжируемом столбце, то такие записи будут иметь одинаковый ранг. Логическое группирование приводит к тому, что числа, соответствующие рангам, идут не подряд, а с промежутками.
5) Функция выполняет ранжирование записей только внутри разделов, определяемых с помощью <OVER-спецификации>.

```
create or replace table rank_example(i char(1), j int);
insert into rank_example (i,j) values ('a', 100);
insert into rank_example (i,j) values ('a', 200);
insert into rank_example (i,j) values ('b', 200);
insert into rank_example (i,j) values ('b', 200);
insert into rank_example (i,j) values ('b', 500);
insert into rank_example (i,j) values ('c', 500);
insert into rank_example (i,j) values ('c', 1000);

select i, j, rank() over (partition by i order by j)
  from rank_example;

 I J
 - -
|a|        100|          1|
|a|        200|          2|
|b|        200|          1|
|b|        200|          1|
|b|        500|          3|
|c|        500|          1|
|c|       1000|          2|

! 1) раздел 1:
|          a|        100|          1|
|          a|        200|          2|
! Так как значения записей различны,
! то и ранги их в разделе различны.

! 2) раздел 2:
|          b|        200|          1|
|          b|        200|          1|
|          b|        500|          3|
! Так как значения двух первых записей одинаковы,
! то и ранги их в разделе совпадают.

! 3) раздел 3:
|          c|        500|          1|
|          c|       1000|          2|
! Так как значения записей различны,
```

```
! то и ранги их в разделе различны.

! В этом примере все записи внутри разделов одинаковые
! (в сортируемом столбце), поэтому их ранги равны 1:
create or replace table rank_example(i char(1), j int);
insert into rank_example (i,j) values ('a', 100);
insert into rank_example (i,j) values ('b', 200);
insert into rank_example (i,j) values ('b', 200);
insert into rank_example (i,j) values ('b', 200);
insert into rank_example (i,j) values ('c', 500);
insert into rank_example (i,j) values ('c', 500);
insert into rank_example (i,j) values ('d', 1000);

select i, j, rank() over (partition by i order by j)
  from rank_example;

|a|        100|          1|
|b|        200|          1|
|b|        200|          1|
|b|        200|          1|
|c|        500|          1|
|c|        500|          1|
|d|       1000|          1|
```

6) Данные в разделе сортируются в соответствии с <ORDER BY-спецификацией>, а затем каждой записи присваивается числовой ранг, начиная с 1.
7) Ранг вычисляется при каждом изменении значений выражений, входящих в <ORDER BY-спецификацию>.
8) Если не задана опция ORDER BY функция RANK считает все записи в разделе (группе) одинаковыми и, соответственно, всем им присваивает ранг 1.
9) При ранжировании NULL-значения считаются одинаковыми.
10) Если столбцы для группировки и ранжирования данных не заданы, то разделом считается вся выборка, а ранжирование не выполняется, т.к. функция RANK сравнивает конкретные значения и ей надо указать столбец (или набор столбцов), данные которого предназначены для этого сравнения.

```
select rank() over () from rank_tst;

|          1|
|          1|
|          1|
|          1|
|          1|
|          1|
|          1|
```

11) Если столбцы для группировки заданы, а для ранжирования нет, то по умолчанию ранжирование выполняется по столбцам группировки, а т.к. в этом случае все записи внутри раздела будут иметь одинаковые значения в ранжируемых столбцах, то ранги всех записей будут равны 1:

```
select rank() over (partition by i) from rank_tst;

|          1|
|          1|
|          1|
|          1|
|          1|
|          1|
|          1|
```

12) В <OVER-спецификации> можно задавать выражение, по которому должны формироваться разделы ранжируемых значений. Например, так можно проранжировать все записи таблицы, не создавая в ней специального столбца, по которому выполняется разбивка выборки на разделы (на примере приведенной выше таблицы rank_tst):

```
select rank() over (partition by 'aaa' order by i) from rank_tst;

|          1|
|          2|
|          2|
|          2|
|          5|
|          5|
|          7|
```

Примеры

1) Оценить цветовые предпочтения владельцев автомобилей производства фирмы GENERAL MOTORS.

```
select distinct a, b, c
  from (select make a,
               color b,
               rank() over (partition by make order by color) c
          from auto
         where make ='GENERAL MOTORS')
 order by a;

 A                B       C
 -                -       -
| GENERAL MOTORS | BLACK  |   1 |
| GENERAL MOTORS | BLUE   |  67 |
| GENERAL MOTORS | BROWN  |  99 |
| GENERAL MOTORS | GREEN  | 111 |
| GENERAL MOTORS | GREY   | 125 |
| GENERAL MOTORS | RED    | 165 |
| GENERAL MOTORS | WHITE  | 188 |
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
| GENERAL MOTORS | YELLOW | 269 |
```

2) Определить трех самых высокооплачиваемых сотрудников в каждом отделе.

```
CREATE OR REPLACE TABLE employee (employee_id INT, dept_id INT,
  first_name VARCHAR(25), last_name VARCHAR(25),
  salary INT, job VARCHAR(20));
INSERT INTO employee
  VALUES (1111, 10, 'Martha', 'White', 4400, 'IT_PROG');
INSERT INTO employee
  VALUES (1112, 10, 'John', 'Black', 8800, 'IT_PROG');
INSERT INTO employee
  VALUES (1113, 20, 'Bill', 'Austin', 7600, 'MK_REP');
INSERT INTO employee
  VALUES (1114, 20, 'Diana', 'Kimes', 4300, 'MK_MAN');
INSERT INTO employee
  VALUES (1115, 20, 'David', 'Peters', 7600, 'IT_PROG');
INSERT INTO employee
  VALUES (1116, 30, 'Sibille', 'Peterson',12000,'AX_ASST');
INSERT INTO employee
  VALUES (1117, 30, 'Jack', 'Klein', 9900, 'MK_REP');
INSERT INTO employee
  VALUES (1118, 30, 'Alex', 'Armstrong', 8500, 'MK_REP');
INSERT INTO employee
  VALUES (1119, 30, 'Jennifer', 'May', 6700, 'AX_ASST');
INSERT INTO employee
  VALUES (1120, 40, 'Roy', 'Hunt', 9900, 'IT_PROG');
INSERT INTO employee
  VALUES (1121, 40, 'Wendy', 'Blunt', 8800, 'AX_ASST');
INSERT INTO employee
  VALUES (1122, 50, 'Valli', 'Begg', 7900, 'MK_MAN');
INSERT INTO employee
  VALUES (1123, 50, 'Pat', 'Donaldson', 4900, 'MK_MAN');

SELECT dept_id, last_name, salary, dept_rank
  FROM (SELECT dept_id,
    job,
    last_name,
    salary,
    RANK() OVER
      (PARTITION BY dept_id ORDER BY salary DESC) dept_rank
      FROM employee) AS part_dept
 WHERE dept_rank <= 3
 ORDER BY dept_id, dept_rank;
```

Ранжирование с исключением дубликатов записей (DENSE_RANK)

Функция

Определяет ранг записей раздела выборки данных с исключением дубликатов записей.

Спецификация

[1] <ранжирование с исключением дубликатов>::=
DENSE_RANK() <OVER-спецификация>

Синтаксические правила

1) <Значимые выражения> в <OVER-спецификации> задают столбцы выборки, по которым выполняется разбивка выборки на разделы.
2) <Имена столбцов> в <ORDER BY-спецификации> внутри <OVER-спецификации> задают имена ранжируемых столбцов.

Общие правила

1) Ранжирование записей – это присвоение им определенных значений (рангов).
2) Рангом текущей записи с исключением дубликатов является количество уже ранжированных уникальных (без дубликатов) записей (без учета текущей записи), плюс единица, то есть значение ранга характеризует количество всех уникальных предшествующих или последующих записей другого ранга (более высокого или низкого – в зависимости от заданного упорядочивания) раздела выборки по ранжируемому значению, например:

Значение ранжируемого столбца записи	Ранг записи в выборке
100	1
200	2
200	2
200	2
500	3
500	3
1000	4

   В приведенном примере значение ранга 1 свидетельствует о том, что у данной записи в выборке нет предшествующих записей; значение ранга 2 – у текущей записи есть одна предшествующая уникальная запись другого ранга (более высокого или низкого – в зависимости от заданного упорядочивания); значение ранга 3 – у текущей записи есть две уникальные предшествующие записи другого ранга и т.д.
3) Дубликатами считаются записи, имеющие одинаковые значения в ранжируемых столбцах.
4) Функция DENSE_RANK, в отличие от функции RANK, выполняет «плотное» (то есть без промежутков) ранжирование записей.
5) Функция выполняет ранжирование записей только внутри разделов, определяемых с помощью <OVER-спецификации>.

```
create or replace table rank_exmp(i int, j int);
insert into rank_exmp (i,j) values (1, 100);
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
insert into rank_exmp (i,j) values (1, 200);
insert into rank_exmp (i,j) values (1, 200);
insert into rank_exmp (i,j) values (1, 200);
insert into rank_exmp (i,j) values (2, 500);
insert into rank_exmp (i,j) values (2, 500);
insert into rank_exmp (i,j) values (2, 1000);
select i, j, dense_rank() over (partition by i order by j)
  from rank_exmp;

 -           -           -
|          1|        100|          1|
|          1|        200|          2|
|          1|        200|          2|
|          1|        200|          2|
|          2|        500|          1|
|          2|        500|          1|
|          2|       1000|          2|
```

6) Данные в разделе сортируются в соответствии с <ORDER BY-спецификацией>, а затем каждой записи присваивается числовой ранг, начиная с 1.
7) Ранг вычисляется при каждом изменении значений выражений, входящих в <ORDER BY-спецификацию>.
8) Если не задана опция ORDER BY, функция DENSE_RANK считает все записи в разделе одинаковыми и, соответственно, всем им присваивает ранг 1.
9) При ранжировании NULL-значения считаются одинаковыми.
10) Алгоритм функции DENSE_RANK в случае, когда не заданы столбцы для группировки записей и/или столбцы для ранжирования записей, аналогичен алгоритму функции RANK при этих же условиях (см. описание функции RANK).
11) В <OVER-спецификации> можно задавать выражение, по которому должны формироваться разделы ранжируемых значений. Например, так можно проранжировать все записи таблицы в обратном порядке, не создавая в ней специального столбца, по которому выполняется разбивка выборки на разделы (на примере приведенной выше таблицы rank_tst):

```
select dense_rank() over (partition by 'aaa' order by i desc)
  from rank_tst;

|          4|
|          3|
|          3|
|          3|
|          2|
|          2|
|          1|
```

Последовательное ранжирование записей (ROW_NUMBER)

Функция

Определяет последовательный номер записи в разделе выборки данных.

Спецификация

[1] <номер записи>::= ROW_NUMBER() <OVER-спецификация>

Синтаксические правила

1) <Значимые выражения> в <OVER-спецификации> задают столбцы выборки, по которым выполняется разбивка выборки на разделы.
2) <Имена столбцов> в <ORDER BY-спецификации> внутри <OVER-спецификации> задают имена ранжируемых столбцов.

Общие правила

1) Ранжирование записей – это присвоение им определенных значений (рангов).
2) Рангом текущей записи при последовательном ранжировании является её порядковый номер в текущем разделе выборки, то есть значение ранга характеризует количество всех предшествующих или последующих записей после упорядочивания записей в разделе по ранжируемому значению, например:

Значение ранжируемого столбца записи	Ранг записи в выборке
100	1
200	1
200	2
200	3
500	1
500	2
1000	1

В приведенном примере значение ранга 1 свидетельствует о том, что у данной записи в разделе нет предшествующих записей; значение ранга 2 – у текущей записи в разделе выборки есть одна предшествующая запись; значение ранга 3 – у текущей записи в разделе есть две предшествующие записи и т.д.

3) Функция выполняет ранжирование записей только внутри разделов, определяемых с помощью <OVER-спецификации>.

```
create or replace table rank_exmp(i int, j int);
insert into rank_exmp (i,j) values (1, 100);
insert into rank_exmp (i,j) values (1, 200);
insert into rank_exmp (i,j) values (1, 200);
insert into rank_exmp (i,j) values (1, 200);
insert into rank_exmp (i,j) values (2, 500);
insert into rank_exmp (i,j) values (2, 500);
insert into rank_exmp (i,j) values (2, 1000);
select rownum, i, j, row_number() over (partition by i order by j)
  from rank_exmp;

|          1|          1|        100|          1|
|          2|          1|        200|          2|
|          3|          1|        200|          3|
|          4|          1|        200|          4|
|          5|          2|        500|          1|
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
|           6|           2|         500|           2|
|           7|           2|        1000|           3|
```

4) Данные в разделе сортируются в соответствии с <ORDER BY-спецификацией>, а затем каждой записи присваивается числовой ранг, начиная с 1.
5) Алгоритм функции ROW_NUMBER в случае, когда не заданы столбцы для группировки записей и/или столбцы для ранжирования записей, аналогичен алгоритму функции RANK при этих же условиях (см. описание функции RANK).

```
select row_number() over () from rank_tst;

|           1|
|           2|
|           3|
|           4|
|           5|
|           6|
|           7|

select rownum, row_number() over (partition by i) from rank_tst;

|           1|           1|
|           2|           1|
|           3|           2|
|           4|           3|
|           5|           1|
|           6|           2|
|           7|           1|
```

6) В <OVER-спецификации> можно задавать выражение, по которому должны формироваться разделы ранжируемых значений. Например, так можно проранжировать все записи таблицы, не создавая в ней специального столбца, по которому выполняется разбивка выборки на разделы (на примере приведенной выше таблицы rank_tst):

```
select row_number() over (partition by 'aaa' order by i) from
 rank_tst;

|           1|
|           2|
|           2|
|           2|
|           5|
|           5|
|           7|
```

Получение первых записей интервалов агрегирования (FIRST_VALUE)

Функция

Определяет для каждой записи выборки первое значение в её интервале агрегирования.

Спецификация

[1] <первая запись>::=
FIRST_VALUE(<значимое выражение>) [<OVER-спецификация>]

Синтаксические правила

1) <Значимое выражение> задает имя упорядочиваемого столбца.

```
create or replace table rank_exmp(i int, j int);
insert into rank_exmp (i,j) values (1, 100);
insert into rank_exmp (i,j) values (1, 700);
insert into rank_exmp (i,j) values (2, 200);

select first_value(i) from rank_exmp;

| 1|
```

2) <Значимые выражения> в <OVER-спецификации> задают столбцы выборки, по которым выполняется разбивка выборки на интервалы агрегирования.

3) <Имена столбцов> в <ORDER BY-спецификации> внутри <OVER-спецификации> задают имена упорядочиваемых столбцов.

Общие правила

1) Для каждой записи выборки возвращается первое значение того интервала агрегирования, которому принадлежит эта запись.

```
create or replace table ntest(i int, j int);
insert into ntest values(1,NULL);
insert into ntest values(2,NULL);
insert into ntest values(2,NULL);
insert into ntest values(2,20);
insert into ntest values(3,30);
insert into ntest values(3,NULL);
insert into ntest values(3,NULL);
insert into ntest values(4,40);
insert into ntest values(4,NULL);
insert into ntest values(4,41);
insert into ntest values(5,50);

select * from ntest;

 I           J
 -           -
|          1|             |
|          2|             |
|          2|             |
|          2|            20|
|          3|            30|
|          3|             |
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
|          3|             |
|          4|           40|
|          4|             |
|          4|           41|
|          5|           50|
```

```
select i as "Интервал",
  nvl(cast first_value(j) over(partition by i) as char, 'null')
    as "Первое значение"
  from ntest;
```

```
 Интервал    Первое значение
 --------    ---------------
|          1|NULL           |
|          2|NULL           |
|          2|NULL           |
|          2|NULL           |
|          3|30             |
|          3|30             |
|          3|30             |
|          4|40             |
|          4|40             |
|          4|40             |
|          5|50             |
```

2) Если значение первой записи равно NULL, возвращается также NULL.

Пример

Для каждого владельца авто получить сведения о марке его авто, возрасте владельца авто и максимальном возрасте среди всех владельцев аналогичных авто.

```
select auto.model, person.name, person.age,
  first_value(person.age)
    over(partition by auto.model order by person.age desc)
      from auto,person
   where auto.personid=person.personid
     order by auto.model;
```

```
| 124 SPORT COUPE | POORE    | 33 | 68 |
| 124 SPORT COUPE | TANIMOTO | 56 | 68 |
| 124 SPORT COUPE | HALL     | 45 | 68 |
| 124 SPORT COUPE | PRATT    | 43 | 68 |
| …               |          |    |    |
| 1275 GT         | MERTA    | 48 | 52 |
| 1275 GT         | MCDONALD | 52 | 52 |
| 1302 S          | POORE    | 64 | 64 |
```

Получение последних записей интервалов агрегирования (LAST_VALUE)

Функция

Определяет для каждой записи выборки последнее значение в её интервале агрегирования.

Спецификация

[1] <последняя запись>::= LAST_VALUE() <OVER-спецификация>

Синтаксические правила

1) <Значимые выражения> в <OVER-спецификации> задают столбцы выборки, по которым выполняется разбивка выборки на интервалы агрегирования.
2) <Имена столбцов> в <ORDER BY-спецификации> внутри <OVER-спецификации> задают имена упорядочиваемых столбцов.

Общие правила

1) Для каждой записи выборки возвращается последнее значение того интервала агрегирования, которому принадлежит эта запись.

```
create or replace table ntest(i int, j int);
insert into ntest values(1,NULL);
insert into ntest values(2,NULL);
insert into ntest values(2,NULL);
insert into ntest values(2,20);
insert into ntest values(3,30);
insert into ntest values(3,NULL);
insert into ntest values(3,NULL);
insert into ntest values(4,40);
insert into ntest values(4,NULL);
insert into ntest values(4,41);
insert into ntest values(5,50);

select * from ntest;

 I           J
 -           -
|          1|            |
|          2|            |
|          2|            |
|          2|           20|
|          3|           30|
|          3|            |
|          3|            |
|          4|           40|
|          4|            |
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
|          4|          41|
|          5|          50|

SELECT i as "Интервал",
  nvl(cast LAST_VALUE(j) over(partition by i) as char, 'null')
    as "Первое значение"
  FROM ntest;

 Интервал    Первое значение
 --------    ---------------
|          1|NULL            |
|          2|20              |
|          2|20              |
|          2|20              |
|          3|NULL            |
|          3|NULL            |
|          3|NULL            |
|          4|41              |
|          4|41              |
|          4|41              |
|          5|50              |
```

Агрегатные функции для интервалов агрегирования

Функция

Определяет агрегатные функции для интервалов агрегирования.

Спецификация

[1] <агрегатная функция для интервала>::= COUNT (*) | <функция>
[2] <функция>::=
<тип функции> ([<классификатор>] <значимое выражение>)
[<OVER-спецификация>]
[3] <тип функции>::=
{AVG | MEDIAN | MAX | MIN | SUM | COUNT | VARIANCE | STDDEV}

Синтаксические правила

1) <Значимые выражения> в <OVER-спецификации> задают столбцы выборки, по которым выполняется разбивка выборки на интервалы агрегирования.
2) <Имена столбцов> в <ORDER BY-спецификации> внутри <OVER-спецификации> задают имена упорядочиваемых столбцов.
3) Опция DISTINCT запрещена. При необходимости её следует выносить на верхний уровень.

Получить количество автомобилей разных производителей.

```
SELECT distinct make, count(*) FROM auto GROUP BY make;
```

Получить количество моделей автомобилей разных производителей.

Правильный запрос:

```
select distinct mk, mo, cnt
  from (select make mk, model mo, count(*)
    over (partition by make, model) cnt FROM auto);
```

Неправильный запрос:

```
select distinct make mk, model mo, count(*)
  over (partition by make, model) cnt
    from auto;
```

(хотя запрос будет оттранслирован без синтаксической ошибки, результат его выполнения окажется неправильным).

Общие правила

1) Агрегатные функции для интервалов агрегирования выполняются аналогично соответствующим агрегатным функциям для множества значений. Отличие в том, что в первом случае агрегирование применяется отдельно для каждого интервала агрегирования, во втором – сразу для всего множества значений.

```
select distinct make, count(*) from auto group by make;
```

```
 MAKE
 ----
| ALPINE              |         7 |
| AMERICAN MOTORS     |        91 |
| BMW                 |        10 |
| CHRYSLER            |       168 |
| CITROEN             |         7 |
| DATSUN              |        10 |
| DE TOMASO           |        12 |
| FERRARI             |        30 |
| FIAT                |        25 |
| FORD                |       118 |
| GENERAL MOTORS      |       284 |
| ISO                 |         6 |
| JAGUAR              |        23 |
| LAMBORGHINI         |         7 |
| LANCIA              |         8 |
| LOTUS               |        17 |
| MASERATI            |        24 |
| MATRA               |         7 |
| MAZDA               |         5 |
| MERCEDES-BENZ       |        33 |
| MG                  |        10 |
| MINI                |         2 |
| MITSUBISHI          |        11 |
| MONTEVERDI          |         6 |
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
| MORGAN              |         9 |
| NSU                 |         6 |
| OPEL                |         7 |
| PORSCHE             |         6 |
| ROLLS-ROYCE         |        16 |
| TRIUMPH             |         9 |
| VOLKSWAGEN          |        10 |
| VOLVO               |         1 |
| VW-PORSCHE          |        15 |
```

2) Значения функций вычисляются после обработки GROUP BY и HAVING-спецификаций, но до выполнения DISTINCT и ORDER BY-спецификаций. Т.е. в случае наличия в запросе конструкции GROUP BY сначала формируется временная таблица – результат GROUP BY с обычными агрегатными функциями, без OVER-конструкции, а уже затем вычисляется функция с OVER-конструкцией, в качестве входных данных для которой берутся данные из временной таблицы, сформированной при обработке GROUP BY-конструкции.
3) Если в запросе с GROUP BY нужно использовать в SELECT или в HAVING пользовательскую функцию, рекомендуется поставить в список группировки в точности такое же выражение для пользовательской функции (а не просто столбец или столбцы, от которого она зависит).

Пример

```
! Создание пользовательской функции:
create or replace procedure get_sname(in snum char(7))
  result char(20) for debug
declare
  var c cursor(vc char(20)); //
code
  open c for "select sname from s where snum=?;", snum; //
  return c.vc; //
end;

! Корректные запросы:
! 1)
select get_sname(s.snum) sname, sum(qty) x
  from s left join sp on s.snum=sp.snum
 group by get_sname(s.snum)
 order by sname;

! 2)
select get_sname(s.snum) sname, sum(qty) x
  from s left join sp on s.snum=sp.snum
 group by s.snum, get_sname(s.snum)
 order by sname;

! Некорректный запрос:

select get_sname(s.snum) sname, sum(qty) x
  from s left join sp on s.snum=sp.snum
```

```
 group by s.snum
 order by sname;
```

Примеры

1) Запрос, который находит суммарное, среднее и медианное значение зарплаты. У этого запроса два интервала агрегирования: по столбцу «вид работы» (job) и по столбцу «отдел» (dept_id). В первом вычисляется сумма зарплат в зависимости от различных видов работ. Во втором – средняя зарплата по отделам.

```
create or replace table payment(dept_id int, job char(20),
  manager char(20), salary numeric);
insert into payment values(101,'Инженер', 'Ivanov', 100);
insert into payment values(101,'Инженер', 'Ivanov_1', 105);
insert into payment values(101,'Инженер', 'Ivanov_2', 110);
insert into payment values(101,'Старший инженер', 'Petrov', 120);
insert into payment values(101,'Зав. сектором', 'Sidorov', 130);
insert into payment values(101,'Нач. отдела', 'Kozlov', 10000);
insert into payment values(201,'Программист', 'Иванов', 300);
insert into payment values(201,'Программист', 'Иванов_1', 320);
insert into payment values(201,'Программист', 'Иванов_2', 335);
insert into payment values(201,'Администратор', 'Петров', 370);
insert into payment values(201, 'Дизайнер', 'Сидоров', 280);
insert into payment values(201,'Нач. группы', 'Козлов', 1500);

select * from payment;

DEPT_ID JOB            SUM_SAL_JOB AVG_SAL_DEPT   MEDIAN_SAL_DEPT
------- ------         ----------- ------------   ---------------
|101   |Нач. отдела    |10000.0   |1760.8333333333|115.0          |
|101   |Инженер        |315.0     |1760.8333333333|115.0          |
|101   |Зав. сектором  |130.0     |1760.8333333333|115.0          |
|101   |Старший инженер|120.0     |1760.8333333333|115.0          |
|201   |Дизайнер       |280.0     |517.5          |352.5          |
|201   |Администратор  |370.0     |517.5          |352.5          |
|201   |Нач. группы    |1500.0    |517.5          |352.5          |
|201   |Программист    |955.0     |517.5          |352.5          |
```

2) Запрос, который находит сотрудника с самой высокой зарплатой по каждому виду работ и самую высокую зарплату в каждом отделе. Для этого он создает два интервала агрегирования (по столбцам job и dept_id) из таблицы Employee. Запрос использует одну и ту же функцию MAX для агрегирования, но применяет ее к этим двум интервалам раздельно. Раздельное применение необходимо потому, что самая высокая зарплата по каждому виду работ не имеет ничего общего с самой высокой зарплатой в каждом отделе.

```
SELECT job, dept_id, last_name AS name, salary, max_dept_sal
  FROM (SELECT dept_id, job, last_name, MAX(salary)
    OVER (PARTITION BY job) max_job_sal, salary, MAX(salary)
      OVER (PARTITION BY dept_id) max_dept_sal
          FROM employee) AS part_deptid
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
 WHERE salary = max_job_sal;
```

3) Запросы, выбирающие список наиболее популярных автомобилей из всех производителей.

```
select mk, mo, cnt, rn
  from (select mk, mo, cnt, row_number()
    over (partition by mk order by cnt desc) rn
      from (select distinct mk, mo, cnt
        from (select auto.make mk, auto.model mo, count(*)
          over (partition by auto.make, auto.model
            order by auto.model) cnt
              from auto, person
                where auto.personid=person.personid)))
  where rn = 1;

select mk, mo, cnt, fv from
  (select mk, mo, cnt, first_value(cnt)
    over (partition by mk order by cnt desc) fv from
  (select distinct mk, mo, cnt
    from (select auto.make mk,auto.model mo, count(*)
      over(partition by auto.make, auto.model
        order by auto.model) cnt
    from auto,person
      where auto.personid=person.personid)))
  where cnt = fv;
```

Кванторные функции для интервалов агрегирования

Функция

Определяет кванторные функции для интервалов агрегирования.

Спецификация

[1] <кванторная функция для интервала>::=
<тип функции> ([<классификатор>] <значимое выражение>)
<OVER-спецификация>

[2] <тип функции>::= EVERY | ANY | SOME

Синтаксические правила

1) <Значимые выражения> в <OVER-спецификации> задают столбцы выборки, по которым выполняется разбивка выборки на интервалы агрегирования.
2) <Имена столбцов> в <ORDER BY-спецификации> внутри <OVER-спецификации> задают имена упорядочиваемых столбцов.
3) Опция DISTINCT запрещена. При необходимости её следует выносить на верхний уровень.

Проверить, действительно ли, что все модели автомобилей производства FORD в таблице AUTO коричневого цвета?

```
select distinct mo, ev
```

```
  from (select model mo,
       some(case color when 'BROWN' then true else false end)
         over (partition by make, model) ev
           from auto
         where make='FORD');

 MO                    EV
 --                    --
| CAPRI RS 2600        | F |
| GRAN TORINO SPORT V8 | F |
| LINCOLN CONTINENTAL  | F |
| LTD COUNTRY SQUIRE   | F |
| MERCURY COMET GT V8  | T |
| MERCURY MONTEREY     | T |
| MUSTANG BOSS 351     | T |
| PINTO RUNABOUT       | F |
```

Общие правила

1) Кванторные функции для интервалов агрегирования выполняются аналогично соответствующим кванторным функциям для множества значений. Отличие в том, что в первом случае агрегирование применяется отдельно для каждого интервала агрегирования, во втором – сразу для всего множества значений.

   Сравните:

```
create or replace table test(i int, b boolean);
insert into test values (1, true);
insert into test values (1, true);
insert into test values (2, true);
insert into test values (2, false);
insert into test values (3, false);
insert into test values (4, true);
insert into test values (4, NULL);
insert into test values (5, NULL);
insert into test values (5, true);
insert into test values (6, NULL);

select every(b) from test group by i;

|T|
|F|
|F|
|T|
|T|
| |

select every(b) over (partition by i) from test;
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
|T|
|T|
|F|
|F|
|F|
|T|
|T|
|T|
|T|
| |
```

Идентификация сгруппированных данных (GROUPING)

Функция GROUPING предназначена для различения сгруппированных данных.

Спецификация

[1] <идентификация сгруппированных данных>::=
GROUPING (<значимое выражение>[, …])

Синтаксические правила

1) Функция используется только совместно с предложением GROUP BY.
2) Функция возвращает уровень агрегирования, соответствующий каждой записи результирующей выборки данных.
3) Элементами <значимого выражения> могут быть:
   - <имя столбца> (или <псевдоним столбца>);
   - <агрегатная функция>;
   - <спецификация типа>;
   - <спецификация значения по условию>.
4) <Значимые выражения> должны ссылаться на группируемый элемент таблицы.
5) <Значимые выражения> не могут быть константами.

Например, запрос:

```
select 1, order_year, grouping(1,order_year), count(*) from
 order_details
 group by rollup(1, order_year);
```

является синтаксически неправильным.

А запрос

```
select cast 1 as int, order_year,
  grouping(cast 1 as int,order_year), count(*)
  from order_details
  group by rollup(cast 1 as int, order_year);
```

является синтаксически правильным.

Общие правила

1) Функция GROUPING возвращает 0, если NULL-значение столбца взято из фактических данных, и 1, если NULL-значение столбца сформировано операцией ROLLUP или CUBE. В SELECT-запросе можно использовать функцию GROUPING, чтобы заменить любое формируемое NULL-значение нужной записью (например «Итого:»). Так как NULL-значения из фактических данных указывают на то, что значение данных неизвестно, в SELECT-запросе можно также указать возвращение записи 'Неизвестно' вместо любых NULL-значений из фактических данных.

Примеры

1)

```
create or replace table
  test( a int, b int, c int, d int, e int, f int);
insert into test values (1,10,100,1000,10000,100000);
insert into test values (1,11,101,1001,10001,100001);
insert into test values (2,20,200,2000,20000,200000);
insert into test values (3,30,300,3000,30000,300000);
insert into test values (3,31,301,3001,30001,300001);

select a, b, c, count(*), grouping(a,b,c)
  from test group by (a,rollup(b,c));
```

```
 A        B        C
 -        -        -
|       1|      10|     100|       1|                0|
|       1|      11|     101|       1|                0|
|       2|      20|     200|       1|                0|
|       3|      30|     300|       1|                0|
|       3|      31|     301|       1|                0|
|       1|      10|        |       1|                1|
|       1|      11|        |       1|                1|
|       2|      20|        |       1|                1|
|       3|      30|        |       1|                1|
|       3|      31|        |       1|                1|
|       1|        |        |       2|                3|
|       2|        |        |       1|                3|
|       3|        |        |       2|                3|
```

2)

```
SELECT
  CASE GROUPING(make)
    when 0 THEN make else 'Всего: ' END AS make,
  CASE GROUPING(COLOR)
    when 0 THEN color else 'Итого по фирме: ' END AS color,
  count(*)
  FROM auto
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
    GROUP BY GROUPING SETS ((make, color), (make),(color));

 MAKE             COLOR
 ----             -----
| ALPINE          |BLACK           |            1|
|ALPINE           |GREY            |            3|
|ALPINE           |WHITE           |            3|
|AMERICAN MOTORS  |BLACK           |           23|
|AMERICAN MOTORS  |BLUE            |           11|
…
|ALPINE           |Итого по фирме: |            7|
|AMERICAN MOTORS  |Итого по фирме: |           91|
|BMW              |Итого по фирме: |           10|
|CHRYSLER         |Итого по фирме: |          168|
|CITROEN          |Итого по фирме: |            7|
|DATSUN           |Итого по фирме: |           10|
…
|Всего:           |BLACK           |          262|
|Всего:           |BLUE            |          108|
|Всего:           |BROWN           |           46|
…
```

Получение предшествующих данных (LAG)

Определяет записи результирующей выборки данных, предшествующие текущей записи выборки данных.

Спецификация

[1] <предшествующие данные>::=
LAG (<выражение> [,<смещение>] [,<стандартное значение>])
[<OVER-спецификация>[<ORDER BY-спецификация>]]
[2] <смещение>::= вещественное число
[3] <стандартное значение>::= <значимое выражение>

Синтаксические правила

1) <Выражение> может быть столбцом или выражением.
2) Если <смещение> не задано, по умолчанию используется 1.
3) <Значимые выражения> в <OVER-спецификации> задают столбцы выборки, по которым выполняется разбивка выборки на интервалы агрегирования.
4) <Имена столбцов> в <ORDER BY-спецификации> задают имена ранжируемых столбцов.

Общие правила

1) Функция предоставляет доступ к записям, предшествующим на заданное <смещение> от текущей записи интервала агрегирования, к которому применена функция (см. раздел «OVER-спецификация»).
2) Если значение <выражения> с учетом <смещения> выходит за пределы интервала агрегирования, или в качестве <смещения> указано отрицательное или дробное число, возвращается <стандартное значение>.

Пример

```
select make, lag(make, 2, 'Нет данных') as make_lag
  from auto where personid < 5;

 MAKE              MAKE_LAG
 ----              --------
| FORD            | Нет данных        |
| ALPINE          | Нет данных        |
| AMERICAN MOTORS | FORD              |
| MASERATI        | ALPINE            |
```

Получение последующих данных (LEAD)

Определяет записи результирующей выборки данных, находящиеся после текущей записи выборки данных.

Спецификация

[1] <последующие данные>::=
LEAD (<выражение>[, <смещение>][, <стандартное значение>])
[<OVER-спецификация>[<ORDER BY-спецификация>]]
[2] <смещение>::= целое положительное число
[3] <стандартное значение>::= <значимое выражение>

Синтаксические правила

1) <Выражение> может быть столбцом или выражением.
2) Если <смещение> не задано, по умолчанию используется 1.
3) <Значимые выражения> в <OVER-спецификации> задают столбцы выборки, по которым выполняется разбивка выборки на интервалы агрегирования.
4) <Имена столбцов> в <ORDER BY-спецификации> задают имена ранжируемых столбцов.

Общие правила

1) Функция предоставляет доступ к записям, находящимся на заданном <смещении> от текущей записи интервала агрегирования, к которому применена функция (см. раздел «OVER-спецификация»).
2) Если значение <выражения> с учетом <смещения> выходит за пределы раздела данных, возвращается <стандартное значение>.

Примеры

1)

```
select MAKE, Lead(MAKE, 2, 'Нет данных') as MAKE_Lead
  from AUTO where personid < 5;

 MAKE              MAKE_LEAD
 ----              ---------
|FORD             | AMERICAN MOTORS |
|ALPINE           | MASERATI        |
|AMERICAN MOTORS  | Нет данных      |
```


© Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
|MASERATI         | Нет данных       |
```

2) Пример реальной задачи.

Пусть есть таблица, в которой хранятся идентификаторы водителей автобусов (person_id), названия населённых пунктов (location_name) и время прохождения автобусом этого населённого пункта (date_time).

Пусть, для примера, эта таблица выглядит следующим образом:

```
create or replace table person_location_time
  (person_id int, location_name varchar(1), date_time date);
insert into person_location_time values
  (1,'A',to_date('1/1/2008 10:00','MM/DD/YYYY HH:MI'));
insert into person_location_time values
  (1,'A',to_date('1/1/2008 12:00','MM/DD/YYYY HH:MI'));
insert into person_location_time values
  (1,'B',to_date('1/1/2008 14:00','MM/DD/YYYY HH:MI'));
insert into person_location_time values
  (1,'A',to_date('1/1/2008 16:00','MM/DD/YYYY HH:MI'));
insert into person_location_time values
  (1,'A',to_date('1/5/2008 21:00','MM/DD/YYYY HH:MI'));
insert into person_location_time values
  (1,'B',to_date('1/6/2008 14:00','MM/DD/YYYY HH:MI'));
insert into person_location_time values
  (1,'A',to_date('1/7/2008 14:00','MM/DD/YYYY HH:MI'));
insert into person_location_time values
  (1,'A',to_date('1/8/2008 14:00','MM/DD/YYYY HH:MI'));
insert into person_location_time values
  (2,'A',to_date('1/1/2008 10:00','MM/DD/YYYY HH:MI'));
insert into person_location_time values
  (2,'A',to_date('1/1/2008 12:00','MM/DD/YYYY HH:MI'));
insert into person_location_time values
  (2,'B',to_date('1/1/2008 14:00','MM/DD/YYYY HH:MI'));
insert into person_location_time values
  (2,'B',to_date('1/2/2008 16:00','MM/DD/YYYY HH:MI'));
insert into person_location_time values
  (2,'B',to_date('1/2/2008 17:00','MM/DD/YYYY HH:MI'));
insert into person_location_time values
  (2,'B',to_date('1/2/2008 18:00','MM/DD/YYYY HH:MI'));
insert into person_location_time values
  (2,'A',to_date('1/4/2008 16:00','MM/DD/YYYY HH:MI'));
insert into person_location_time values
  (2,'A',to_date('1/5/2008 21:00','MM/DD/YYYY HH:MI'));
insert into person_location_time values
  (2,'B',to_date('1/7/2008 19:00','MM/DD/YYYY HH:MI'));
insert into person_location_time values
```

```
  (2,'A',to_date('1/8/2008 19:00','MM/DD/YYYY HH:MI'));
```

Необходимо получить информацию о том, сколько времени назад водитель проезжал через этот же пункт (т.е. соблюдение водителем графика движения):

```
select x.*,divtime(8,prior_date_time,date_time) lag_hours
  from (select person_location_time.*, lag(date_time)
  over (partition by person_id,location_name order by date_time)
    prior_date_time, lead(date_time)
  over (partition by person_id,location_name order by date_time)
    next_date_time
  from person_location_time) x
    order by date_time;
```

Результат:

```
PERSON_ID LOCATION_NAME DATE_TIME  PRIOR_DATE_TIME NEXT_DATE_TIME
   LAG_HOURS

|1|A|01.01.2008:10:00:00.00|                      |
01.01.2008:12:00:00.00| |
|2|A|01.01.2008:10:00:00.00|                      |
01.01.2008:12:00:00.00| |
|1|A|01.01.2008:12:00:00.00|01.01.2008:10:00:00.00|
01.01.2008:16:00:00.00|2|
|2|A|01.01.2008:12:00:00.00|01.01.2008:10:00:00.00|
04.01.2008:16:00:00.00|2|
|1|B|01.01.2008:14:00:00.00|                      |
06.01.2008:14:00:00.00| |
|2|B|01.01.2008:14:00:00.00|                      |
02.01.2008:16:00:00.00| |
|1|A|01.01.2008:16:00:00.00|01.01.2008:12:00:00.00|
05.01.2008:21:00:00.00|4|
...
```

Запрос также (для наглядности) выдаёт для каждой записи текущее время (DATE_TIME), время предыдущего нахождения того же самого водителя в том же населённом пункте (PRIOR_DATE_TIME) и время следующего нахождения того же самого водителя в том же населённом пункте (NEXT_DATE_TIME).

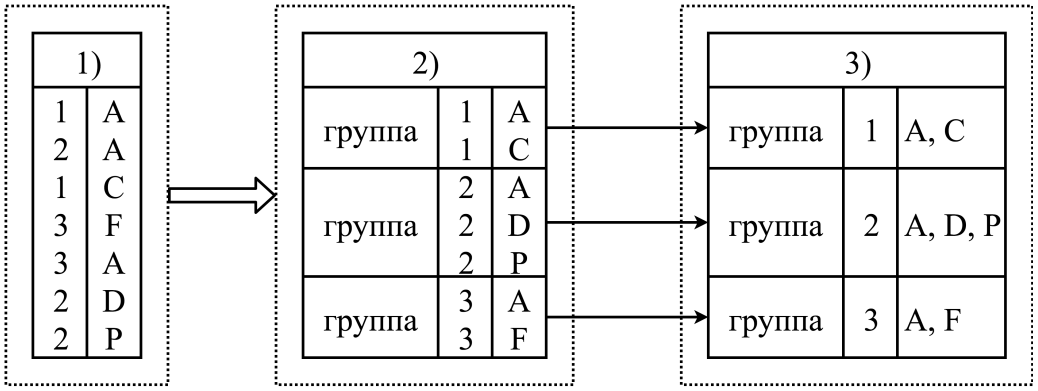
Объединение значений столбца

Функция

Объединяет выбираемые значения столбца в одно значение (т.е. вертикальное представление значений столбца возвращает в виде горизонтального представления).

Схема выполнения функции представлена на рисунке 2.

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026



1) – неупорядоченная выборка данных.
2) – вертикальная группировка данных и сортировка внутри группы с помощью конструкции <WITHIN GROUP-спецификация>.
3) – горизонтальная группировка отсортированных внутри группы данных с помощью функции LISTAGG.

Рисунок 2. Схема объединения значений столбца

Спецификация

[1] <объединение значений столбца>::=
LISTAGG (<выражение>
[,<разделитель> [, <длина возвращаемого значения>]])
[<WITHIN GROUP-спецификация> | <OVER-спецификация>]
[2] <разделитель>::= <символьный литерал>
[3] <длина возвращаемого значения>::=
целочисленное положительное значение

Синтаксические правила

1) Аргумент <выражение> может быть любым выражением, кроме типа данных BOOLEAN, BLOB, EXTFILE. NULL-значения игнорируются.

```
select LISTAGG(rtrim(c), ':') within group (order by c) from tst;
```

Примечание
Для выражения типа BOOLEAN можно вначале привести значение выражения к типу данных INT, например, `select LISTAGG(cast st as int) from tab_w1;`

2) Аргумент <разделитель> задает символьный литерал, который разделяет значения в результирующей записи выборки данных. Аргумент необязательный, по умолчанию он равен NULL-значению, вследствие чего результирующая строка будет без разделителей.

```
create or replace table
  tphones (dept int, name varchar(20), phone varchar(30));
insert into tphones(dept, name, phone)
  values (1,'Петров','1-00-01');
insert into tphones(dept, name, phone)
  values (1,'Иванов','1-11-01');
insert into tphones(dept, name, phone)
  values (1,'Иванов','1-11-03');
insert into tphones(dept, name, phone)
```

```
  values (2,'Сидоров','1-33-03');
insert into tphones(dept, name, phone)
  values (2,'Сидоров','1-33-01');

! а) с разделителем
select LISTAGG(phone,'; ', 50)
  within group (order by phone) phones_phone from tphones;

|1-00-01; 1-11-01; 1-11-03; 1-33-01; 1-33-03|

! б) без разделителя
select LISTAGG(phone) within group (order by phone) phones_phone
  from tphones;

 PHONES_PHONE
 ------------
|1-00-011-11-011-11-031-33-011-33-03|
```

Примечание
Разделитель вставляется после первого значения в результирующей записи выборки данных.

3) Аргумент <длина возвращаемого значения> определяет допустимую для данного запроса длину результирующей записи выборки данных. Аргумент необязательный, по умолчанию длина равна 4000 символов (т.е. функция возвращает значение типа `VARCHAR(4000)`).

Примечание
Если при формировании результирующей записи выборки данных ее длина начнет превышать <длину возвращаемого значения>, возвращается код завершения 1133 «`Результат строковой конкатенации слишком велик`».

4) Одновременное использование <OVER-спецификации> и <WITHIN GROUP-спецификации> не разрешается.

Общие правила

1) Функция `LISTAGG` упорядочивает данные, объединенные в группы, после чего соединяет значения <выражение>:
   - как одиночная агрегатная функция, `LISTAGG` обрабатывает все записи выборки данных и возвращает одно значение;

```
Тестовые данные.
create or replace table
  tphones (dept int, name varchar(20), phone varchar(30));
insert into tphones(dept, name, phone)
  values (1,'Петров','1-00-01');
insert into tphones(dept, name, phone)
  values (1,'Иванов','1-11-01');
insert into tphones(dept, name, phone)
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
  values (1,'Иванов','1-11-03');
insert into tphones(dept, name, phone)
  values (1,'Иванов','1-11-02');
insert into tphones(dept, name, phone)
  values (1,'Иванов','1-11-05');
insert into tphones(dept, name, phone)
  values (2,'Сидоров','1-33-03');
insert into tphones(dept, name, phone)
  values (2,'Сидоров','1-33-01');
insert into tphones(dept, name, phone)
  values (2,'Сидоров','1-33-02');

! Выдать одной строкой отсортированный список
! всех номеров телефонов, внесенных в таблицу tphones:
select LISTAGG(phone, '; ', 80)
  within group (order by phone) phones_phone
    from tphones;

 PHONES_PHONE
 ------------
|1-00-01; 1-11-01; 1-11-02; 1-11-03; 1-11-05; 1-33-01; 1-33-02;
 1-33-03      |
```

- как групповая агрегатная функция, LISTAGG обрабатывает и возвращает данные для каждой группы, указанной в <GROUP BY-спецификации>;

```
! Выдать для каждого человека одной строкой
! отсортированный список его телефонов:
select name, LISTAGG(phone, '; ', 50)
  within group (order by phone) phones_phone
  from tphones group by name;

 NAME                 PHONES_PHONE
 ----                 ------------
|Иванов               |1-11-01; 1-11-02; 1-11-03; 1-11-05     |
|Петров               |1-00-01                                |
|Сидоров              |1-33-01; 1-33-02; 1-33-03              |
```

- как аналитическая функция, LISTAGG обрабатывает данные, разбитые на блоки, задаваемые <OVER-спецификацией>.

```
! Выдать для каждой записи таблицы (относящейся к человеку)
! все отсортированные телефоны этого человека:
select distinct name, LISTAGG(phone, '; ', 40)
  over (partition by name order by phone) phones from tphones;

 NAME                 PHONES
 ----                 ------
|Иванов               |1-11-01; 1-11-02; 1-11-03; 1-11-05      |
```

```
|Петров                |1-00-01                                    |
|Сидоров               |1-33-01; 1-33-02; 1-33-03                  |
```

Спецификация типа данных

Функция

Определение оператора явного преобразования типов данных.

Спецификация

[1] <спецификация типа>::=
CAST [( ] <значимое выражение> AS <тип данных> [ )]

Синтаксические правила

1) <Значимое выражение> должно иметь числовой, дата-время, логический, байтовый или символьный тип данных языка SQL (возможно, с указанием длины, точности и масштаба) либо иметь NULL-значение.
2) Если из контекста запроса невозможно определить тип данных NULL-значения <значимого выражения>, по умолчанию используется `INTEGER`.

   Сравните:

   - используется значение по умолчанию:

```
select NULL from $$$USR where rowid >2;
```

   - тип данных берется из контекста:

```
select 'A'
  union
select NULL;

|          A|
|           |
```

   - явное указание типа данных:

```
select 1, null
  union
select null, cast null as char(10);
```

3) <Значимое выражение> не должно быть <запросом выборки>.
4) Каждое числовое выражение может быть явно преобразовано в любой другой допустимый числовой тип или в другую точность представления. Если результат преобразования (с возможным округлением) не попадает в диапазон значений типа результата, то выдается ошибка.

```
select cast 5.7 as int, cast (2000+length(user))/76.6 as real;

|         5|     26.97128|

select 5.78865366547895, cast 5.78865366547895 as decimal(11,5);

|5.78865366547895 |5.78865 |
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

Общие правила

1) Символьное выражение может быть преобразовано в любой числовой тип. При этом в результате символьного выражения отсекаются ведущие и концевые пробелы, а остальные символы преобразуются в числовое значение по правилам языка SQL. Невозможность преобразования диагностируется ошибкой.

```
select cast '6754' as bigint,
       cast '6774.9' as double,
       cast '123' || '456' as bigint;

|                6754|                6774.9|              123456|
```

2) Если явно заданная длина символьного типа недостаточна, и преобразованное значение не помещается в нем, то значение усекается справа, и ошибка не выдается.

3) Возможно явное преобразование символьного типа в символьный тип с другой длиной. Если длина результата больше длины аргумента, то значение дополняется пробелами; если меньше, то усекается. Сообщений об ошибках при этом не выдается.

```
select cast model as char(30) as "Модель" from auto;
select cast ' Модель ' as char(20)
  union
select model from auto order by 1;
```

Обеспечение совместимости типов при объединении запросов:

```
select name from person
  union
select cast null as char(20) from person;
```

4) NULL-значение преобразуется в NULL-значение соответствующего типа.

5) Числовое выражение может быть преобразовано в символьный тип. Если длина символьного типа не указана явно, то она устанавливается следующим образом:

- для SMALLINT – 6 символов;
- для INTEGER – 11 символов;
- для BIGINT – 20 символов;
- для REAL – 15 символов;
- для DOUBLE – 7 символов;
- для DECIMAL – 32 символа.

```
select cast 567 as char(10);
```

6) Поддерживаются следующие неявные преобразования типов:

- преобразование NULL в любой тип данных;
- для строковых типов – объединение элементов с разной длиной;
- для констант – преобразования CHAR в VARCHAR, INTEGER в SMALLINT, INTEGER в BIGINT, DOUBLE в REAL, DECIMAL в DOUBLE, DECIMAL в REAL.

  Все остальные преобразования необходимо выполнять с помощью оператора CAST.

7) При преобразовании логических выражений константные значения преобразуются сразу, например:

```
cast(0 as boolean) -> FALSE
cast(true as boolean) -> TRUE
cast(true as integer) -> 1
```

а неконстантные выражения – через выражение CASE, например:

```
cast (b as integer) -> case b when false then 0 else 1 end
```

8) Поддерживаются запросы типа:
   - SELECT CASE ... WHEN ... TNEN IS NULL ELSE <значение>;
   - SELECT CASE ... WHEN ... TNEN <значение> ELSE NULL;
   - SELECT NULL FROM ... UNION SELECT <значение> FROM ...;
   - SELECT <значение> FROM ... UNION SELECT NULL FROM ...;

   для них не надо явно писать CAST NULL AS.

9) Если в UNION стоят значения одинакового типа (символьного), но разной длины, то происходит автоматическое приведение типа без явного задания оператора CAST.

10) При операции UNION преобразование строковых констант типа VARCHAR в тип VARCHAR не выполняется.

11) При преобразовании CAST CHAR AS VARCHAR дополнение пробелами VARCHAR не выполняется.

12) При присвоении по команде INSERT (UPDATE) значения CAST AS <строковый тип> некоторому столбцу без явного указания длины строкового типа длина делается равной длине столбца.

13) При обработке числовой константы, если не удалось ее преобразовать к типу DECIMAL, делается попытка преобразовать ее к типу DOUBLE.

14) При преобразовании значений типа DATE в строковые и целые типы данных и обратно через конструкцию CAST используются стандартные форматы (см. функции TO_CHAR, TO_DATE и TO_NUMBER).

```
select cast '28.04.2003' as date, cast sysdate as char(20);

|28.04.2003:00:00:00.00 |10.04.2003:14:08:59.00 |
```

15) Если аргумент агрегатной функции SUM имеет тип данных INTEGER, то при вычислении выражения типа CAST (SUM(<значимое выражение>) AS INTEGER) он не преобразуется в тип данных DECIMAL (сумма считается в целых числах). Это справедливо и для типов данных SMALLINT, BIGINT.

Спецификация значения по условию

Функция

Выбор из множества значений одного значения по заданному условию.

Спецификация

Возможны два варианта:

[1] <спецификация значения по условию>::=
CASE <значимое выражение условия>

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

{WHEN <значимое выражение условия>[, …]
THEN <значимое выражение>}
[ELSE <значимое выражение>] END

[2] <значимое выражение условия>::=
<значимое выражение>| (<список значений>)

[3] <список значений>::= <значимое выражение>[, ...]

[1] <спецификация значения по условию>::=
CASE
{WHEN <предикат> THEN <значимое выражение> }...
ELSE <значимое выражение> END

Синтаксические правила

1) В первом случае вычисляется <значимое выражение условия>, стоящее сразу после CASE; если одно из его значений совпадает со <значимым выражением условия>, указанным в первом WHEN, то возвращается результат <значимого выражения> из первого THEN и т.д. Если же оно не совпадает ни с одним из значений, указанных в WHEN, то возвращается результат выражения, стоящего после ELSE, либо NULL-значение, если ELSE отсутствует.

2) При сравнении NULL-значения <значимого выражения условия> в опции CASE с NULL-значением <значимого выражения условия> в опции WHEN:

   - TRUE, если ядро СУБД запущено без ключа /COMPATIBILITY=STANDARD;
   - FALSE, если ядро СУБД запущено с ключом /COMPATIBILITY=STANDARD.

3) Во втором случае сначала проверяется первое условие; если оно выполняется, то возвращается результат, указанный в первом THEN и т.д.

4) Все возвращаемые выражения должны иметь одинаковый тип (символьные могут быть разной длины, тип результата берется по большей длине).

5) Допустимыми <предикатами> являются:

   - <предикат сравнения>

```
SELECT make, CASE WHEN year=70 THEN 1970
ELSE 1971 END FROM Auto;

select distinct make, model from auto
 where make=case cylnders when 8 then upper('ford')
                          else upper('chrysler') end;

|CHRYSLER |DODGE CHALLENGER SIX |
|FORD     |LINCOLN CONTINENTAL  |
|FORD     |LTD COUNTRY SQUIRE   |
|FORD     |MERCURY COMET GT V8  |
|FORD     |MERCURY MONTEREY     |
|FORD     |MUSTANG BOSS 351     |
```

Две одинаковые по результатам конструкции:

```
SELECT
  CASE Make
    WHEN 'FORD' THEN 1
```

```
    WHEN 'FERRARI' THEN 2
    ELSE 3
  END
  FROM Auto;

SELECT
  CASE
    WHEN Make='FORD' THEN 1
    WHEN Make='FERRARI' THEN 2
    ELSE 3
  END
  FROM Auto;
```

- <интервальный предикат>

```
SELECT make, CASE WHEN year between 60 and 70 THEN 1970 ELSE 1971
 END FROM Auto;
```

- <предикат вхождения>

```
SELECT make,
  CASE WHEN year in (60, 70) THEN 1970 ELSE 1971 END FROM Auto;
select * from test where case when
  id in (select id from test where rowid>1)
  then 1 else 0 end = 1;
select * from test where case when
  (id in (select id from test where rowid=1)
  or id in (select id from test where rowid=3))
  then 1 else 0 end = 1;
```

- <предикат подобия>

```
SELECT make,
  CASE WHEN Make like 'FO%' THEN 1
       WHEN Make='GE%' THEN 2
       ELSE 3 END FROM Auto;
```

- <предикат неопределенного значения>

```
SELECT CASE C WHEN NULL THEN
  'NULL'
  ELSE
    'NOT-NULL'
  END FROM T;
```

Общие правила

1) Если в CASE-выражении выбираются альтернативы, имеющие различные типы данных (CHAR/VARCHAR, NCHAR/NCHAR VARYING), то результату присваивается тип данных переменной длины (VARCHAR, NCHAR VARYING).
2) В качестве <значимого выражения> не могут использоваться BLOB и EXTFILE столбцы. Эти конструкции некорректны:

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
case ... when .. then NULL else BLOB_COLUMN end
case ... when .. then BLOB_COLUMN else NULL end
```

3) Если числовые <значимые выражения> в конструкции CASE WHEN <предикат> имеют различный тип данных, то выполняется автоматическое преобразование их к типу данных основного <значимого выражения> конструкции CASE:

- в качестве типа данных основного <значимого выражения> выбирается тип данных первого неконстантного выражения (например, столбца);

```
create or replace table int2(i2 smallint, d2 real);
insert into int2 values(1,1.1);
insert into int2 values(2,2.2);
insert into int2 values(3,3.3);

! Тип данных результата  – короткое целое:
select CASE WHEN (i2 < 3) THEN 2.2 ELSE i2 END from int2;

|          2|
|          2|
|          3|
```

- если все выражения константные, выбирается тип первого выражения по порядку следования;

```
! Тип данных результата  – целое число:
select CASE WHEN (i2 < 3) THEN 0 ELSE 1.1 END from int2;

|          0|
|          0|
|          1|

! Тип данных результата  – число с фиксированной точкой:

select CASE WHEN (i2 < 3) THEN 0.1 ELSE 1 END from int2;

|                    0.1         |
|                    0.1         |
|                    1.0         |
```

- если в CASE-конструкции имеется несколько неконстантных выражений разных числовых типов, будет выдаваться код завершения 2031 о несовместимости типов данных (в этом случае необходимо использовать явное преобразование типов с помощью конструкции CAST);

```
! Выдается код завершения  2031 – несовместимые  типы данных

select CASE WHEN (i2 < 3) THEN d2 ELSE i2 END from int2;
```

- если неконстантные выражения отсутствуют или имеется только одно из них, типы данных всех выражений, отличающиеся от основного типа данных, будут приведены к основному с помощью неявной конструкции CAST.

```
select CASE WHEN (i2 < 3) THEN i2  ELSE 0 END from int2;
```

```
       |     1|
       |     2|
       |     0|
select CASE WHEN (i2 < 3) THEN 0 ELSE i2 END from int2;
       |     0|
       |     0|
       |     3|
```

4) В общем случае все предикаты обрабатывают только одну запись (за исключением предикатов сравнения). В конструкции CASE для предиката EXISTS/NOT EXISTS разрешается использовать выборку подзапроса из нескольких записей.

```
SELECT CASE WHEN (EXISTS (SELECT MAKE FROM AUTO)) THEN TRUE END
 FROM (SELECT 1);
```

Пример

```
create or replace table test(n int, ch char(10));
insert into test values (0,'val0');
insert into test values (1,'val1');
insert into test values (2,'val2');
insert into test values (3,'val3');
insert into test values (4,'val4');
insert into test values (5,'val5');
insert into test values (NULL,NULL);

select n,
       case n
         when 1, 0.0, 3e0 then 'defined {0|1|3}'
         when 5.0 then 'defined 5'
         when 2e0,4 then 'defined {2|4}'
         when NULL then 'defined NULL'
         else 'undefined'
       end status1,
       case ch
         when 'val1', 'val' || '0' then 'defined {val0|val1}'
         when 'val' || '5', 'val3', 'val4'
           then 'defined {val3|val4|val5}'
         when NULL then 'defined NULL'
         when 'val2' then 'defined val2'
         else 'undefined'
       end status2,
       case (n, ch)
         when (1,'val1'), (2.0,'val' || '2'), (3e0,'val3')
           then 'defined {1|2|3}'
         when (5e0, 'val' || '5') then 'defined 5'
         when (0e0, 'val0'),(4,'val4')  then 'defined {0|4}'
         when (NULL,NULL) then 'defined NULL'
         else 'undefined'
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
       end status3
  from test;
drop table test;
```

Результат работы select-запроса примера:

```
 N       STATUS1          STATUS2                    STATUS3
 -       -------          -------                    -------
|   0|defined {0|1|3}|defined {val0|val1}      |defined {0|4}   |
|   1|defined {0|1|3}|defined {val0|val1}      |defined {1|2|3}|
|   2|defined {2|4}  |defined val2              |defined {1|2|3}|
|   3|defined {0|1|3}|defined {val3|val4|val5}|defined {1|2|3}|
|   4|defined {2|4}  |defined {val3|val4|val5}|defined {0|4}   |
|   5|defined 5      |defined {val3|val4|val5}|defined 5       |
|    |defined NULL   |defined NULL              |defined NULL    |
```

Табличное выражение

Функция

Определение простой или сгруппированной табличной выборки данных.

Спецификация

[1] <табличное выражение>::=
<FROM-спецификация>
[<WHERE-спецификация>]
[<GROUP BY-спецификация>]
[<HAVING-спецификация>]

Общие правила

1) Если в <табличном выражении> все необязательные выражения опущены, то результирующая таблица является результатом выполнения <FROM-спецификации>. В противном случае каждая указанная спецификация применяется к результату предшествующей спецификации, и таблица представляет собой результат применения последней спецификации.

```
select count(*) from auto, person;

|986000 |

select count(*) from auto, person
  where auto.personid=person.personid;

|986 |

select make, count(*) from auto, person
  where auto.personid=person.personid group by make;

|MAKE            |CNT |
|ALPINE          |7   |
|AMERICAN MOTORS |88  |
```

```
|BMW              |10   |
...
! ответ на запрос содержит 33 записи

select make from auto, person
  where auto.personid=person.personid
    group by make having count(cylnders)=8;

|LANCIA |
```

2) Результатом <табличного выражения> является полученная таблица, в которой i-й столбец наследует описание i-го столбца таблицы, заданной посредством <FROM-спецификации>.

FROM-спецификация

Функция

Определение таблицы, получаемой из объединения одной или нескольких таблиц (представлений).

Спецификация

[1] <FROM-спецификация>::= FROM <табличная ссылка> [,...]
[2] <табличная ссылка>::=
<первичная таблица>
| <порожденная таблица>
| <соединение таблиц>
| <курсорная процедура>
[3] <первичная таблица>::=
[<имя схемы>.] <имя таблицы>
[[AS] <псевдоним таблицы> [(<псевдоним столбца> [, ...])]
[TABLESAMPLE [<метод выборки>](<процент выборки>)]
[REPEATABLE <номер выборки>]]
[4] <порожденная таблица>::=
(<подзапрос>) [AS] <псевдоним таблицы> [(<псевдоним столбца> [,...])]
[5] <курсорная процедура>::= <идентификатор>
[6] <метод выборки>::= BERNOULLI | SYSTEM
[7] <процент выборки>::= неотрицательное целочисленное значение
[8] <номер выборки>::= положительное целочисленное значение

Синтаксические правила

1) <Имя таблицы> можно использовать в запросе только тогда, когда <табличная ссылка> не задает <псевдоним таблицы>.
2) Максимальное количество <табличных ссылок> равно 32.
3) <Имя таблицы>, указанное во <FROM-спецификации>, не должно совпадать с другим <именем таблицы>, задаваемым в этой же <FROM-спецификации>.

```
SELECT count(*) FROM auto a, auto b
 WHERE a.PersonID=b.PersonID;

SELECT * FROM (SELECT * FROM Person P, Auto A
  WHERE P.PersonID=A.PersonID) AS Pa,
    (SELECT * FROM PERSON P, Finance F
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
  WHERE P.PersonID=F.PersonID) AS Pf limit 5;
```

4) <Псевдоним таблицы> не должен совпадать с другим псевдонимом в той же <FROM-спецификации> и не должен совпадать с <именем таблицы>, заданном в той же <FROM-спецификации>.

5) Областью видимости псевдонимов и имен таблиц во <FROM-спецификации> является вложенный <подзапрос>, содержащий <табличное выражение> с этой <FROM-спецификацией>. Область видимости псевдонима не включает саму <FROM-спецификацию>.

6) Список <псевдонимов столбцов> по количеству должен совпадать с количеством столбцов в <первичной таблице> или <порожденной таблице>.

```
create or replace table tst(i int, c char(5));
insert into tst(i,c) values(200, '1111');

select col1 from tst as a (col1,col2);
select sysdate, user, col2 from tst b (col1,col2);

select col2, field1 from tst a (col1,col2),
  (select sysdate, user) as b (field1, field2);
```

7) Варианты:

- если во <FROM-спецификации> одна <табличная ссылка>, то описание результата <FROM-спецификации> то же, что и описание таблицы из этой <табличной ссылки>;
- если во <FROM-спецификации> содержится более одной <табличной ссылки>, то описание результата <FROM-спецификации> представляет собой конкатенацию описаний таблиц, идентифицированных этими <табличными ссылками>, в порядке следования во <FROM-спецификации>;
- спецификация псевдонима (открытого <имени таблицы>) устанавливает, что этот псевдоним (<имя таблицы>) используется как ее обозначение, определенное таблицей, которая, в свою очередь, определена <именем таблицы> или <порожденной таблицей>.

```
select * from (select (select count(*) from auto
  where person.personid=auto.personid and make='FORD') n
    from person);
```

8) Конструкция TABLESAMPLE предназначена для выборки из большой (по числу записей) таблицы заданного количества случайных записей.

9) Конструкция TABLESAMPLE не применима к представлениям.

10) <Процент выборки> задаёт число (процент) от всех записей в таблице, которые будут включены в выборку. Допустимое значение – в диапазоне от 0 до 100 включительно. Реально будет возвращено не строго заданное количество процентов записей, а приблизительное.

```
SELECT make, model FROM auto TABLESAMPLE (5);
```

11) Опция <метод выборки> задаёт метод вычисления номера записи таблицы для включения её в случайную выборку. В данной версии СУБД методы BERNOULLI и SYSTEM можно использовать только в качестве допустимых элементов синтаксической конструкции, но реальной нагрузки они пока не несут (игнорируются): номер записи для выборки формируется на основе генератора псевдослучайных чисел, встроенного в СУБД.

12) Опция REPEATABLE задает номер повторяемой выборки (число от 1 до 65535). Одинаковые значения <номера выборки> и <процента выборки> приводят к повторному возвращению заданной выборки (если только в таблице не произошли изменения). Если опция REPEATABLE указывается с другим значением <номера выборки> (или <процента выборки>), то будет сформирован иной набор строк таблицы. Изменениями считаются следующие действия над таблицей: вставка, обновление, удаление, перестройка индекса.

```
select count(make) from auto tablesample (5) repeatable (1);

|         60|
select count(make) from auto tablesample (5) repeatable (2);

|         54|
select count(make) from auto tablesample (5) repeatable (7);

|         48|
select count(make) from auto tablesample (5) repeatable (1);

|         60|
```

13) Конструкция TABLESAMPLE может применяться ко всем или нескольким таблицам во <FROM-спецификации>, при этом значения <процента выборки> и <номера выборки> берутся из первой конструкции TABLESAMPLE, значения из остальных конструкций TABLESAMPLE игнорируются.

```
select * from
  auto T1 TABLESAMPLE (10) REPEATABLE (1),
  person T2 TABLESAMPLE (20) REPEATABLE (1)
    where T1.personid=T2.personid;
```

14) <Курсорная процедура> – хранимая процедура, возвращающая курсор. При использовании <курсорной процедуры> обращение к полям курсора недопустимо.

```
create or replace procedure proc_test (in i int)
result cursor(
  MAKE char(20),
  MODEL char(20),
  BODYTYPE char(15),
  CYLNDERS integer,
  HORSEPWR integer,
  DSPLCMNT integer,
  WEIGHT integer,
  COLOR char(10),
  YEAR integer,
  SERIALNO char(16),
  CHKDATE integer,
  CHKMILE integer,
  PERSONID integer)
declare
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
  var d typeof(result);//
code
  open d for direct "select * from auto where personid = " +
 itoa(i) + ";";//
  return d;//
end;

select * from proc_test (5);
```

15) В качестве входных параметров <курсорной процедуры> допускается использовать SELECT-запросы, возвращающие единственное значение требуемого типа.

16) При выполнении SELECT-запроса с <курсорной процедурой> во FROM в случае возврата неоткрытого курсора ошибка будет игнорироваться, и запрос будет работать так, как если бы процедура вернула пустой курсор.

Примеры

```
select * from a TABLESAMPLE(10) REPEATABLE(2) order by aj, ai;

select * from a TABLESAMPLE(10) REPEATABLE(2), b where ai=bi;

select * from a TABLESAMPLE(10) REPEATABLE(2), b where ai>=bi;

select * from a TABLESAMPLE(10) REPEATABLE(2),
  b where aj=bj and ai=bi;
```

WHERE-спецификация

Функция

Задает таблицу, которая получается путем применения <логического выражения> к результату предшествующей <FROM-спецификации>.

Спецификация

[1] <WHERE-спецификация>::= WHERE <логическое выражение>

Синтаксические правила

1) Каждая <спецификация столбца>, непосредственно содержащаяся в <логическом выражении>, должна однозначно указывать на столбец таблицы (представления) из <FROM-спецификации>.

```
Получить список автовладельцев, у которых предел кредита равен
 $100, и марки их автомобилей:
select B.NAME, A.MODEL from auto A, person B
 where A.personid=B.personid and exists(
         select * from finance C
          where C.personid=A.personid and crditlim=100);

 NAME          MODEL
 ---------------------------------
```

```
|KIM          |BUICK SKYLARK V8    |
|MERTA        |GREMLIN X           |
|SPARCK JONES |MERCURY COMET GT V8 |
|SPIEGEL      |CADILLAC DE VILLE   |
```

2) <Значимое выражение>, заданное в <логическом выражении>, не должно включать в себя ссылок на столбцы, которые являются результатом агрегатной функции.

Недопустимая конструкция:

```
select count(college) as CNT from finance where CNT>3;
```

3) <Значимое выражение>, заданное в <логическом выражении>, может включать в себя значение CURRVAL-последовательности. Использование NEXTVAL запрещено. В этом случае в условии WHERE используется то значение CURRVAL, которое было установлено перед выполнением запроса. Если значение CURRVAL не установлено, будет выдан код завершения 1102 (текущее значение последовательности не установлено).

```
! Например, если используется запрос вида
select "TEST".NEXTVAL, "TEST".CURRVAL from AUTO
where rowid < "TEST".CURRVAL;
! то значения "TEST".CURRVAL в секциях SELECT и WHERE будут
! разными, а именно: первое значение в секции SELECT будет
! отличаться от значения в секции WHERE на величину шага
! последовательности.
DROP SEQUENCE "TEST";
CREATE SEQUENCE "TEST";
!error 1102:
select rowid,"TEST".NEXTVAL, "TEST".CURRVAL
  from AUTO where rowid < "TEST".CURRVAL;
!3 rows with rowid 1,2,3:
select rowid,"TEST".NEXTVAL, "TEST".CURRVAL
  from AUTO where rowid < 4;
!2 rows with rowid 1,2:
select rowid,"TEST".NEXTVAL, "TEST".CURRVAL
  from AUTO where rowid < "TEST".CURRVAL;

! Результат работы примера:
! error 1102:
INL : состояние выполнения : 1102
текущее значение последовательности не установлено
3 rows with rowid 1,2,3:
 ROWID
 -----
|          1|                    1|                    1|
|          2|                    2|                    2|
|          3|                    3|                    3|
INL : выдано строк     : 3
2 rows with rowid 1,2:
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
 ROWID
 -----
|          1|                    4|                    4|
|          2|                    5|                    5|
 INL : выдано строк      : 2
```

То есть после второго SELECT-запроса значение CURRVAL стало равно 3, и именно это значение было использовано в условии WHERE последнего (третьего) запроса.

4) NEXTVAL запрещено в <WHERE-спецификации>.

5) Если <значимое выражение>, непосредственно содержащееся в <логическом выражении>, является агрегатной функцией, то <WHERE-спецификация> должна содержаться в <HAVING-спецификации>, а аргументом этой функции должна быть внешняя ссылка.

Получить список учебных заведений, которые закончили не менее 5 сотрудников, имеющих кредитные карточки AMERICAN EXPRESS:

```
select college from finance
where crditcrd  = 'AMERICAN EXPRESS'
  group by college having count(college) > 5;

 COLLEGE
------------
|ARIZONA     |
|CONNECTICUT |
|RUTGERS     |
|…           |
```

6) Запрещено использование множественных функций в <WHERE-спецификации> без внешнего HAVING-запроса.

7) Числовые значения <значимого выражения>, заданного в <логическом выражении>, допускается представлять в строковом виде.

```
select count(*) from person where age>'40' and salary between
 '20000' and 50000;
```

Общие правила

1) <Логическое выражение> применяется к каждой строке, получаемой в результате <FROM-спецификации>. Результат <WHERE-спецификации> – таблица, состоящая из тех строк, для которых результат <логического выражения> истинен.

2) Подзапрос (из <логического выражения>) выполняется для каждой строки, получаемой в результате <FROM-спецификации>, и его результаты используются при применении к данной строке <логического выражения>. Если подзапрос содержит внешнюю ссылку на столбец, то это будет ссылка на значение этого столбца в данной строке в результате выполнения <FROM-спецификации>.

Получить список владельцев автомобилей черного цвета:

```
select name || ' ' || firstnam from person where exists
(select * from auto
  where person.personid=auto.personid and color = 'BLACK');
```

```
|ADKINSON SYLVIA |
|ADKINSON ED     |
|ADKINSON STEVE  |
|ADKINSON ART    |
|ALDEN ARTHUR    |
|ALDEN MARTHA    |
|ALDEN CLAUDE    |
|ALEXANDER TY    |
```

GROUP BY-спецификация

Функция

Задает сгруппированную таблицу, которая получается путем применения <спецификации группировки> к результату предшествующей <FROM-спецификации>.

Спецификация

[1] <GROUP BY-спецификация>::=
GROUP BY <спецификация группировки> [, …]

[2] <спецификация группировки>::=
{
{[<имя таблицы>.]PK |<элемент группировки>}
| <группировка со сведением данных>
| <группировка по заданным группам>
}

[3] <элемент группировки>::=
<значимое выражение>|[<имя таблицы>.]<имя столбца>|<SQL-параметр>

[4] <группировка со сведением данных>::=
ROLLUP | CUBE (<элемент группировки> [, …])

[5] <группировка по заданным группам>::=
GROUPING SETS ((<группа>) [, …])

[6] <группа>::=
<группировка со сведением данных>
| <элемент группировки> [, …]

Синтаксические правила

1) Каждая <спецификация группировки> в <GROUP BY-спецификации> должна однозначно указывать на <имя столбца> или <значимое выражение> (их синонимы не допустимы), получаемое в результате выполнения предшествующей <FROM-спецификации> или <WHERE-спецификации>.

```
select make from auto where color='BLACK' group by make;
SELECT name, make, MAX(cylnders)
  FROM auto, person
  WHERE auto.personid=person.personid
  GROUP BY name, make;

select year+1900, model
  from auto group by year,model;
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

2) Опция PK задает группировку по первичному ключу таблицы. Это позволяет исключить повторение всех неагрегируемых столбцов записей выборки данных из конструкции GROUP BY.

   Например, если в конструкции GROUP BY col1, col2, col3… столбец col1 является первичным ключом, то столбцы col2, col3… при группировке уже не учитываются, поэтому их можно заменить конструкцией PK.

   Допускается синтаксис <имя таблицы>.PK, так и просто PK для случая одной таблицы.

```
create or replace table
  test (id int primary key, salary int, dept int);

insert into test values(1, 10, 1);
insert into test values(2, 15, 1);
insert into test values(3, 15, 2);
insert into test values(4, 20, 2);
insert into test values(5, 20, 3);
```

Нижеследующие запросы эквивалентны:

```
select id, salary*1000, 'department #' + to_char(dept)
  from test
 group by id,salary,dept
having dept < 3;

select id, salary*1000, 'department #' + to_char(dept)
  from test group by pk
having dept < 3;
```

3) Если в таблице есть и первичный ключ, и столбец с именем "PK", то группировка GROUP BY PK будет производиться по первичному ключу.
4) В <значимом выражении> <спецификации группировки>, соответствующем GROUP BY, не должен содержаться <подзапрос>. Можно использовать конструкции <спецификация значения по условию> (CASE), <спецификация типа> (CAST), знаки операций, скалярные функции и логические выражения.

Примеры.

```
! a)
create or replace table tst (i int, b boolean);
insert into tst values (1,TRUE);
insert into tst values (2,FALSE);
insert into tst values (1,TRUE);
insert into tst values (1,TRUE);

select i, count(b) from tst group by i,2>1;

 I
 -
|          1|          3|
```

```
|          2|          1|

! б) Определить название отдела в зависимости от его номера и
 сгруппировать полученные данные по названию отдела:
select dept_id, case dept_id when 1 then 'testing'
  when 2 then 'development'
  when 3 then 'directing'
  when 4 then 'management'
  else 'common department' end as name_of_department
  from tab_aggr
  group by dept_id, case dept_id when 1 then 'testing'
                    when 2 then 'development'
                    when 3 then 'directing'
                    when 4 then 'management'
                    else 'common department' end;

 DEPT_ID     NAME_OF_DEPARTMENT
 -------     ------------------
|          1|testing           |
|          2|development       |
|          3|directing         |
|          4|management        |
|          5|common department |
|          6|common department |
|          7|common department |
```

5) В <списке выборки> можно указывать целиком <значимое выражение>, по которому делается группировка, или другие столбцы, используемые как аргументы агрегатных функций.

```
select round(weight/1000) as "Вес в тоннах", model
  from auto group by round(weight/1000), model;
```

6) Пользовательскую функцию, содержащую внутри себя <запросы выборки>, можно не указывать в <спецификации группировки>.

```
create or replace table t1( i int, j int );
insert into t1 values(1,4);
insert into t1 values(2,5);
insert into t1 values(2,6);
insert into t1 values(3,6);
insert into t1 values(3,7);

create or replace procedure test(in i int) result int for debug
declare var s cursor( i int );//
code
  open s for direct "select i from t1 where i="+itoa(i)+";";//
  fetch s;//
  return s.i;//
end;
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
! Эти конструкции эквивалентны:
select i, test(i) from t1 group by i, test(i);
select i, test(i) from t1 group by i;

|          1|          1|
|          2|          2|
|          3|          3|

! Аналогичные конструкции с пользовательскими функциями, не
 содержащими <запрос выборки>, недопустимы.
! Не выполняется:
select i, max(5)  from t1 group by i, max(5);
! Выполняется:
select i, (select 5)  from t1 group by i, (select 5);
! Выполняется:
select i, (select 5)  from t1 group by i;
```

7) Операция GROUP BY запрещена для столбцов типа BLOB и EXTFILE.

8) Если <элементом группировки> является <SQL-параметр>, то должен явно указываться тип данных этого параметра.

```
select year+1900, :model (char (20))
  from auto group by year,:model (char (20));
FORD

|       1970|FORD                |
|       1971|FORD                |
```

9) В операции GROUP BY запрещается использование более чем одной функции с атрибутом DISTINCT, кроме случая дублирования одного и того же аргумента функции (например, в SELECT и в HAVING).

Правильная конструкция:

```
select round(avg( cylnders),0), count( model) from auto group by
 color;
```

Недопустимая конструкция (DISTINCT применяется для разных выражений):

```
select round(avg( distinct cylnders),0), count( distinct model)
 from auto group by color;
```

Допустимая конструкция (DISTINCT применяется для одинаковых выражений):

```
select round(avg( distinct (cylnders+length(make))),0),
  count(distinct (cylnders+length(make)))
  from auto group by color;
```

10) На первом месте в списке <групп> должна быть непустая группа.

Недопустимая конструкция:

```
select make from auto group by grouping sets (),make;
```

Общие правила

1) Результат <GROUP BY-спецификации> – разбивка предшествующей <FROM-спецификации> или <WHERE-спецификации> на набор групп. Набор групп – это минимальное число таких групп, в каждой из которых все значения группируемых столбцов равны.

```
select model, count(*) from auto where year= 70 group by model;

| 124 SPORT COUPE | 6 |
| 1302S           | 3 |
| 1600            | 7 |
  …
```

2) Каждая строка полученной группы содержит одно и то же значение каждого группирующего столбца. Если к группе применяется <HAVING-спецификация> или <значимое выражение>, то ссылка на группирующий столбец является ссылкой на это значение.
3) Конструкция <группировка со сведением данных> указывает на то, что помимо стандартных строк, предоставленных предложением GROUP BY, в результирующий набор добавляются сводные (итоговые) строки. Сводная строка возвращается для всех возможных сочетаний групп и подгрупп в результирующем наборе. Имя столбца в сводной строке результирующего набора выводится как NULL-значение. Чтобы определить, представляют ли NULL-значения в результирующем наборе сводные значения или фактические данные, используется функция GROUPING.
4) Опция ROLLUP указывает на то, что помимо стандартных строк, предоставленных GROUP BY, в результирующий набор вводятся сводные строки.

Подсчитать количество автомобилей разного цвета:

```
select color, count(*)
  from auto
  group by color;

 COLOR
 -----
| BLACK         |        262 |
| BLUE          |        108 |
| BROWN         |         46 |
| GREEN         |         43 |
| GREY          |        104 |
| RED           |         52 |
| WHITE         |        323 |
| YELLOW        |         62 |
```

Подсчитать количество автомобилей разного цвета с выдачей сводной (итоговой) суммы можно двумя способами:

а) без конструкции ROLLUP:

```
select color, count(*)
  from auto
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
  group by color
  union
  select 'Итого: ', count(*) from auto;

COLOR
 -----
| BLACK          |        262 |
| BLUE           |        108 |
| BROWN          |         46 |
| GREEN          |         43 |
| GREY           |        104 |
| RED            |         52 |
| WHITE          |        323 |
| YELLOW         |         62 |
| Итого:         |       1000 |
```

б) с использованием конструкции ROLLUP:

```
select color, count(*)
  from auto
  group by rollup(color);

COLOR
 -----
| BLACK          |        262 |
| BLUE           |        108 |
| BROWN          |         46 |
| GREEN          |         43 |
| GREY           |        104 |
| RED            |         52 |
| WHITE          |        323 |
| YELLOW         |         62 |
|                |       1000 |
```

Чтобы итоговая сумма имела название (например, 'Итого:'), надо применить функцию GROUPING.

Подсчитать количество автомобилей разного цвета с выдачей сводной (итоговой) суммы:

```
select case GROUPING(color)
  when 0 then color else 'Итого: '  end as color,
  count(*) from auto group by rollup(color);

COLOR
-----
|BLACK           |        262|
|BLUE            |        108|
```

```
|BROWN            |          46|
|GREEN            |          43|
|GREY             |         104|
|RED              |          52|
|WHITE            |         323|
|YELLOW           |          62|
|Итого:           |        1000|
```

5) При указании опции ROLLUP группы обобщаются в иерархическом порядке, начиная с самого нижнего уровня в группе и заканчивая самым верхним. Иерархия группы определяется порядком, в каком заданы столбцы, по которым производится группирование. Изменение порядка столбцов, по которым производится группирование, может повлиять на количество строк в результирующем наборе.

   Подсчитать количество автомобилей разного цвета, выпущенных разными производителями:

```
select make, color, count(*)
  from auto
  group by rollup(make,color);
```

```
MAKE              COLOR
 ----              -----
| ALPINE            | BLACK  |  1 |
| ALPINE            | GREY   |  3 |
| ALPINE            | WHITE  |  3 |
| AMERICAN MOTORS   | BLACK  | 23 |
| AMERICAN MOTORS   | BLUE   | 11 |
| AMERICAN MOTORS   | BROWN  |  6 |
| AMERICAN MOTORS   | GREEN  |  2 |
| AMERICAN MOTORS   | GREY   |  6 |
```

…

6) Опция CUBE означает, что помимо строк, перечисленных в GROUP BY, в результирующий набор должны включаться сводные (итоговые) строки для всех возможных сочетаний групп и подгрупп результирующего набора (в отличие от опции ROLLUP, которая возвращает итоговую сумму только по одной группе).

7) Количество сводных строк в результирующем наборе определяется по количеству столбцов, включенных в предложение GROUP BY. Каждый операнд (столбец) в предложении GROUP BY привязывается к группирующему NULL-значению, и группирование применяется ко всем остальным операндам (столбцам). Поскольку оператор CUBE возвращает все возможные сочетания групп и подгрупп, количество строк остается тем же, независимо от заданного порядка группирования столбцов.

   Подсчитать количество автомобилей разного цвета, выпущенных разными производителями, с выдачей сводных данных по цветовой гамме:

```
select make, color, count(*)
  from auto
  group by cube(make,color);
```

```
----              -----
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
| ALPINE          | BLACK |  1 |
| ALPINE          | GREY  |  3 |
| ALPINE          | WHITE |  3 |
| AMERICAN MOTORS | BLACK | 23 |
| AMERICAN MOTORS | BLUE  | 11 |
…

(сводные данные по первой группе)
|ALPINE          |       | 7  |
|AMERICAN MOTORS |       | 91 |
|BMW             |       | 10 |
|CHRYSLER        |       | 168|
|CITROEN         |       | 7  |
|DATSUN          |       | 10 |
|DE TOMASO       |       | 12 |
…

(сводные данные по второй группе)
|          |BLACK  |262 |
|          |BLUE   |108 |
|          |BROWN  |46  |
|          |GREEN  |43  |
|          |GREY   |104 |
|          |RED    |52  |
|          |WHITE  |323 |
|          |YELLOW  62
|          |        1000
```

8) Различия между опциями CUBE и ROLLUP:

   - CUBE создает результирующий набор, содержащий статистические выражения для всех комбинаций значений заданных столбцов;
   - ROLLUP создает результирующий набор, содержащий статистические выражения иерархии значений в заданных столбцах.

9) Конструкция <группировка по заданным группам> (GROUPING SETS) позволяет группировать данные по любым произвольным группам или их комбинациям. Если группы или критерии группировки включают более одного столбца или выражения, то группы заключаются в скобки. Пустые скобки используются для обозначения единственной группы, которая охватывает всю таблицу.

10) Использование <GROUP BY-спецификация> с опцией GROUPING SETS эффективно в следующих случаях:

    - необходим только один проход базовой таблицы;
    - не нужно объединение (UNION) сложных операторов;
    - чем больше <групп>, тем больше выигрыш в производительности.

Алгоритм группировки данных:

- количество изолированных групп в результирующей выборке данных равно указанному количеству <групп>;
- создаваемые группы располагаются в выборке данных в той очередности, в какой они перечислены в опции GROUPING SETS;
- количество столбцов в результирующей выборке данных равно сумме не совпадающих столбцов в <элементах группировки>.

Например:

```
GROUPING SETS ((a, b), (b, c) – будут созданы две группы, в
 выборке данных будут присутствовать столбцы  a, b, c.
GROUPING SETS ((a, b), (c, d) – будут созданы две группы, в
 выборке данных будут присутствовать столбцы a, b, c, d
```

- для столбцов группы, у которых нет значений в формируемой результирующей выборке данных, подставляются NULL-значения.

11) <GROUP BY-спецификация> с опцией GROUPING SETS эквивалента полному объединению данных, получаемых из отдельных запросов для каждой группы.

Например:

```
create or replace table "Города" ("Название" char(15),
  "Статус" char(4), "Население, чел." bigint);

insert into "Города" values ('Москва','рспб', 12000000),
  ('Воронеж', 'облс', 1000000), ('Борисоглебск', 'р-он', 400000),
  ('Семилуки', 'пгт', 120000), ('Курск', 'облс', 450000),
  ('Елец', 'р-он', 80000);

! Получить список городов с их численностью населения и суммарное
! количество населения в городах соответствующего статуса.

! Эти запросы эквивалентны:

SELECT "Название", "Статус", SUM("Население, чел.")
  FROM "Города"
 GROUP BY GROUPING SETS (("Название"), ("Статус"));

SELECT "Название", NULL as "Статус", SUM("Население, чел.")
  FROM "Города"
 GROUP BY "Название"
 UNION ALL
SELECT NULL as "Название", "Статус", SUM("Население, чел.")
  FROM "Города" GROUP BY "Статус";

    Название        |   Статус   |                |
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
|   Борисоглебск    |            |      400000.0   |
|   Воронеж         |            |     1000000.0   |
|   Елец            |            |       80000.0   |
|   Курск           |            |      450000.0   |
|   Москва          |            |    12000000.0   |
|   Семилуки        |            |      120000.0   |
|                   |   облс     |     1450000.0   |
|                   |   пгт      |      120000.0   |
|                   |   р-он     |      480000.0   |
|                   |   рспб     |    12000000.0   |
```

Примеры

1)

```
GROUP BY a эквивалентно GROUP BY GROUPING SETS((a))
```

2)

```
GROUP BY a,b,c  эквивалентно GROUP BY GROUPING SETS((a,b,c))
```

3)

```
GROUP BY ROLLUP(a,b) эквивалентно GROUP BY GROUPING SETS((a,b),
(a),())
```

4)

```
GROUP BY CUBE(a,b,c)эквивалентно
GROUP BY GROUPING SETS((a,b,c),
                       (a,b),
                       (a,c),
                       (a),
                       (b,c),
                       (b),
                       (c),
                       ())
```

5)

```
GROUP BY a, ROLLUP(b,c) эквивалентно
GROUP BY GROUPING SETS((a,b,c)
                       (a,b)
                       (a))
```

6)

```
GROUP BY a, b, ROLLUP(c,d) эквивалентно
GROUP BY GROUPING SETS((a,b,c,d),
                       (a,b,c),
                       (a,b) )
```

7)

```
GROUP BY ROLLUP(a), ROLLUP(b,c) эквивалентно
```

```
GROUP BY GROUPING SETS((a,b,c),
                       (a,b),
                       (a),
                       (b,c),
                       (b),
                       () )
```

8)

```
GROUP BY ROLLUP(a), CUBE(b,c) эквивалентно
GROUP BY GROUPING SETS((a,b,c),
                       (a,b),
                       (a,c),
                       (a),
                       (b,c),
                       (b),
                       (c),
                       () )
```

9)

```
GROUP BY CUBE(a,b), ROLLUP(c,d) эквивалентно
GROUP BY GROUPING SETS((a,b,c,d),
                       (a,b,c),
                       (a,b),
                       (a,c,d),
                       (a,c),
                       (a),
                       (b,c,d),
                       (b,c),
                       (b),
                       (c,d),
                       (c),
                       () )
```

10)

```
GROUP BY a, ROLLUP(a,b) эквивалентно
GROUP BY GROUPING SETS((a,b),
                       (a))
```

11) Группировка по первичным ключам разных таблиц:

```
create or replace table
  test1 (i int primary key, j int, ch char(10));
insert into test1 values(0, 1, 't0_1');
insert into test1 values(1, 1, 't1_1');
insert into test1 values(2, 1, 't1_2');
insert into test1 values(3, 3, 't1_3');

create or replace table
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
  test2 (i int, j int, ch char(10), primary key(i,j));
insert into test2 values(1, 1, 't2_1');
insert into test2 values(1, 2, 't2_2');
insert into test2 values(2, 1, 't2_3');
insert into test2 values(3, 1, 't2_4');

select t1.i as t1i, t1.j as t1j, t2.i as t2i, t2.j as t2j,
  min(t1.ch) as mint1ch, max(t2.ch) as maxt2ch
  from test1 t1, test2 t2
  where t1.i=t2.i
  group by t1.pk, t2.pk
  having min(t1.ch) > 't1';
```

```
   T1I       T1J     T2I      T2J    MINT1CH  MAXT2CH
   ---       ---     ---      ---    -------  -------
|      1|       1|       1|       1|t1_1    |t2_1    |
|      1|       1|       1|       2|t1_1    |t2_2    |
|      2|       1|       2|       1|t1_2    |t2_3    |
|      3|       3|       3|       1|t1_3    |t2_4    |
```

12) Группировка данных по двум группам:

```
select model as "Модели автомобилей",
  color as "Окраска автомобилей"
  from auto
  group by grouping sets ((model),(color));
```

```
Модели автомобилей   Окраска автомобилей
 ------------------   -------------------
|124 SPORT COUPE     |                   |
|1275 GT             |                   |
|1302 S              |                   |
|1600                |                   |
…
|SM                  |                   |
|TR 6                |                   |
|XJ 6 4.2            |                   |
|                    |BLACK              |
|                    |BLUE               |
|                    |BROWN              |
|                    |GREEN              |
|                    |GREY               |
|                    |RED                |
|                    |WHITE              |
|                    |YELLOW             |
```

13) Использование функции `GROIPING` и конструкции `GROUPING SETS`:

```
select "Название", "Статус",
```

```
/* выдает битовую маску: первый бит - если значение агрегировано
 */
/* по всем "Городам", второй - если по всем "Статусам" */
  grouping ("Название","Статус")
  from "Города"
 group by
/* группировка идентична CUBE, но расписан список группирующих
 множеств */
   grouping sets (("Название","Статус"),("Название"),("Статус"),
());
```

```
Название        Статус
 --------        ------
|Борисоглебск   |р-он  |    0|
|Воронеж        |облс  |    0|
|Елец           |р-он  |    0|
|Курск          |облс  |    0|
|Москва         |рспб  |    0|
|Семилуки       |пгт   |    0|
|Борисоглебск   |      |    1|
|Воронеж        |      |    1|
|Елец           |      |    1|
|Курск          |      |    1|
|Москва         |      |    1|
|Семилуки       |      |    1|
|               |облс  |    2|
|               |пгт   |    2|
|               |р-он  |    2|
|               |рспб  |    2|
|               |      |    3|
```

HAVING-спецификация

Функция

Задает ограничение для сгруппированной таблицы, полученной путем применения предшествующей <FROM-спецификации>, <WHERE-спецификации> или <GROUP BY-спецификации>, которое состоит в том, что исключаются группы, не удовлетворяющие <логическому выражению>.

Спецификация

[1] <HAVING-спецификация> ::= HAVING <логическое выражение>

Синтаксические правила

1) Каждый элемент <логического выражения> (<спецификация столбца>, <значимое выражение> или <SQL-параметр>) должен однозначно указывать на группируемый столбец из результата предшествующих <FROM-спецификации>, <WHERE-спецификации> или <GROUP BY-спецификации>.

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

2) Каждая <спецификация столбца>, содержащаяся в <подзапросе> <логического выражения> и указывающая на внешний столбец, должна быть группирующим столбцом предшествующих <FROM-спецификации>, <WHERE-спецификации> или <GROUP BY-спецификации> либо аргументом агрегатной функции.

```
select distinct make from auto
 group by make
having max(year) > (select min(auto.year)
                      from auto, person
                     where person.personid = auto.personid
                       and length(auto.make) =
 length(person.name));
```

3) Запрещено обращение из подзапроса в <HAVING-спецификации> к негруппируемым столбцам.

4) Числовые значения <значимого выражения>, заданного в <логическом выражении>, допускается представлять в строковом виде.

```
select name, count(*) from person
 group by name having max(salary) > '50000';
```

Общие правила

1) Если предшествующая спецификация не содержит <GROUP BY-спецификацию>, то результатом <HAVING-спецификации> является единственная группа, а группируемый столбец отсутствует.

2) <Логическое выражение> применяется к каждой группе из результата предшествующих спецификаций. Результатом <HAVING-спецификации> будет сгруппированная таблица, в которую войдут те группы, для которых результатом <логического выражения> будет значение «истина».

3) Когда <логическое выражение> наложено на группу, то эта группа является аргументом любой агрегатной функции, непосредственно содержащейся в <логическом выражении>, если только <спецификация столбца> в агрегатной функции не является внешней ссылкой.

4) Все <подзапросы> в <логическом выражении> выполняются для каждой группы, а полученные результаты применяются к каждой группе, удовлетворяющей <логическому выражению>. Если <подзапрос> содержит внешнюю ссылку на столбец, то она является ссылкой на значение этого столбца в данной группе.

```
select make from auto
 group by make
having max(year+1900) <
         (select to_number(to_char(sysdate, 'yyyy')));
```

5) В <HAVING-спецификации> можно указывать только те столбцы и значимые выражения, по которым делается группировка.

Получить список производителей машин черного, белого и зеленого цвета с числом цилиндров больше 6:

```
select make, cylnders, color from auto
 where color in ('BLACK','WHITE','GREEN')
 group by make, cylnders, color
```

```
having max(cylnders) > 6;
```

6) Если в <HAVING-спецификации> используется <SQL-параметр>, то его тип данных должен быть указан обязательно.

```
select distinct model, :year (int)+1900 from auto
 group by model, year+1900
having :year (int)+1900 > 1970 limit 3;
71

|124 SPORT COUPE     |       1971|
|1275 GT             |       1971|
|1302 S              |       1971|
```

7) В <списке выборки> можно указывать также целиком <значимое выражение>, по которому создается группировка, или другие столбцы, используемые как аргументы агрегатных функций.

```
select make, year + 1900 from auto
 group by make, year + 1900
having max(year + 1900) between 1969 and 1972;
```

8) Если запрос с <HAVING-спецификацией> не содержит опции GROUP BY, то столбцы в SELECT и в HAVING конструкциях могут использоваться только как аргументы агрегатных функций.

Допустимый запрос:

```
select avg(salary) from tab_aggr having max(salary) > 20000;
```

Недопустимые запросы:

```
select salary from tab_aggr having max(salary) > 20000;
select avg(salary) from tab_aggr having salary > 20000;
```

9) В <HAVING-спецификации> разрешено использовать группированные CAST- и CASE-выражения.

```
select make, cast(year+1900 as double)  from auto
 group by make, cast(year+1900 as double)
having cast(year+1900 as double) > 1970.05;
```

ORDER BY-спецификация

Функция

Определение порядка сортировки выбранных данных.

Спецификация

[1] <ORDER BY-спецификация>::=
ORDER [SIBLINGS] BY <порядок сортировки> [, …]
[2] <порядок сортировки>::=
<ключ сортировки> [<тип сортировки>][<сортировка NULL-значений>]
[3] <ключ сортировки>::=
<беззнаковое целое> | <имя столбца> | <значимое выражение>
[4] <тип сортировки>::= <по возрастанию> | <по убыванию>
[5] <по возрастанию>::= ASC

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

[6] <по убыванию> : : = DESC
[7] <сортировка NULL-значений> : : = NULLS {FIRST | LAST}

Синтаксические правила

1) Опция SIBLINGS применяется только к иерархическим запросам (см. пункт «Иерархический запрос»).
2) Если <запрос выборки> содержит <ORDER BY-спецификацию> и относится к одной таблице (обновляемому представлению), то результатом выполнения <запроса выборки> будет обновляемая таблица с заданным порядком упорядочения.
3) Каждое <имя столбца> в <ORDER BY-спецификации> должно идентифицировать столбец <запроса выборки>.

```
select * from auto order by model;
select distinct model, make from auto order by model desc;
select distinct model, make, cylnders
  from auto order by make asc, cylnders desc;
```

4) <Значимое выражение> может быть логическим выражением.

```
create or replace table tst (i int, b boolean);
insert into tst values (1,TRUE);
insert into tst values (2,FALSE);
insert into tst values (1,TRUE);
insert into tst values (1,TRUE);

select * from tst order by not b;

 I B
- -
|1|T|
|1|T|
|1|T|
|2|F|
```

5) Именованный столбец может быть указан с помощью порядкового номера или по <имени столбца>. Неименованный столбец идентифицируется указанием:
   - <беззнакового целого>

```
select distinct model, make,
  to_char(weight) || ' кг.' "Вес", year+1900 as "Год выпуска"
  from auto
  where rowid = 1 order by model, 3,4;

 MODEL                MAKE   Вес         Год выпуска
|MERCURY COMET GT V8 |FORD  |2900 кг.  |1971         |
```

   - или псевдонима

```
create or replace table t(i int);
insert into t (i) values (1),(2),(1),(2),(3);
```

```
select i,sum(i) as "Сумма" from t group by i order by "Сумма";

 I           Сумма
 -           -----
|          1|                    2.0         |
|          3|                    3.0         |
|          2|                    4.0         |
```

6) Каждое <беззнаковое целое> должно быть больше 0 и не больше числа столбцов таблицы, порождаемой <запросом выборки>.

7) Если не указан вид сортировки, то по умолчанию выполняется сортировка по возрастанию.

8) <Порядок сортировки> определяет относительную значимость упорядочиваемых столбцов. Сначала результат выборки упорядочивается по первому столбцу. Затем полученное упорядоченное множество повторно упорядочивается по второму столбцу и т.д.

```
create table tab1 ( c1 varchar(10), c2 varchar(10));
insert into tab1 values ('23456', 'abcd');
insert into tab1 values ('13456', 'bacd');
insert into tab1 values ('32456', 'cdba');
insert into tab1 values ('13456', 'abcd');
select * from tab1;

| 23456 | abcd  |
| 13456 | bacd  |
| 32456 | cdba  |
| 13456 | abcd  |

select * from tab1 order by 1;

| 13456 | bacd  |
| 13456 | abcd  |
| 23456 | abcd  |
| 32456 | cdba  |

select * from tab1 order by 1,2;

| 13456 | abcd  |
| 13456 | bacd  |
| 23456 | abcd  |
| 32456 | cdba  |
```

9) Опция <сортировка NULL-значений> задаёт местоположение NULL-значений в отсортированном столбце: FIRST – перед отсортированными данными, LAST – в конце. Если опция не задана, по умолчанию используется LAST.

```
create or replace table test(c char(1), i int);
insert into test (c, i) values('a', 1);
insert into test (c, i) values('b', null);
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
insert into test (c, i) values(null, 3);
insert into test (c, i) values('d', null);
insert into test (c, i) values(null, 5);
select * from test order by c nulls first, i desc nulls last;

|NULL    | 5     |
|NULL    | 3     |
|a       | 1     |
|b       | NULL  |
|d       | NULL  |
```

10) Если опция <сортировка NULL-значений> не задана, NULL-значения сортируются так, как если бы они имели наибольшее значение в выборке. Т.е. в случае сортировки по возрастанию (неявной или с использованием спецификации ASC) по умолчанию используется вариант NULLS LAST, в случае сортировки по убыванию (спецификация DESC) по умолчанию используется вариант NULLS FIRST.
11) Операция ORDER BY запрещена для столбцов типа BLOB.
12) Операция ORDER BY разрешена в подзапросах.

OVER-спецификация

Функция

Определение разбивки на группы результирующей выборки данных.

Спецификация

[1] <OVER-спецификация>::=
OVER ([PARTITION BY <значимое выражение> [, …]]
[<ORDER BY-спецификация>])

Синтаксические правила

1) <Значимое выражение> должно быть именем столбца. Использование псевдонимов или выражений не допускается.
2) В <запросе выборки> может использоваться только одна <OVER-спецификация>, то есть запрос вида

```
select (rank() over (order by rowid)) /
       (rank() over (order by rowid desc))
  from auto where rowid < 10;
```

выполняться не будет. Это ограничение можно обойти следующим образом:

```
select a/b from
  (select (rank() over (order by rowid)) a,
  (rank() over (order by rowid desc)) b
  from auto where rowid < 10);
```

Примечание
Для агрегатных функций такого ограничения нет, то есть приведенный ниже запрос будет выполнен:

```
select sum(personid) over (partition by make) /
```

```
       sum(personid) over (partition by model)
  from auto where rowid < 10;
```

Общие правила

1) Конструкция PARTITION BY выполняет разбивку результирующего набора данных на разделы с целью последующего применения к ним агрегатных или аналитических функций.
2) Если конструкция PARTITION BY опущена, то <OVER-спецификация> распространяется на весь результирующий набор данных.
3) <ORDER BY-спецификация> задает логический порядок, в котором должны выполняться вычисления применяемой к разделу данных агрегатной или аналитической функции (а не порядок представления результата, как обычная <ORDER BY-спецификация>).
4) Если <ORDER BY-спецификация> опущена, то <OVER-спецификация> распространяется на весь результирующий набор данных.
5) В тексте запроса перед <OVER-спецификацией> должна указываться аналитическая функция, применяемая к получаемым наборам данных.

```
select personid,
  make,
  lag(make)
  over (partition by model, bodytype order by personid desc)
  from auto;
```

6) Если задана опция DISTINCT, то она применяется к результату OVER (а не наоборот).

```
select distinct first_value(InId)
  over (order by nvl(s.n1,0) desc,
  nvl(s.n2,0) desc,
  nvl(s.n3,0) desc) as InId1 from temp_table s;
```

WITHIN GROUP-спецификация

Функция

Задает порядок группировки записей при обработке агрегатных функций.

Спецификация

[1] <WITHIN GROUP-спецификация>::=
WITHIN GROUP (<ORDER BY-спецификация>)

Синтаксические правила

1) Использование <WITHIN GROUP-спецификации> совместно с <OVER-спецификацией> недопустимо.
2) <WITHIN GROUP-спецификация> может применяться как при наличии, так и при отсутствии в <запросе выборки> <GROUP BY-спецификации>.
3) Количество <WITHIN GROUP-спецификаций> в <запросе выборки> не ограничено, однако при наличии в <запросе выборки> <GROUP BY-спецификации> все <ORDER BY-спецификация> внутри <WITHIN GROUP-спецификации> должны быть идентичны.

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
Допустимый запрос (конструкции within group (order by model)
 одинаковы у обеих агрегатных функций):

select bodytype,
  FIRST_VALUE(model) within group (order by model) as first_val,
  LAST_VALUE(model) within group (order by model) as last_val
  from auto group by bodytype;

Недопустимые запросы (конструкции within group не идентичны):

    а)
select bodytype,
  FIRST_VALUE(model) within group (order by model) as first_val,
  LAST_VALUE(model) within group (order by color) as last_val
  from auto group by bodytype;

    б)
select FIRST_VALUE(make) within group (order by make asc) as
 first_val,
  LAST_VALUE(make) within group (order by make desc) as last_val
  from auto group by make;
```

Общие правила

1) При наличии в запросе <GROUP BY-спецификации> сортировка записей в соответствии с конструкцией <WITHIN GROUP-спецификация> будет осуществляться отдельно внутри каждой группы.
2) При отсутствии в запросе <GROUP BY-спецификации> сортироваться будут все записи выборки.
3) Порядок сортировки аргументов важен только для агрегатных функций (FIRST_VALUE, LAST_VALUE, LISTAGG), результат которых зависит от заданной сортировки. Для агрегатных функций, возвращающих количественные значения (SUM, COUNT и другие), порядок сортировки не имеет значения, поэтому для них конструкция <WITHIN GROUP-спецификация> игнорируется.

```
select count(*),
  FIRST_VALUE(model)
    within group (order by year, model, color) as first_val,
  LAST_VALUE(model)
    within group (order by year, model, color) as last_val
  from auto group by make;

             FIRST_VAL             LAST_VAL
             ---------             --------
|          7|A-310                |A-310                |
|         91|AMBASSADOR SST V8    |MATADOR STATION      |
|         10|3.0 CSI              |3.0 CSI              |
|        168|DODGE CHALLENGER SIX|PLYMOUTH ROAD RUNNER|
```

...

Соединение таблиц

Функция

Определение правил соединения таблиц.

Спецификация

[1] <соединение таблиц> : : =
<соединяемая таблица> [<тип соединения>] JOIN <соединяемая таблица>
[<условие соединения>]
[2] <соединяемая таблица> : : =
<табличная ссылка> | (<табличная ссылка>)
[3] <условие соединения> : : =
<ON-спецификация> | <USING-спецификация>
[4] <ON-спецификация> : : = ON <логическое выражение>
[5] <USING-спецификация> : : = USING (<имя столбца> [,...])
[6] <тип соединения> : : =
<внутреннее соединение>
| <внешнее соединение>
| <естественное соединение>
| <перекрестное соединение>
| <консолидированное соединение>
[7] <внутреннее соединение> : : = [NATURAL]INNER
[8] <внешнее соединение> : : =
[NATURAL]
<левостороннее внешнее соединение>
| <правостороннее внешнее соединение>
| <полное внешнее соединение>
[9] <естественное соединение> : : = NATURAL
[10] <перекрестное соединение> : : = CROSS
[11] <консолидированное соединение> : : = UNION
[12] <левостороннее внешнее соединение> : : = LEFT [OUTER]
[13] <правостороннее внешнее соединение> : : = RIGHT [OUTER]
[14] <полное внешнее соединение> : : = FULL [OUTER]

Синтаксические правила

1) В `<ON-спецификации>` разрешены ссылки только к таблицам соединения (внешние ссылки запрещены).

2) В случае использования `<ON-спецификации>` `<логическое выражение>` не может быть `<предикатом внешнего соединения в стиле ORACLE>` или включать его в себя как составную часть.

3) Для соединения таблиц без использования оператора JOIN используется `<WHERE-спецификация>`, в которой `<логическое выражение>` может быть `<предикатом внешнего соединения в стиле ORACLE>` или включать его в себя как составную часть.

4) `<USING-спецификация>` используется для указания списка имен столбцов, по значениям которых должно выполняться соединение (столбцы с указанными именами должны содержаться в обеих соединяемых таблицах), при этом результирующий набор содержит только один столбец для каждой пары одноименных столбцов исходных таблиц.

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

Примечание
В отличие от NATURAL JOIN, при котором соединение таблиц выполняется по всем одноименным столбцам, использование <USING-спецификации> позволяет ограничиться отдельными одноименными столбцами.

5) Если <тип соединения> не указан, но <условие соединения> присутствует, по умолчанию используется <внутреннее соединение> (INNER JOIN).

```
Эти конструкции эквивалентны:
SELECT * FROM A INNER JOIN B ON A.X=B.X;
SELECT * FROM A JOIN B ON A.X=B.X;
SELECT * FROM A, B WHERE A.X=B.X;
```

6) Если <тип соединения> и <условие соединения> не указаны, то по умолчанию используется <перекрестное соединение> (CROSS JOIN).

```
Эти конструкции эквивалентны:
select count(*) from auto cross join person;
select count(*) from auto join person;
select count(*) from auto, person;
```

7) Если указаны [INNER] JOIN, LEFT [OUTER] JOIN, RIGHT [OUTER] JOIN, FULL [OUTER] JOIN без ключевого слова NATURAL и без <условия соединения>, то результат соединения будет таким же, как для CROSS JOIN.
8) Для <естественного соединения> (NATURAL JOIN) и <перекрестного соединения> (CROSS JOIN) <ON-спецификация> и <USING-спецификация> недопустимы.
9) Каждая таблица, входящая в <соединение таблиц>, имеет своей областью видимости весь подзапрос, содержащий это соединение, за исключением той части <FROM-спецификации>, которая не содержит соединения, включающего данную таблицу.
10) При соединении таблиц следует различать использование <условия соединения> (<ON-спецификация>, <USING-спецификация>), которое непосредственно определяет условие, по которому будут соединены таблицы, и <WHERE-спецификации>, которая ограничивает выборку, полученную в результате соединения.

```
Тестовые таблицы:
create or replace table tab1 (i int, ch char(1));
insert into tab1 (i,ch) values (1, 'a');
insert into tab1 (i,ch) values (2, 'b');
insert into tab1 (i,ch) values (3, 'c');
insert into tab1 (i,ch) values (4, 'd');

create or replace table tab2 (j int, cm char(1));
insert into tab2 (j,cm) values (2, 'b');
insert into tab2 (j,cm) values (4, 'g');
insert into tab2 (j,cm) values (5, 'e');
insert into tab2 (j,cm) values (7, 'f');

a)
Наличие <ON-спецификации> и <WHERE-спецификации>
```

```
SELECT * FROM tab1 LEFT JOIN tab2 ON tab1.i=tab2.j WHERE (i > 2);
```

Сначала выполняется условие соединения ON (ищутся строки, имеющие равные значения столбцов i и j), в результате которого получается выборка:

```
 I           CH J           CM
 -           -- -           --
|          1|a |            |  |
|          2|b |          2|b |
|          3|c |            |  |
|          4|d |          4|g |
```

Затем к полученной выборке применяется условие WHERE (строки, для которых i > 2), получаем результирующий набор данных:

```
 I           CH J           CM
 -           -- -           --
|          3|c |            |  |
|          4|d |          4|g |
```

б)
Отсутствие <WHERE-спецификации> (выполняется условие соединения, включающее в себя два логических выражения):

```
SELECT * FROM tab1 LEFT JOIN tab2 ON (tab1.i=tab2.j) AND (i > 2);
```

```
 I           CH J           CM
 -           -- -           --
|          1|a |            |  |
|          2|b |            |  |
|          3|c |            |  |
|          4|d |          4|g |
```

в)
При отсутствии <ON-спецификации> соединение LEFT JOIN эквивалентно CROSS JOIN (сначала формируется полное декартово произведение строк, затем применяется условие WHERE)

```
SELECT * FROM tab1 LEFT JOIN tab2 WHERE (tab1.i=tab2.j) AND (i >
 2);
```

```
 I           CH J           CM
 -           -- -           --
|          4|d |          4|g |
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

11) Запрещено повторение имен таблиц и заменяющих их имен в одной <FROM-спецификации>. Например, конструкции вида:

```
SELECT ... FROM A, A ...
SELECT ... FROM A, B AS A ...
SELECT ... FROM B AS A, A ...
SELECT ... FROM B AS A, C AS A ...
```

являются недопустимыми.

Общие правила

1) Соединение двух и более таблиц используется для сопоставления строк одной таблицы строкам другой таблицы, т.е. позволяет выводить информацию из нескольких таблиц, связывая их по значениям определенных полей.
2) Определение того, какие именно исходные строки войдут в результирующий набор и в каких сочетаниях, зависит от типа соединения и от заданного условия соединения. Условие соединения (<ON-спецификация>), то есть условие сопоставления строк исходных таблиц друг с другом, представляет собой логическое выражение.
3) Конструкция <внутреннее соединение> (INNER JOIN) создает результирующий набор, содержащий пары строк двух таблиц, для которых выполняется <условие соединения>. Если <условие соединения> не указано, результирующий набор будет содержать полное декартово произведение строк двух таблиц (тип соединения CROSS JOIN).

```
Тестовые данные:
Пусть имеются таблицы сотрудников (persons) и отделов
 (departments), соединенные между собой по номеру отдела (d_id).

create or replace table departments (d_id int, d_name char(20));
insert into departments values (1, 'Sales');
insert into departments values (2, 'IT-technologies');
insert into departments values (3, 'Finance');
insert into departments values (4, 'Management');
insert into departments values (5, 'Design');

create or replace table persons (p_id int, p_name char (20), d_id
 int);
insert into persons values (1, 'John',  3);
insert into persons values (2, 'Mary',  2);
insert into persons values (3, 'Kate',  4);
insert into persons values (4, 'Jack',  2);
insert into persons values (5, 'Peter', 7);
insert into persons values (6, 'Ann',   5);

Получить список сотрудников, работающих в имеющихся отделах.

Эти конструкции эквивалентны:
```

```
SELECT p.p_name, d.d_name FROM persons p, departments d WHERE
 p.d_id = d.d_id;
SELECT p.p_name, d.d_name FROM persons p INNER JOIN departments d
 USING (d_id);
SELECT p.p_name, d.d_name FROM persons p INNER JOIN departments d
 ON p.d_id = d.d_id;
```

```
  P_NAME              D_NAME
  ------              ------
 |John                |Finance             |
 |Mary                |IT-technologies     |
 |Kate                |Management          |
 |Jack                |IT-technologies     |
 |Ann                 |Design              |
```

```
SELECT * FROM A INNER JOIN B ON <логическое выражение1>
 WHERE <логическое выражение2>
есть то же самое, что
SELECT * FROM A, B WHERE <логическое выражение1> AND <логическое
 выражение2>
```

Примечание
Здесь и далее в разделе обозначения A, B представляют собой табличные выражения.

4) При внешнем соединении, в отличие от внутреннего, в результат выборки попадают не только все связанные строки обеих таблиц, но и строки одной из таблиц (или обеих), для которых нет связанных строк в другой таблице. Недостающим значениям столбцов другой таблицы при этом присваивается значение NULL.

5) Конструкция <левостороннее внешнее соединение> (LEFT [OUTER] JOIN) возвращает результирующий набор, содержащий:

   - пары строк двух таблиц, для которых выполняется <условие соединения>;
   - все строки левой таблицы, для которых не нашлось по <условию соединения> соответствующих строк в правой таблице, дополненные NULL-значениями в столбцах результата, соответствующих столбцам правой таблицы.

```
Примеры с использованием тестовых данных, описанных ранее:

a)
Получить полный список сотрудников и соответствующих отделов, в
 которых они работают, включая сотрудников, для которых отделы не
 указаны.

Эти конструкции эквивалентны:

SELECT p.p_name, d.d_name FROM persons p LEFT JOIN departments d
 ON p.d_id = d.d_id;
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
SELECT p.p_name, d.d_name FROM persons p LEFT JOIN departments d
 USING (d_id);
SELECT p.p_name, d.d_name FROM persons p, departments d WHERE
 p.d_id = d.d_id(+);

 P_NAME          D_NAME
 ------           ------
|John            |Finance             |
|Mary            |IT-technologies     |
|Kate            |Management          |
|Jack            |IT-technologies     |
|Peter           |                    |
|Ann             |Design              |
```

б)
Получить полный список отделов и работающих в них сотрудников, включая отделы, для которых не указано ни одного сотрудника.

Эти конструкции эквивалентны:

```
SELECT d.d_name, p.p_name FROM departments d LEFT JOIN persons p
 ON p.d_id = d.d_id;
SELECT d.d_name, p.p_name FROM persons p, departments d WHERE
 p.d_id(+) = d.d_id;
SELECT d.d_name, p.p_name FROM departments d LEFT JOIN persons p
 USING (d_id);

 D_NAME               P_NAME
 ------               ------
|Sales               |           |
|IT-technologies     |Mary       |
|IT-technologies     |Jack       |
|Finance             |John       |
|Management          |Kate       |
|Design              |Ann        |
```

Примечание
Оператор (+) ставится у таблицы, в которой могут отсутствовать строки соответствия и которая, соответственно, дополняется записями с NULL-значениями.

6) Для получения результата <левостороннего внешнего соединения> оператор (+) ставится у правой таблицы (см. пункт «Предикат внешнего соединения в стиле ORACLE»).

```
SELECT * FROM A LEFT OUTER JOIN B ON <логическое выражение> есть
 то же самое, что
SELECT * FROM A, B WHERE <логическое выражение>
 UNION
```

```
SELECT A.*, <пустые значения> FROM A
 WHERE NOT EXISTS (SELECT * FROM B WHERE <логическое выражение>)
```

Примечание
Под <пустыми значениями> в этом и следующих примерах понимаются NULL-значения в таком количестве и таких типов, чтобы содержащий их запрос совпадал по количеству и типам значений с тем запросом, с которым он объединяется по UNION.

7) Конструкция <правостороннее внешнее соединение> (RIGHT [OUTER] JOIN) возвращает результирующий набор, содержащий:
   - пары строк двух таблиц, для которых выполняется <условие соединения>;
   - все строки правой таблицы, для которых не нашлось по <условию соединения> соответствующих строк в левой таблице, дополненные NULL-значениями в столбцах результата, соответствующих столбцам левой таблицы.
8) Конструкция RIGHT [OUTER] JOIN работает так же, как и LEFT [OUTER] JOIN, с той лишь разницей, что результирующий набор формируется по правой таблице.
9) Конструкции типа «A LEFT OUTER JOIN B» и «B RIGHT OUTER JOIN A» являются эквивалентными.

```
Эти 2 конструкции эквивалентны:
SELECT tab1.* FROM tab1 LEFT JOIN tab2 WHERE tab1.i=tab2.j;
SELECT tab1.* FROM tab2 RIGHT JOIN tab1 WHERE tab1.i=tab2.j;
```

10) Для получения результата <правостороннего внешнего соединения> в <WHERE-спецификации> оператор (+) ставится у левой таблицы (см. пункт «Предикат внешнего соединения в стиле ORACLE»).

```
Примеры с использованием тестовых данных, описанных ранее:
a)
Получить полный список отделов и работающих в них сотрудников,
 включая отделы, для которых не указано ни одного сотрудника.

Эти конструкции эквивалентны:

SELECT d.d_name, p.p_name FROM persons p RIGHT JOIN departments d
 ON p.d_id = d.d_id;
SELECT d.d_name, p.p_name FROM persons p RIGHT JOIN departments d
 USING (d_id);

SELECT d.d_name, p.p_name FROM departments d LEFT JOIN persons p
 ON p.d_id = d.d_id;
SELECT d.d_name, p.p_name FROM persons p, departments d WHERE
 p.d_id(+) = d.d_id;

 D_NAME                P_NAME
 ------                ------
|Sales                 |
|IT-technologies       |Mary
|IT-technologies       |Jack
|Finance               |John
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
|Management           |Kate           |
|Design               |Ann            |
```

```
б)
Получить полный список сотрудников и соответствующих отделов, в
 которых они работают, включая сотрудников, для которых отдел не
 указан.

Эти конструкции эквивалентны:

SELECT p.p_name, d.d_name FROM departments d RIGHT JOIN persons p
 ON p.d_id = d.d_id;
SELECT p.p_name, d.d_name FROM departments d RIGHT JOIN persons p
 USING (d_id);

SELECT p.p_name, d.d_name FROM persons p LEFT JOIN departments d
 ON p.d_id = d.d_id;
SELECT p.p_name, d.d_name FROM persons p, departments d WHERE
 d.d_id(+) = p.d_id;

 P_NAME               D_NAME
 ------               ------
|John                 |Finance            |
|Mary                 |IT-technologies    |
|Kate                 |Management         |
|Jack                 |IT-technologies    |
|Peter                |                   |
|Ann                  |Design             |

в)
SELECT * FROM A RIGHT OUTER JOIN B ON <логическое выражение>

есть то же самое, что:

SELECT * FROM A, B WHERE <логическое выражение>
 UNION
SELECT <пустые значения>, B.*  FROM B
 WHERE NOT EXISTS (SELECT * FROM A WHERE <логическое выражение>)
```

11) Конструкция <полное внешнее соединение> FULL [OUTER] JOIN возвращает все строки обеих соединяемых таблиц. Если строке из одной таблицы нет соответствующей строки в другой таблице, возвращается NULL-значение.

12) Для получения результата <полного внешнего соединения> в <WHERE-спецификации> оператор (+) ставится у обеих таблиц: и у левой и у правой (см. пункт «Предикат внешнего соединения в стиле ORACLE»).

```
Примеры с использованием тестовых данных, описанных ранее:
а)
```

```
Получить полный список имеющихся сотрудников и полный список
 имеющихся отделов с соответствием между сотрудниками и отделами.

Эти конструкции эквивалентны:

SELECT p.p_name, d.d_name FROM persons p FULL JOIN departments d
 ON p.d_id = d.d_id;
SELECT p.p_name, d.d_name FROM persons p FULL JOIN departments d
 USING (d_id);
SELECT p.p_name, d.d_name FROM persons p, departments d WHERE
 p.d_id(+) = d.d_id(+);

  P_NAME            D_NAME
 ------              ------
|                  |Sales              |
|Mary              |IT-technologies    |
|Jack              |IT-technologies    |
|John              |Finance            |
|Kate              |Management         |
|Ann               |Design             |
|Peter             |                   |

б)
SELECT * FROM A FULL OUTER JOIN B ON <логическое выражение>

есть то же самое, что

SELECT * FROM A, B WHERE <логическое выражение>
 UNION
SELECT A.*, <пустые значения> FROM A
 WHERE NOT EXISTS (SELECT * FROM B WHERE <логическое выражение>)
 UNION
SELECT <пустые значения>, B.* FROM B
 WHERE NOT EXISTS (SELECT * FROM A WHERE <логическое выражение>)
```

13) Конструкция <естественное соединение> NATURAL [INNER | {LEFT | RIGHT | FULL} [OUTER] | UNION] JOIN – это эквисоединение, при котором автоматически (столбцы специально не указываются) происходит сравнение всех столбцов в обеих таблицах, которые имеют одинаковые имена.
14) Если соединяемые таблицы не имеют совпадающих по именам столбцов, то результат <естественного соединения> будет эквивалентен результату <перекрестного соединения> CROSS JOIN.
15) Условием для выполнения <естественного соединения> служит наличие идентичных значений в совпадающих столбцах. Результирующий набор содержит только один столбец для каждой пары одноименных столбцов.
16) Выражения NATURAL [INNER | {LEFT | RIGHT | FULL} [OUTER] | UNION] JOIN для двух таблиц определяется как семантический эквивалент INNER JOIN, LEFT JOIN, RIGHT JOIN, FULL JOIN или UNION JOIN с выражением USING, в котором указаны все совпадающие столбцы обеих таблиц.

Примеры с использованием тестовых данных, описанных ранее:
а)
Получить список сотрудников, работающих в имеющихся отделах.

Эти конструкции эквивалентны:

```
SELECT p.p_name, d.d_name FROM persons p NATURAL JOIN departments
 d;
SELECT p.p_name, d.d_name FROM persons p INNER JOIN departments d
 USING (d_id);
SELECT p.p_name, d.d_name FROM persons p JOIN departments d USING
 (d_id);
```

```
 P_NAME               D_NAME
 ------               ------
|John                |Finance             |
|Mary                |IT-technologies     |
|Kate                |Management          |
|Jack                |IT-technologies     |
|Ann                 |Design              |
```

б)
Получить полный список сотрудников и соответствующих отделов, в которых они работают, включая сотрудников, для которых отдел не указан.

Эти конструкции эквивалентны:

```
SELECT p.p_name, d.d_name FROM persons p NATURAL LEFT JOIN
 departments d;
SELECT p.p_name, d.d_name FROM persons p LEFT JOIN departments d
 USING (d_id);
```

```
 P_NAME               D_NAME
 ------               ------
|John                |Finance             |
|Mary                |IT-technologies     |
|Kate                |Management          |
|Jack                |IT-technologies     |
|Peter               |                    |
|Ann                 |Design              |
```

в)
```
SELECT * FROM A NATURAL JOIN B;
```
есть то же самое, что

```
SELECT * FROM A INNER JOIN B USING (<столбцы>),

г)
SELECT * FROM A NATURAL {LEFT | RIGHT| FULL} [OUTER] JOIN B;
есть то же самое, что
SELECT * FROM A {LEFT | RIGHT| FULL}[OUTER] JOIN B USING
 (<столбцы>),

д)
SELECT * FROM A NATURAL UNION JOIN B;
есть то же самое, что
SELECT * FROM A UNION JOIN B USING (<столбцы>),

В примерах в)-д) под <столбцами> подразумевается перечисление имен
 совпадающих
 столбцов соединяемых таблиц.
```

17) Конструкция <перекрестное соединение> CROSS JOIN возвращает полное декартово произведение строк двух таблиц, т.е. каждая строка одной таблицы будет соединяться с каждой строкой другой таблицы. Количество записей в результирующем наборе будет равно произведению числа записей в соединяемых таблицах.

```
select count(*) from auto;

| 1001|

select count(*) from person;

| 987|

1001*987=987987

select count(*) from auto cross join person;

| 987987|


Пример с использованием тестовых данных, описанных ранее:
Получить полный список сотрудников со всеми возможными вероятными
 отделами, или, что то же самое, получить полный список отделов со
 всеми возможными вероятными сотрудниками.

Эти конструкции эквивалентны:

SELECT p.p_name, d.d_name FROM persons p CROSS JOIN departments d;
SELECT p.p_name, d.d_name FROM persons p JOIN departments d;
SELECT p.p_name, d.d_name FROM persons p INNER JOIN departments d;
SELECT p.p_name, d.d_name FROM persons p, departments d;
```

```
 P_NAME               D_NAME
 ------               ------
|John                 |Sales               |
|John                 |IT-technologies     |
|John                 |Finance             |
|John                 |Management          |
|John                 |Design              |
|Mary                 |Sales               |
|Mary                 |IT-technologies     |
|Mary                 |Finance             |
|Mary                 |Management          |
|Mary                 |Design              |
|Kate                 |Sales               |
|Kate                 |IT-technologies     |
|Kate                 |Finance             |
|Kate                 |Management          |
|Kate                 |Design              |
|Jack                 |Sales               |
|Jack                 |IT-technologies     |
|Jack                 |Finance             |
|Jack                 |Management          |
|Jack                 |Design              |
|Peter                |Sales               |
|Peter                |IT-technologies     |
|Peter                |Finance             |
|Peter                |Management          |
|Peter                |Design              |
|Ann                  |Sales               |
|Ann                  |IT-technologies     |
|Ann                  |Finance             |
|Ann                  |Management          |
|Ann                  |Design              |
```

18) При добавлении к конструкции CROSS JOIN <WHERE-спецификации> (условия выборки записей) результат будет эквивалентен соединению INNER JOIN (использующим для соединения то же условие).

```
SELECT * FROM A CROSS JOIN B WHERE  <логическое выражение> есть то
 же самое, что
SELECT * FROM A INNER JOIN B ON  <логическое выражение>
есть то же самое, что
SELECT * FROM A, B WHERE  <логическое выражение>
```

19) Конструкция <консолидированное соединение> (UNION JOIN) возвращает все несовпадающие строки обеих таблиц, дополненные NULL-значениями в столбцах результата, соответствующих столбцам другой таблицы.

20) <Консолидированное соединение> (UNION JOIN) является обратным по отношению к <внутреннему соединению> (INNER JOIN).

21) Если из результирующего набора <полного внешнего соединения> удалить строки, полученные в результате <внутреннего соединения>, то получится <консолидированное соединение>.

```
Примеры с использованием тестовых данных, описанных ранее:
1)
Получить список сотрудников, не работающих ни в одном отделе, и
 список отделов, в которых не работает ни один сотрудник.

SELECT p.p_name, d.d_name
  FROM persons p UNION JOIN departments d ON p.d_id = d.d_id;

 P_NAME        D_NAME
 ------        ------
|             |Sales           |
|Peter        |                |

2)
SELECT * FROM A UNION JOIN B ON <логическое выражение>

есть то же самое, что

SELECT A.*, <пустые значения> FROM A
 WHERE NOT EXISTS (SELECT * FROM B WHERE <логическое выражение>)
 UNION
SELECT <пустые значения>, B.*  FROM B
 WHERE NOT EXISTS (SELECT * FROM A WHERE <логическое выражение>)
```

22) На практике наиболее часто используются частные случаи описанных выше типов соединений:

- эквисоединение;
- рефлексивное соединение.

23) Эквисоединение создает результирующий набор, содержащий пары строк двух таблиц, для которых выполняется условие соединения, представляющее собой равенство значений пар соответствующих столбцов.

```
Пример соединения, которое является одновременно внутренним
 соединением и эквисоединением с использованием тестовых данных,
 описанных ранее:

Получить список сотрудников, работающих в имеющихся отделах.

SELECT p.p_name, d.d_name
  FROM persons p INNER JOIN departments d ON p.d_id = d.d_id;

 P_NAME         D_NAME
 ------         ------
|John           |Finance         |
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
|Mary          |IT-technologies  |
|Kate          |Management       |
|Jack          |IT-technologies  |
|Ann           |Design           |
```

24) Рефлексивное соединение используется для соединения таблицы самой с собой. В этом случае для различения разных экземпляров одной и той же таблицы применяются псевдонимы.

```
Пример соединения, которое является одновременно внутренним
 соединением, эквисоединением и рефлексивным соединением с
 использованием тестовых данных, описанных ранее:

Найти пары сотрудников, работающих в одном и том же отделе.

SELECT a.*, b.*
  FROM persons a INNER JOIN persons b ON a.d_id = b.d_id
 WHERE a.p_id > b.p_id;

 P_ID        P_NAME         D_ID       P_ID      P_NAME
 D_ID
 ----        ------         ----       ----      ------
 ----
|          2|Mary          |         2|        4|Jack          |
       2|
```

25) Для соединения нескольких таблиц необходимо применить операцию соединения последовательно несколько раз.

```
Дополнение определенных выше тестовых данных.
Добавлена таблица floors с информацией об этажах (номер и
 название).

create or replace table floors (num_f int, f_name char(20));
insert into floors values (1, 'First');
insert into floors values (2, 'Second');
insert into floors values (3, 'Third');
insert into floors values (4, 'Fourth');
insert into floors values (5, 'Fifth');
insert into floors values (6, 'Sixth');

Добавлена таблицу departments со столбцом num_f, содержащим номер
 этажа, на котором находится каждый из отделов.

create or replace table
  departments (d_id int, d_name char(20), num_f int);
insert into departments values (1, 'Sales',           1);
insert into departments values (2, 'IT-technologies', 3);
insert into departments values (3, 'Finance',         4);
insert into departments values (4, 'Management',      4);
```

```
insert into departments values (5, 'Design',          3);
```

```
Пример с использованием тестовых данных, описанных ранее:
Получить полный список сотрудников, отделов, в которых они
 работают, а также номера этажей этих отделов, упорядочить
 полученный список по номеру сотрудника (p_id) по возрастанию.
```

```
SELECT p.p_name, d.d_name, floors.num_f
  FROM persons p LEFT JOIN departments d ON p.d_id = d.d_id
                 LEFT JOIN floors ON d.num_f = floors.num_f
 ORDER BY p.p_id;
```

```
 P_NAME          D_NAME               NUM_F
 ------          ------               -----
|John            |Finance             |          4|
|Mary            |IT-technologies     |          3|
|Kate            |Management          |          4|
|Jack            |IT-technologies     |          3|
|Peter           |                    |           |
|Ann             |Design              |          3|
```

26) В конструкции JOIN можно использовать вложенные подзапросы, т.е. в качестве табличной ссылки может использоваться SELECT-запрос.

```
Получить полный список отделов и полный список работающих в них
 сотрудников, номер (p_id) которых больше 2, с соответствием
 сотрудников и указанных отделов.
```

```
SELECT tab_p.p_id, tab_p.p_name, d.d_id, d.d_name
  FROM departments d FULL OUTER JOIN
(SELECT * FROM persons WHERE p_id > 2) AS tab_p
ON d.d_id = tab_p.d_id;
```

```
 P_ID        P_NAME      D_ID        D_NAME
 ----        ------      ----        ------
|          3|Kate       |          4|Management          |
|          4|Jack       |          2|IT-technologies     |
|          5|Peter      |           |                    |
|          6|Ann        |          5|Design              |
|           |           |          1|Sales               |
|           |           |          3|Finance             |
```

27) Разрешено выполнять соединение таблиц по их псевдонимам и псевдонимам их столбцов.

```
Пример с использованием тестовых данных, описанных ранее:
Получить список сотрудников, работающих в имеющихся отделах.
```

```
SELECT d.b, p.d
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
  FROM departments d (a, b) NATURAL JOIN persons p (c, d, a);

 B                    D
 -                    -
|Finance              |John                |
|IT-technologies      |Mary                |
|Management           |Kate                |
|IT-technologies      |Jack                |
|Design               |Ann                 |
```

28) При использовании <USING-спецификации> в соединяемых таблицах при отсутствии столбцов с одинаковыми именами, возможен случай использования псевдонимов для таких столбцов.

```
CREATE OR REPLACE TABLE J1TBL (i integer, j integer, t char(32));
CREATE OR REPLACE TABLE J2TBL (k integer,  m integer);

insert into J1TBL values (1, 1, 'aa');
insert into J1TBL values (2, 2, 'bb');
insert into J2TBL values (1, 1);
insert into J2TBL values (3, 2);

SELECT * FROM J1TBL t1 (a, b, c)
  JOIN J2TBL t2 (a, b) USING (a, b);

 A     B      C
 -     -      -
|    1|     1|aa     |
```

Запрос выборки

Функция

Определение запроса выборки данных.

Спецификация

[1] <запрос выборки>::=
{
<запрос с общим табличным выражением>
|<табличный запрос>
|<подзапрос>
}
[<спецификация выборки>]

[2] <запрос с общим табличным выражением>::=
WITH <общее табличное выражение> [,…]
<select-запрос выборки>
|<комбинированный запрос>
|<иерархический запрос>

[3] <табличный запрос>::=
<select-запрос выборки>
|<комбинированный запрос>
|<иерархический запрос>

|<table-запрос выборки>
|<values-запрос выборки>
|<скалярная процедура>

[4] <спецификация выборки>::=
[<тип выборки>]
[<ограничение выборки>]
[<WITHIN GROUP-спецификация>]
[<WHERE-спецификация>]
[<ORDER BY-спецификация>]
[<GROUP BY-спецификация>]
[<HAVING-спецификация>]
[WAIT | NOWAIT]
[{QUANT| QUANTUM} TIMEOUT <время>]
[WITH PRIORITY <приоритет>]

[5] <общее табличное выражение>::=
<имя табличного выражения>[(<имя столбца>[,…])]AS(<подзапрос>)

[6] <имя табличного выражения>::= <идентификатор>

[7] <время>::= целочисленное положительное значение

[8] <приоритет>::= целочисленное положительное значение

Синтаксические правила

1) Если конструкция WAIT (NOWAIT) не задана, по умолчанию используется WAIT.
2) <Имена столбцов> и/или <FROM-спецификация> <select-запроса выборки> или его подзапросов могут содержать ссылки к <общим табличным выражениям>.

```
with "Цвета авто" ("Цвет") as (select color from auto)
  select "Цвет", count(*) as "Количество"
  from "Цвета авто" group by "Цвет";

Цвет        Количество
 ----        ----------
|BLACK      |        262|
|BLUE       |        108|
…
```

3) <Имена столбцов> и/или <FROM-спецификация> <общего табличного выражения> или его подзапросов могут содержать ссылки к определенным ранее <общим табличным выражениям>.

```
WITH t1 AS
  (select color, count(*) as cnt1 from auto group by color),
     t2 AS (select cnt1 as cnt2 from t1 where color='BLUE')
  SELECT color
  FROM t1, t2 WHERE cnt1 <> cnt2 and color not like'B%';

COLOR
 -----
|GREEN      |
|GREY       |
|RED        |
|WHITE      |
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
|YELLOW    |
```

4) Ссылки к <общим табличным выражениям>, определяемым позже, и рекурсивные ссылки не допускаются.

5) <Имя табличного выражения> должно отличаться от любого другого <имени табличного выражения>, определенного в том же предложении WITH, но оно может совпадать с именем базовой таблицы, именем представления или синонимом.

6) Повторение <имен столбцов> в <общем табличном выражении> не допускается.

7) Количество заданных <имен столбцов> должно совпадать с количеством столбцов в результирующей выборке соответствующего <подзапроса>. Список <имен столбцов> необязателен только в том случае, если все результирующие столбцы в соответствующем <подзапросе> имеют уникальные имена.

8) <Общее табличное выражение> можно использовать в команде CREATE VIEW, при этом одно <общее табличное выражение> не должно включать в себя другое <общее табличное выражение>.

```
create or replace view t(Color_Auto, Count_Auto)
  as with "Цвета авто" ("Цвет")
  as (select color from auto)
  select "Цвет", count(*) as "Количество"
  from "Цвета авто" group by "Цвет";
```

9) Несколько <общих табличных выражений> могут объединяться в <комбинированный запрос>. При этом <общие табличные выражения> могут ссылаться на самих на себя, а также на <общие табличные выражения>, определенные до этого в том же предложении WITH. Ссылки на определяемые далее <общие табличные выражения> недопустимы. Допустимые схемы <комбинированных запросов> с использованием <общих табличных выражений>:

```
а)
  with S1, S2, … SN
select from S1 …
 union | except | intersect
select from S2 …
 union | except | intersect
…
select from SN …
 union | except | intersect
б)
  with S1
select from S1 …
 union | except | intersect
  with S2
select from S2 …
 union | except | intersect
…
  with   SN
select from SN …
```

```
Например,
WITH "Список моделей"("Модель")
       as (select distinct model from auto limit 5),
     "Список производителей"("Производитель")
       as (select distinct make from auto limit 4),
     "Окраска авто"("Цвет")
       as (select distinct color from auto limit 3)
SELECT "Модель", count(*)
  from "Список моделей" where "Модель" like '2%'
  GROUP BY "Модель"
  UNION
SELECT "Производитель", count(*)
  from "Список производителей" group by "Производитель";
  UNION
SELECT "Цвет", count(*) from "Окраска авто";
```

10) <Общие табличные выражения> могут содержать вложенные <общие табличные выражения>:

```
with s(model) as
(with k as
  (select model from auto where color='RED' group by model)
  select count(*) from k)
select count(*) from s group by model;

|          1|
```

11) Конструкция WITH допускает использование параметров:

```
…
with q as
(select * from s where status = ?)
```

12) Параметрами <скалярной процедуры> может быть SELECT-запрос, возвращающий одно значение.

Общие правила

1) Максимальное число столбцов в <запросе выборки> равно 250.
2) Каждый из <подзапросов> может возвращать до 250 столбцов. При подсчете числа обработанных столбцов <запроса выборки> столбцы основного запроса и подзапросов не суммируются.
3) <Общее табличное выражение> можно представить себе как временный результирующий набор, определенный на время выполнения <запроса выборки>. Временные результирующие наборы не сохраняются в БД в виде объектов, время их жизни ограничено продолжительностью обработки запроса.
4) Конструкции с <общим табличным выражением> предназначена для:
   - замены представлений в тех случаях, когда создание и использование представления не оправдано, то есть тогда, когда нет необходимости сохранять в БД текст запроса представления;

Эти конструкции эквивалентны:

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
Подсчитать количество владельцев автомобилей, фамилия которых
 начинается на
 букву P, владеющих моделью, название которой начинается с GRAN и
 выпущенных
 компанией FORD.

WITH t as (select model
  from auto, person
  where auto.personid=person.personid and make='FORD'
        and firstnam like 'P%')
SELECT count(*) FROM t WHERE model LIKE 'GRAN%';

select count (*)
  from auto, person
  where auto.personid=person.personid
   and make='FORD'
   and firstnam like 'P%'
   and model like 'GRAN%';

create or replace view Ford_P as
(select model
  from auto, person
 where auto.personid=person.personid
   and make='FORD'
   and firstnam like 'P%');
select count(*) from Ford_P where model like 'GRAN%';
```

- многократных ссылок на одну и ту же выборку данных в одном и том же запросе.

5) При обработке <общего табличного выражения> между подсказками, ссылающимися на <общее табличное выражение>, может быть конфликт с другими подсказками, обнаруживаемыми, когда <общие табличные выражения> обращаются к их базовым таблицам так же, как подсказки обращаются к представлениям в запросах.
6) Опция WAIT заставляет СУБД ЛИНТЕР ожидать разблокирования данных, необходимых для выполнения <запроса выборки>.
7) Опция NOWAIT заставляет СУБД ЛИНТЕР не обрабатывать <запрос выборки> и выдавать соответствующий код завершения, если необходимые для обработки запроса данные заблокированы. В этом случае после некоторой паузы можно попытаться повторить выполнение запроса. Количество таких попыток определяется пользовательским приложением.
8) Параметр <время> задает максимально допустимую продолжительность кванта обработки запроса (от 1 до 65535 сек.). Если по каким-либо причинам квант обработки запроса превысил указанное значение, обработка запроса прекращается с выдачей соответствующего кода завершения.
9) Конструкция WITH PRIORITY <приоритет> устанавливает заданный приоритет выполняемому запросу (в диапазоне от 0 до 255). Если задать приоритет больше текущего приоритета пользователя, от имени которого подается запрос, то выдается код завершения 1022 «Нарушение привилегий».
10) Конструкция WITH PRIORITY назначает приоритет только тому запросу, в котором она указана. На приоритет любых последующих запросов она не влияет.

11) Независимо от значения RECORD SIZE LIMIT, максимальная строка записи выборки данных ограничена 65535 байтами.

Примеры

1)

```
select count(*) from auto, person
  where auto.personid=person.personid and make='FORD';
```

2)

```
! Пусть есть две таблицы:
! T1 – данные о поступивших в магазин продуктах;
! product_id – идентификатор продукта;
! part_id – номер (идентификатор) поставленной партии продукта;
! income – дата поступления партии продукта.

! T2 – справочник продуктов;
! product_id – идентификатор продукта;
! name – название продукта;
! max_term – максимальный срок реализации продукта.

! Создаем и заполняем таблицы
create or replace table T1
(
  product_id int, -- идентификатор продукта
  part_id int,    -- номер (идентификатор) поставленной партии
 продукта
  income date     -- дата поступления партии продукта (т.е.
 записей с одним
                  -- и тем же part_id может быть несколько)
);
create or replace table T2
(
  product_id int,   -- идентификатор продукта
  name varchar(50), -- название продукта
  max_term int      -- максимальный срок реализации продукта в
 днях
);

insert into t1 values (1,1,'05.09.2013');
insert into t1 values (1,2,'10.09.2013');
insert into t1 values (1,3,'12.09.2013');
insert into t1 values (2,1,'15.08.2013');
insert into t1 values (2,2,'01.09.2013');
insert into t1 values (3,24,'20.08.2013');
insert into t1 values (3,35,'02.09.2013');
insert into t1 values (4,7,'15.08.2013');
insert into t1 values (4,8,'01.09.2013');
```

```
insert into t2 values (1, 'Молоко', 3);
insert into t2 values (2, 'Колбаса', 14);
insert into t2 values (3, 'Сыр', 35);
insert into t2 values (4, 'Рыба', 10);

! а)
! Получить общее количество партий продуктов, в которых на текущую
 дату
! могут встретиться просроченные продукты

WITH obsolete as
(select t2.name, t1.part_id
  from t1,t2
  where t1.product_id = t2.product_id and
        sysdate > t1.income + t2.max_term)
select to_char(sysdate,'dd.mm.yyyy'), count(*)
  from obsolete;

|13.09.2013|          5|


! б)
! Получить количество разных наименований просроченных продуктов

WITH obsolete as
(select t2.name, t1.part_id
  from t1,t2
  where t1.product_id = t2.product_id and
        sysdate > t1.income + t2.max_term)
select to_char(sysdate,'dd.mm.yyyy'), count(distinct name)
  from obsolete;

|13.09.2013|          3|

! в)
! Получить названия продуктов с максимальным числом просроченных
 партий и
! количество просроченных партий.

WITH
obsolete as
(select t2.name, t1.part_id from t1,t2 where t1.product_id =
  t2.product_id and sysdate > income + max_term),
obs_count as
(select to_char(sysdate, 'dd.mm.yyyy'), name, count(*) cnt
```

```
  from obsolete group by name)
select * from obs_count where cnt = (select max(cnt)
  from obs_count);

            NAME             CNT
  ---       ----             ---
|13.09.2013|Молоко           |          2|
|13.09.2013|Рыба             |          2|
```

select-запрос выборки

Функция

Определение select-запроса выборки данных.

Спецификация

[1] <select-запрос выборки>::=
[(] SELECT [ALL | DISTINCT] <список выборки>
[<FROM-спецификация>] [)]
[2] <список выборки>::= * | <выбираемое значение> [,...]
[3] <выбираемое значение>::=
{<значимое выражение>
| <спецификация значения>
| <функция> (<OVER-спецификация>)
| <псевдоним столбца>}
[[AS] <псевдоним выбираемого значения>]
[4] <функция>::= <агрегатная функция>|<аналитическая функция>
[5] <псевдоним выбираемого значения>::= <идентификатор>

Синтаксические правила

1) Допустимые привилегии для имен таблиц (представлений), содержащихся в <табличном выражении>, должны включать SELECT.
2) По умолчанию опцию WAIT можно использовать только в select-запросе выборки (не в подзапросах).
3) Количество столбцов таблицы, получаемой по <select-запросу выборки>, равно числу элементов <списка выборки>.
4) Максимальное количество элементов в <списке выборки> равно 250.
5) Если в <списке выборки> задано *, это означает выбор всех столбцов из списка таблиц (представлений). Столбцы выбираются в соответствии с порядком, в котором они были определены при создании таблицы (представления) и порядке очередности таблиц (представлений), заданных во <FROM-спецификации>. Каждый столбец выбирается только один раз.
6) Имена столбцов в каждом <значимом выражении> должны однозначно указывать столбец в <табличном выражении>.
7) Ключевое слово DISTINCT не должно задаваться в <select-запросе выборки> более одного раза, за исключением случаев, когда оно задается в <подзапросах>, входящих в этот <основной запрос выборки>.

```
select distinct make, count (*)
  from auto group by make
having count(*) >(select count (distinct name) from person);
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
| AMERICAN MOTORS | 91  |
| CHRYSLER        | 168 |
| FORD            | 118 |
| GENERAL MOTORS  | 284 |
```

8) <Значимое выражение> может включать пользовательскую функцию СУБД ЛИНТЕР.

9) <Значимое выражение> может быть <логическим выражением>.

```
create or replace table t_bool (b boolean);
insert into t_bool values (TRUE);
insert into t_bool values (FALSE);
insert into t_bool values (NULL);
select b, not (b) "NOT" from t_bool;

 B    NOT
----  ----
| T  | F  |
| F  | T  |
|    |    |
```

```
select t1.b, t2.b, t1.b or t2.b "OR" from t_bool t1, t_bool t2;

 B   B   OR
------------
| T | T | T |
| T | F | T |
| T |   | T |
| F | T | T |
| F | F | F |
| F |   |   |
|   | T | T |
|   | F |   |
|   |   |   |
```

10) В качестве <пользовательской функции> может быть использована любая хранимая процедура, возвращающая значение одного из типов, разрешенных в языке SQL, не имеющая аргументов класса OUT или INOUT и не содержащая обращений к СУБД.

11) Если результатом <табличного выражения> является сгруппированная таблица, то каждая <спецификация столбца> в каждом <значимом выражении> должна быть группируемым столбцом или задаваться посредством агрегатной функции.

12) Если результатом <табличного выражения> не является сгруппированная таблица, а какое-нибудь <значимое выражение> включает в себя агрегатную функцию, то все столбцы каждого <значимого выражения> должны быть заданы посредством агрегатной функции.

13) Тип данных, длина, точность и дробная часть столбца таблицы-результата <запроса выборки> наследуются от <значимого выражения>, из которого получен столбец.

14) Если i-ое <выбираемое значение> в <списке выборки> специфицировано следующим за ним <именем столбца>, то i-ый столбец результата есть

именованный столбец. Если i-ое <выбираемое значение> в <списке выборки> не специфицировано следующим за ним <именем столбца>, но <значимое выражение> представлено <спецификацией столбца>, то i-ый столбец результата есть именованный столбец с именем <спецификация столбца>. В противном случае, i-ый столбец считается неименованным.

15) <Запрос выборки> является обновляемым только тогда, когда выполняются следующие условия:
   - ключевое слово DISTINCT не указано;
   - все <значимые выражения> из <списка выборки> включают в себя только <спецификацию столбца>, и ни одна <спецификация столбца> не встречается более одного раза;
   - <FROM-спецификация> из <табличного выражения> содержит точно одну <табличную ссылку>, и эта таблица (представление) является обновляемой;
   - <WHERE-спецификация> из <табличного выражения> не включает в себя <подзапрос>;
   - <табличное выражение> не содержит ни <GROUP BY-спецификации>, ни <HAVING-спецификации>.

16) <Табличное выражение> можно не задавать. В этом случае запрос SELECT может содержать только константные выражения и пользовательские функции от них, обращения к значениям последовательности (NEXTVAL, CURRVAL), а также константные предикаты. Такой запрос всегда возвращает одну запись.

```
SELECT SYSDATE;
SELECT LOG(2,5);
SELECT MY_SEQ.NEXTVAL;
SELECT 'СУБД ЛИНТЕР','версия 6.0';
SELECT ?(char(10)); //запрос с параметром

create or replace table xxx (i int);
insert into xxx values (10);
insert into xxx values (20);

create or replace procedure ppp () result int
declare var i int; //
code
  execute "select max(i) from xxx;" into i; //
  return i;  //
end;

select ppp() in (20,30);

|T|
```

17) Запрещено использование агрегатных функций в <запросе выборки> без <табличного выражения>.

18) Если в качестве выбираемого значения указано имя несуществующего столбца (<псевдоним столбца>), то оно должно быть перечислено в списке имен столбцов-псевдонимов <FROM-спецификации>.

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
select nmrerr, msg from errors fetch first 2;

|NMRERR |MSG                          |
|0      |операция завершена успешно   |
|1      |строка в таблице отсутствует |
```

```
select "Код завершения", "Текст сообщения" from errors as
 "Ошибки"  ("Код завершения", "Текст сообщения") fetch first 2;

|Код завершения|Текст сообщения              |
|0             |операция завершена успешно   |
|1             |строка в таблице отсутствует |
```

```
select nmrerr as "Код завершения", msg as "Текст сообщения" from
 errors fetch first 2;

|Код завершения|Текст сообщения              |
|0             |операция завершена успешно   |
|1             |строка в таблице отсутствует |
```

19) В качестве <функции> в данной версии допустимы только функции LEAD и LAG.

Общие правила

1) Если результат <табличного выражения> не является сгруппированным, а <список выборки> включает только агрегатные функции, то результат <табличного выражения> представляет собой значение каждой такой функции, а результатом <запроса выборки> будет таблица, состоящая из одной строки. i-ым значением в строке будет значение, специфицированное i-ым <значимым выражением>.

```
select count(*), min(year+1900), max(cylnders), default(make) from
 auto;

|1000 |1970 |12 |null |
```

2) Если результат <табличного выражения> не является сгруппированным, <список выборки> не включает агрегатные функции, то каждое <значимое выражение> применяется к каждой строке результата <табличного выражения>, и в результате получается таблица из «m» строк, где «m» – мощность <табличного выражения>. Если DISTINCT не указан, то результатом <запроса выборки> будет такая таблица. Если DISTINCT указан, то результатом <запроса выборки> будет таблица, полученная из описанной путем исключения дубликатов строк.

3) Если результат <табличного выражения> является сгруппированным и имеет нуль групп, то результатом выполнения <запроса выборки> будет пустая таблица.

   Сравните:

   а) в ответе 31 группа:

```
select make from auto
  where color = 'BLACK' group by make;
```

```
|ALPINE          |
|AMERICAN MOTORS |
|BMW             |
 …
```

б) ответ пуст:

```
select make from auto where color='GOLD' group by make;
```

4) Если результат <табличного выражения> является сгруппированным и имеет одну группу, а каждое <значимое выражение> в <списке выборки> представляет собой агрегатную функцию, то результатом <запроса выборки> будет таблица, имеющая одну строку. I-е значение этой строки – результат i-й функции:

```
select count(*), min(year+1900), max(cylnders), default(make)
  from auto group by 2;

|1000  |1970  |12     |       |
```

5) Если результат <табличного выражения> является сгруппированным и имеет одну или более групп, то результатом применения каждого <значимого выражения> к каждой группе результата является таблица из «m» строк, где «m» – число групп в <табличном выражении>. Когда <значимое выражение> применяется к данной группе, эта группа является аргументом каждой агрегатной функции в <значимом выражении>. Если DISTINCT не указан, то результатом будет описанная таблица. Если указан DISTINCT, результатом <запроса выборки> будет таблица, полученная из описанной путем исключения дубликатов строк.

```
select make, count( make) from auto
  where color='BLACK' group by make;

|ALPINE          | 1  |
|AMERICAN MOTORS | 23 |
|BMW             | 3  |
 …
select make,  count( distinct make) from auto
  where color='BLACK' group by make;

|ALPINE          | 1 |
|AMERICAN MOTORS | 1 |
|BMW             | 1 |
 …
```

6) Строка является дубликатом другой строки только тогда, когда все пары значений в одних и тех же позициях строк равны.

7) Суммарная длина значений всех столбцов ответа должна быть не более 64 Кбайт.

8) Разрешено использование в одном и том же подзапросе двух функций с модификатором DISTINCT, если они ссылаются к разным <значимым выражениям>.

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

9) Если в одном и том же запросе встречаются и UNION DISTINCT, и UNION ALL, то все UNION ALL заменяются на UNION DISTINCT.

Пример

```
select 1 from $$$sysrl group by $$$s11 having
  exists
    (select 1 from $$$attri group by $$$s21
     having avg(distinct $$$s12) = sum(distinct $$$s22)
    );
```

table-запрос выборки

Функция

Определение table-запроса выборки данных.

Спецификация

[1] <table-запрос выборки>::=
TABLE [(][<имя схемы>.] {<имя таблицы> | <имя представления> [)]}
| (<select-запрос выборки>)
| (<table-запрос выборки>)

Синтаксические правила

1) Конструкция «TABLE» является сокращенной записью (синонимом) конструкции «SELECT * FROM».

Эти конструкции эквивалентны:

```
table auto;

table (auto);

select * from auto;
```

2) Допустимые привилегии для таблиц (представлений) должны включать SELECT.

3) <Спецификация выборки> для <table-запроса выборки> аналогична <спецификации выборки> для <запроса выборки>.

```
table auto where color='RED';
table auto where color='RED' order by make;
table auto where color='RED' order by make limit 5;
```

4) Разрешается комбинирование <table-запросов выборки> и <select-запросов выборки>.

```
table (select make, count(make) from auto
where color='RED' group by make ) as A where a.make like 'F%'
 limit 10;

|FERRARI             |          1|
|FIAT                |          1|
|FORD                |          5|

select * from S where (SNUM,SNAME,STATUS,CITY) in (table S);
```

Пример

```
create or replace procedure test_proc() result cursor(snum
 char(5), sname char(20), status int, city char(15))
declare
  var b typeof(result); //
code
  open b for direct "table SYSTEM.S;"; //
  return b; //
end;
```

values-запрос выборки

Функция

Определение values-запроса выборки данных.

Спецификация

[1] <values-запрос выборки>::=
VALUES (<список значений1>) [, (<список значений2>) ...]

Синтаксические правила

1) Конструкция

```
values    (<список значений1>) [, (<список значений2>) …]
```

эквивалента конструкции

```
 select <список значений1> [union all select <список
 значений2>] ...
```

```
values
(1, 'Иванов', to_date('22.01.1985', 'dd.mm.yyyy')),
(2, 'Петров', to_date('15.07.1990', 'dd.mm.yyyy'));

|          1|Иванов|22.01.1985:00:00:00.00|
|          2|Петров|15.07.1990:00:00:00.00|
```

```
select 1, 'Иванов', to_date('22.01.1985', 'dd.mm.yyyy')
UNION ALL
select 2, 'Петров', to_date('15.07.1990', 'dd.mm.yyyy');

|          1|Иванов|22.01.1985:00:00:00.00|
|          2|Петров|15.07.1990:00:00:00.00|
```

2) Конструкция <values-запрос выборки> допустима в тех местах, где по синтаксису разрешена конструкция <select-запрос выборки>.

```
table (values (1, 'Иванов', to_date('22.01.1985', 'dd.mm.yyyy'),
2, 'Петров', to_date('15.07.1990', 'dd.mm.yyyy')));

|          1|Иванов|22.01.1985:00:00:00.00|
|          2|Петров|15.07.1990:00:00:00.00|
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
values
(1, 'Иванов', to_date('22.01.1985', 'dd.mm.yyyy')),
(2, 'Петров', to_date('15.07.1990', 'dd.mm.yyyy'))
limit 1;
```

```
values
(1 as id, 'Иванов', to_date('22.01.1985', 'dd.mm.yyyy')),
(2 as id, 'Петров', to_date('15.07.1990', 'dd.mm.yyyy'))
order by 2 desc;

|          2|Петров|15.07.1990:00:00:00.00|
|          1|Иванов|22.01.1985:00:00:00.00|
```

```
select count(*) from (
values
(1, 'Иванов', to_date('22.01.1985', 'dd.mm.yyyy')),
(2, 'Петров', to_date('15.07.1990', 'dd.mm.yyyy')));

|          2|
```

3) Максимальное число элементов в списке VALUES - 2000 (все значения полей всех добавляемых строк в INSERT образуют список и общее число этих значений полей строк не должно быть больше 2000, то есть для кортежей из 1 поля можно добавить за раз 2000 строк, для кортежей из 4-х полей - 500 строк и т.д.).

Иерархический запрос

Функция

Определение иерархического запроса к таблице.

Спецификация

[1] <иерархический запрос>::=
[START WITH <начальный уровень>]
[CONNECT BY <условие связи>]
[ORDER SIBLINGS BY <имя столбца>]
[2] <начальный уровень>::= <предикат> [, …]
[3] <условие связи>::=
<предыдущий уровень> = <последующий уровень>
[<предикат> <условие связи>, ...]
[4] <предыдущий уровень>::= PRIOR <значимое выражение>
[5] <последующий уровень>::= <значимое выражение>

Синтаксические правила

1) Опция `START WITH` задает условия выбора начальной записи иерархической структуры данных.
2) Опция `CONNECT BY <условие связи>` задает отношение между узлами иерархической структуры данных. Часть элементов `<условия связи>` должна быть использована в опции `PRIOR` для ссылки на предыдущий уровень иерархии.
3) Опция `PRIOR` устанавливает предыдущий уровень текущей записи иерархии. `PRIOR` – унарный оператор и обрабатывается как обычные унарные + и - операторы.

В неиерархических запросах при одной исходной таблице во всех <условиях связи> имена столбцов (не считая внешних ссылок) рассматриваются как ссылки к одной и той же записи исходной таблицы. Напротив, в иерархическом запросе <условие связи> должно содержать ссылки к двум записям – текущего и предыдущего уровня, которые оно и связывает между собой.

4) В <условиях связи> <предыдущий уровень> и <последующий уровень> можно менять местами. <Условие связи> должно содержать ссылки к двум записям – текущего и предыдущего уровня, которые связывает между собой. Но так как в иерархическом запросе может участвовать только одна таблица, то различие между обращениями к записям текущего и предыдущего уровня делается следующим образом: если перед аргументом предиката стоит ключевое слово PRIOR, то все обращения к столбцам в этом аргументе предиката относятся к записи предыдущего уровня, если же PRIOR не стоит – то к записи текущего уровня.

Эти конструкции эквивалентны:

```
 SELECT *
   FROM "Служащие"
  START WITH "Должность" = 'РУКОВОДИТЕЛЬ'
CONNECT BY PRIOR "Таб_Номер"="Таб_Номер_Руковод";
```

```
 SELECT *
   FROM "Служащие"
  START WITH "Должность" = 'РУКОВОДИТЕЛЬ'
CONNECT BY "Таб_Номер_Руковод" = PRIOR "Таб_Номер";
```

5) <Предыдущий уровень> и <последующий уровень> должны ссылаться на <значимые выражения>, содержащие разные столбцы таблицы.

6) Типы данных <предыдущего уровня> и <последующего уровня> должны быть совместимыми.

7) Условия WHERE, START и CONNECT в иерархических запросах можно задавать в произвольном порядке.

8) Столбец LEVEL внутри иерархического запроса используется для обозначения уровня иерархии, в остальных запросах – как идентификатор (имя столбца).

```
create table tst(s int, f int);
insert into tst values (1,2), (2,3);

SELECT level, COUNT(*) as COUNT
  FROM (SELECT level FROM TST START WITH s=1 CONNECT BY S=PRIOR f)
 a
 GROUP BY level;
```

или

```
  WITH t1 as (SELECT level FROM TST START WITH s=1 CONNECT BY
 S=PRIOR f)
SELECT level, COUNT(*) as COUNT
  FROM t1 GROUP BY level;

 LEVEL       COUNT
 -----       -----
|          1|          1|
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
|          2|          1|
```

Примечание
Столбец LEVEL в опции CONNECT BY обрабатывается корректно только для простых условий типа LEVEL <значение> (для сложных условий результат может быть не верным).

9) Опция SIBLINGS в <ORDER BY-спецификации> задаёт порядок сортировки записей внутри уровней иерархического запроса. (Без сортировки вывод записей выполняется в порядке обхода иерархического дерева, т.е. сверху-вниз, слева-направо).

```
create or replace table test(name char(5), id int, parent int);
insert into test values ('F', 1, NULL);
insert into test values ('C', 2, 1);
insert into test values ('H', 3, 2);
insert into test values ('E', 4, 2);
insert into test values ('B', 5, 1);
insert into test values ('G', 6, 5);
insert into test values ('D', 7, 6);
insert into test values ('A', 8, 5);
```

а) обычная сортировка

```
 select name, id, parent, level
   from test start with parent is null
connect by prior id = parent order by name;
```

```
 NAME  ID          PARENT      LEVEL
 ----  --          ------      -----
|A    |          8|          5|          3|
|B    |          5|          1|          2|
|C    |          2|          1|          2|
|D    |          7|          6|          4|
|E    |          4|          2|          3|
|F    |          1|       NULL|          1|
|G    |          6|          5|          3|
|H    |          3|          2|          3|
```

б) иерархическая сортировка

```
 select name, id, parent, level
   from test start with parent is null
connect by prior id = parent order siblings by name;
```

```
 NAME  ID          PARENT      LEVEL
 ----  --          ------      -----
|F    |          1|       NULL|          1|
|B    |          5|          1|          2|
```

```
|A    |          8|          5|          3|
|G    |          6|          5|          3|
|D    |          7|          6|          4|
|C    |          2|          1|          2|
|E    |          4|          2|          3|
|H    |          3|          2|          3|
```

10) Запрещается использовать одновременно <WHERE-спецификацию> и <ORDER BY-спецификацию> с опцией SIBLINGS. Т.е. запрос вида

```
 select * from test start with parent is null
connect by prior id = parent where id <> 1 order siblings by name;
```

является синтаксически неправильным. Будет выдан код завершения 2279 «Использование конструкции ORDER SIBLINGS недопустимо в данном запросе».

11) Псевдостолбцы ROWID и ROWTIME в иерархических запросах запрещены. Например, нижеследующие запросы синтаксически некорректны:

```
 select * from TEST start with PARENT is null
connect by prior ID = PARENT ORDER by ROWID;

 select * from TEST start with PARENT is null
connect by prior ID = PARENT ORDER siblings by ROWID;
```

12) Опция GROUP BY в иерархических запросах разрешена, например, нижеследующий запрос:

```
 select COUNT(*) COUNT, LEVEL from test start with PARENT is null
connect by prior ID = PARENT GROUP BY LEVEL;

 COUNT LEVEL
 ----- -----
|    1|    1|
|    2|    2|
|    4|    3|
|    1|    4|
```

Общие правила

1) Результирующая таблица по умолчанию имеет тот же список столбцов, что и исходная, с добавлением псевдостолбца с именем «LEVEL» типа INTEGER, который указывает номер уровня записи в иерархии.
2) Наличие псевдостолбца с именем «LEVEL» в выборке вида SELECT * FROM ... зависит от ключа /COMPATIBILITY=ORACLE в команде запуска ядра СУБД:
   - псевдостолбец будет присутствовать, если ключ не задан;
   - псевдостолбец будет отсутствовать, если ключ задан.
3) Псевдостолбец LEVEL можно указывать в <основном запросе выборки> и <WHERE-спецификации> и нельзя в конструкциях START WITH и CONNECT BY.
4) Сортировка записей в ответе (если не задана <GROUP BY-спецификация>) выполняется в таком порядке, чтобы все записи, подчиненные в иерархии некоторой записи, следовали за ней.

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

5) Условие, содержащееся в конструкции START, определяет записи верхнего уровня в иерархии.
6) Условие, содержащееся в конструкции CONNECT, определяет связи между записями предыдущего и последующего уровней.
7) <Иерархический запрос> выполняется столько раз, сколько <начальных уровней> найдено в таблице, задаваемой <FROM-спецификацией>.
8) Максимальное количество уровней для сортируемых иерархических запросов равно 4096.
9) Если в иерархии записей обнаружено нарушение структуры дерева, фиксируется исключительная ситуация.
10) Для запросов с CONNECT BY можно подавать команды позиционного обновления и удаления (WHERE CURRENT OF).

Примеры

```
create or replace table tab1 (id int, "Фамилия" char(15), "Имя"
 char(10), "Год рождения" int, "Статус" char(10), "Поколение" int,
 "Отношение" int);
insert into tab1 values(10,'Петрова', 'Мария', 1989, 'дочь', 3,5);
insert into tab1 values(9,'Петров', 'Сергей', 1989, 'сын', 3, 5);
insert into tab1 values(8,'Иванов', 'Сергей', 1989, 'сын', 3,3 );
insert into tab1 values(7,'Иванов', 'Михаил', 1994, 'сын', 3,4 );
insert into tab1 values(6,'Иванов', 'Иван', 1972, 'сын', 2, 1);
insert into tab1 values(5,'Петрова', 'Анна', 1970, 'дочь', 2, 1);
insert into tab1 values(4,'Иванов', 'Андрей', 1974, 'сын', 2, 1);
insert into tab1 values(3,'Иванов', 'Сергей', 1968, 'сын', 2,1 );
insert into tab1 values(2,'Иванова', 'Мария', 1950, 'бабушка',
 1,0);
insert into tab1 values(1,'Иванов', 'Иван', 1948, 'дедушка', 1,0);
```

Сколько детей и внуков у Иванова Ивана:

```
 select count(*)-1 from tab1 start with id=1
connect by prior id="Отношение" and "Отношение"<>0;

|8 |
```

Сколько всего внучат:

```
 select count(*) from tab1 start with "Поколение"=3
connect by prior id="Отношение" and
  prior id="Поколение" and "Отношение"<>0;

|4 |
```

Сколько всего внучат, родившихся до 1990 года:

```
 select count(*) from tab1 start with "Поколение"=3
connect by prior id="Отношение" and
  prior id="Поколение" and "Отношение"<>0 where "Год
 рождения"<1990;
```

```
|3 |
```

Сколько внучат от 10 до 15 лет:

```
 select count(*) from tab1 start with "Поколение"=3 and
 datesplit(to_date('2000','yyyy'),'y')- "Год рождения" between 10
 and 15
connect by prior id="Отношение"
    and prior id="Поколение" and "Отношение"<>0;
```

```
|3 |
```

Сколько лиц мужского пола в семействе:

```
 select count(id) from tab1 start with id=1
connect by prior id="Отношение"
    and "Статус"='сын' and "Отношение"<>0;
```

```
|6 |
```

Сколько дядей и тетей у детей Анны Петровой (дочери Ивана Иванова):

```
 select count(id)-1 from tab1
  start with id=(select distinct id from tab1 where
 "Фамилия"='Иванов' and "Имя"='Иван' and "Поколение"=1)
connect by prior id="Отношение"
    and "Фамилия" not like 'Петров%'
    and "Поколение"=2
    and "Отношение"<>0;
```

```
|3 |
```

Имена и фамилии дядей и тетей детей Анны Петровой (дочери Ивана Иванова):

```
select cast "Фамилия" || ' ' || "Имя" as char(30) from tab1
  where id in (select id from tab1
  start with id = (select distinct id from tab1
                   where "Фамилия" = 'Иванов'
                     and "Имя" = 'Иван'
                     and "Поколение" = 1)
connect by prior id = "Отношение"
    and "Фамилия" not like 'Петров%'
    and "Поколение" = 2
    and "Отношение" <> 0
 except
 select distinct id from tab1
  where "Фамилия" = 'Иванов'
    and "Имя" = 'Иван'
    and "Поколение" = 1);
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
|Иванов Сергей|
|Иванов Андрей|
|Иванов Иван  |
```

```
create table "Служащие"
(
  "Фамилия" char(40),
  "Таб_Номер" int primary key,
  "Таб_Номер_Руковод" int references "Служащие"("Таб_Номер"),
  "Должность" char(20)
);
INSERT INTO "Служащие" VALUES ('ИВАНОВ',7839,NULL,'РУКОВОДИТЕЛЬ');
INSERT INTO "Служащие" VALUES ('ПЕТРОВ',7698,7839,'МЕНЕДЖЕР');
INSERT INTO "Служащие" VALUES ('СИДОРОВ',7782,7839,'МЕНЕДЖЕР');
INSERT INTO "Служащие" VALUES ('СМИРНОВ',7566,7839,'МЕНЕДЖЕР');
INSERT INTO "Служащие" VALUES ('СИМОНОВ',7902,7566,'АНАЛИТИК');
INSERT INTO "Служащие" VALUES ('СОЛОВЬЕВ',7369,7902,'СЛУЖАЩИЙ');
INSERT INTO "Служащие" VALUES ('АНТОНОВ',7499,7698,'ПРОДАВЕЦ');
INSERT INTO "Служащие" VALUES ('СЕМЕНОВ',7521,7698,'ПРОДАВЕЦ');
INSERT INTO "Служащие" VALUES ('КУЗНЕЦОВ',7654,7698,'ПРОДАВЕЦ');
INSERT INTO "Служащие" VALUES ('МИРОНОВ',7788,7566,'АНАЛИТИК');
INSERT INTO "Служащие" VALUES ('ТИХОНОВ',7844,7698,'ПРОДАВЕЦ');
INSERT INTO "Служащие" VALUES ('БОРИСОВ',7876,7788,'СЛУЖАЩИЙ');
INSERT INTO "Служащие" VALUES ('ВАСИЛЬЕВ',7900,7698,'СЛУЖАЩИЙ');
INSERT INTO "Служащие" VALUES ('ТИХОНОВ',7934,7782,'СЛУЖАЩИЙ');

SELECT *
  FROM "Служащие"
  START WITH "Должность" = 'РУКОВОДИТЕЛЬ'
CONNECT BY PRIOR "Таб_Номер" = "Таб_Номер_Руковод";
```

```
 Фамилия   Таб_Номер Таб_Номер_Рук Должность     LEVEL
 ----      ----      ----          ----          ----
|ИВАНОВ    | 7839    | <NULL>     |РУКОВОДИТЕЛЬ | 1   |
|ПЕТРОВ    | 7698    | 7839        |МЕНЕДЖЕР     | 2   |
|АНТОНОВ   | 7499    | 7698        |ПРОДАВЕЦ     | 3   |
|СЕМЕНОВ   | 7521    | 7698        |ПРОДАВЕЦ     | 3   |
|КУЗНЕЦОВ  | 7654    | 7698        |ПРОДАВЕЦ     | 3   |
|ТИХОНОВ   | 7844    | 7698        |ПРОДАВЕЦ     | 3   |
|ВАСИЛЬЕВ  | 7900    | 7698        |СЛУЖАЩИЙ     | 3   |
|СИДОРОВ   | 7782    | 7839        |МЕНЕДЖЕР     | 2   |
|ТИХОНОВ   | 7934    | 7782        |СЛУЖАЩИЙ     | 3   |
|СМИРНОВ   | 7566    | 7839        |МЕНЕДЖЕР     | 2   |
|СИМОНОВ   | 7902    | 7566        |АНАЛИТИК     | 3   |
|СОЛОВЬЕВ  | 7369    | 7902        |СЛУЖАЩИЙ     | 4   |
```

```
|МИРОНОВ   | 7788     | 7566          |АНАЛИТИК      | 3    |
|БОРИСОВ   | 7876     | 7788          |СЛУЖАЩИЙ      | 4    |
```

Если необходимо явно указать уровень иерархии в нужной позиции выборки, в запросе надо прописать имя LEVEL большими буквами в кавычках.

```
select "LEVEL", "Фамилия" from "Служащие"
  start with "Должность" = 'АНАЛИТИК'
connect BY PRIOR "Таб_Номер" = "Таб_Номер_Руководитель";

 LEVEL Фамилия
 ----  ----
|1    |СИМОНОВ   |
|2    |СОЛОВЬЕВ  |
|1    |МИРОНОВ   |
|2    |БОРИСОВ   |
```

Комбинированный запрос

Функция

Определение комбинированного запроса.

Спецификация

[1] <комбинированный запрос>::=
<объединение запросов>
| <разность запросов>
| <пересечение запросов>

Синтаксические правила

1) Суммарная длина текста <комбинированного запроса> не должна превышать 32 Кбайта.
2) Ограничение на максимальное количество связываемых объектов в <запросе выборки> распространяется только на составной <запрос выборки> <комбинированного запроса>, а не на весь <комбинированный запрос>.
3) Порядок выполнения составных <запросов выборки> <комбинированного запроса> регулируется группирующими скобками.

```
create table tab1 (I int);
create table tab2 (I int);
create table tab3 (I int);
create table tab4 (I int);
create table tab5 (I int);
```

Содержимое таблиц tab1–tab5:

tab1	tab2	tab3	tab4	tab5
1	6	8	11	8
2	7	9	12	9
3	1	10	13	10
4	2	1	14	11

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

tab1	tab2	tab3	tab4	tab5
5	3	2	15	12

Последовательное применение операций комбинированного запроса к его составным запросам выборки.

Схема комбинированного запроса:

```
S1 union S2 except S3 union S4 intersect S5:
select * from tab1
 union
select * from tab2
except
select * from tab3
 union
select * from tab4
intersect
select * from tab5;

|3 |
|4 |
|5 |
|6 |
|7 |
|11|
|12|
```

Применение операций комбинированного запроса к его составным запросам выборки с учетом группирующих скобок.

Схема комбинированного запроса:

```
S1 union (S2 except S3) union (S4 intersect S5)
select * from tab1
 union
(select * from tab2
 except
 select * from tab3)
 union
(select * from tab4
 intersect
 select * from tab5);

|1 |
|2 |
|3 |
|4 |
|5 |
```

```
|6 |
|7 |
|11|
|12|
```

Схема комбинированного запроса:

```
S1 union S2 except (( S3 union S4)  intersect S5)
select * from tab1
 union
select * from tab2
except
((select * from tab3
   union
  select * from tab4)
intersect
select * from tab5);

|1|
|2|
|3|
|4|
|5|
|6|
|7|
```

Схема комбинированного запроса:

```
(((S1 union S2) except S3) union S4) intersect S5
(((select * from tab1
 union
select * from tab2)
except
select * from tab3)
 union
select * from tab4)
intersect
select * from tab5;

|11|
|12|
```

4) Именование столбцов комбинированного запроса выполняется по следующим правилам:
   - если соответствующие столбцы операндов комбинированного запроса имеют одинаковое имя, результирующий столбец будет иметь то же имя;
   - если соответствующие столбцы операндов комбинированного запроса имеют разные имена, результирующий столбец будет иметь имя столбца из первого операнда;

```
select 1 as aaa
 union
select 2 as bbb
 union
select 3 as ccc;

AAA
----
1
----
2
----
3
```

- если среди соответствующих столбцов операндов комбинированного запроса имеются неименованные столбцы, результирующий столбец будет иметь имя первого именованного столбца из списка этих столбцов;

```
select 1
 union
select 2
 union
select 3 as ccc;

CCC
----
1
----
2
----
3
```

- если все столбцы операндов комбинированного запроса являются неименованными, то результирующий столбец также будет неименованным.

Объединение запросов

Функция

Определение объединения результатов запросов выборки данных.

Спецификация

[1] <объединение запросов> : : =
[(]<составной запрос> UNION [ALL | DISTINCT]
[CORRESPONDING [BY (<список столбцов>)] <составной запрос>][)]
[2] <список столбцов> : : = <имя столбца> [, …]
[3] <составной запрос> : : =
[(]<запрос выборки>[)] | [(]<комбинированный запрос>[)]

Синтаксические правила

1) Если не используется конструкция CORRESPONDING, то все <составные запросы> одного <объединения запросов> должны иметь одинаковое количество совместимых по типу данных столбцов (различными могут быть только имена столбцов).
2) Допускается объединение столбцов с фиксированной и переменной длиной без явного указания приведения типа:
   - для столбцов типа CHAR и VARCHAR результат VARCHAR;
   - для столбцов типа BYTE и VARBYTE результат VARBYTE;
   - для столбцов типа NCHAR и NCHAR VARYING результат NCHAR VARYING.

```
select 'Модели автомобиля' from auto
 union
select distinct model from auto;

select 'Модель', 'год выпуска' from auto
 union
select distinct model, to_char(year+1900) from auto;
```

3) Типы числовых литералов при необходимости преобразуются к другим числовым типам при выполнении объединения: SMALLINT => INTEGER => BIGINT => DECIMAL => REAL => DOUBLE, для столбцов и выражений с числовыми типами данных неявное приведение типов не производится.

   Запрос:

```
select 5.78
 union
select 10;

эквивалентен:
select 5.78
 union
select cast 10 as decimal;

| 5.78|
| 10.0|
```

4) Синтаксические скобки «(» и «)» являются необязательными и предназначены, в основном, для логического выделения элементов SQL-запроса.

```
from S1 union S2
from (S1) union (S2)
from (S1 union S2)
from ((S1) union (S2))
from S1 union (S2)
from (S1 union (S2))
from (S1) union S2
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
from ((S1) union S2)
create or replace table t1 (i int, c char(5));
insert into t1 (i,c) values(1,'aaaaa');
insert into t1 (i,c) values(2,'bbbbb');

create or replace table t2 (i int, c char(5));
insert into t1 (i,c) values(3,'ccccc');
insert into t1 (i,c) values(4,'ddddd');
```

Эти 3 конструкции эквивалентны:

```
select * from ((select * from t1) union (select * from t2));
select * from ((table t1) union (table t2));
select * from (values (1,'aaaaa'), (2, 'bbbbb')) union (values
 (3,'ccccc'), (4, 'ddddd'));

|          1|aaaaa|
|          2|bbbbb|
|          3|ccccc|
|          4|ddddd|

select * from (select * from t1 union select * from t2);
select * from select * from t1 union select * from t2;  не
 работает
select * from (select * from t1) union (select * from t2);

select * from (select * from t1) union select * from t2;
select * from (select * from t1 union (select * from t2));
select * from ((select * from t1) union select * from t2);
```

5) Если задана конструкция CORRESPONDING BY (объединение перечисленных столбцов), то <объединение запросов> будет содержать столбцы, указанные в <списке столбцов>.
6) <Список столбцов> должен содержать не повторяющиеся имена столбцов, совпадающие с именами столбцов из <составных запросов>.
7) Каждое имя столбца в списке соответствия должно быть явно задано во всех <составных запросах>.

```
select x1, x2, a, x3, b, c from T1 …
 union corresponding by (a, b, c)
select a, y1, b, c, y2, y3 from T2 …
```

8) Дублирование имен столбцов в списке CORRESPONDING запрещено. Недопустимая конструкция: CORRESPONDING BY (make, make).

Общие правила

1) Если UNION не указан, то <объединение запросов> является просто <запросом выборки>.
2) Если UNION указан, то результат <объединения запросов> получается следующим образом:
   - результат инициализируется пустой таблицей;

- в результат вставляются все строки первого <составного запроса> и затем все строки второго <составного запроса>.

3) Если ALL не указан, из результата исключаются дубликаты строк.

Пусть есть две таблицы:

```
 tab1   tab2
 1       2
 2       4
 3       5
 4       7
 3
```

```
select * from tab1      select * from tab1
 union all               union
select * from tab2;     select * from tab2;
```

```
| 1|                    |          1|
| 2|                    |          2|
| 3|                    |          3|
| 4|                    |          4|
| 3|                    |          5|
| 2|                    |          7|
| 4|
| 5|
| 7|
```

4) В <объединении запросов> <ORDER BY-спецификация> может использоваться только один раз. Она располагается в последнем операторе SELECT <объединения запросов> и применяется ко всему результату. Столбцы упорядочения в этом случае задаются путем указания их порядковых номеров или имен столбцов, если имена столбцов идентичны во всех <составных запросах>.

```
select 'Модель', 'год выпуска' from auto
 union
select distinct model, to_char(year+1900) from auto order by 2;
```

```
select personid from auto
 union all
select personid from person order by personid;
```

5) В <объединении запросов> <GROUP BY-спецификация> может применяться как к составным запросам, так и ко всей комбинированной выборке.

```
create or replace table test1(i int);
insert into test1 values(2);
insert into test1 values(1);
insert into test1 values(2);

create or replace table test2(j int);
insert into test2 values(3);
insert into test2 values(2);
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
insert into test2 values(3);

select v1, count(v1) from
(select i as v1 from test1
  union all
 select j as v1 from test2)
group by v1;

 V1
 --
|          1|          1|
|          2|          3|
|          3|          2|
(select i,count(i) from test1 group by i)
 union all
(select j,count(j) from test2 group by j);

|          1|          1|
|          2|          2|
|          2|          1|
|          3|          2|
```

6) I-й столбец результата <объединения запросов> будет именован только в том случае, если среди исходных i-х столбцов есть хотя-бы один именованный столбец. В качестве имени результирующего столбца будет взято имя первого именованного столбца из операндов комбинированного запроса.

```
select 1 as aaa, 1,         1
 union
select 2 as bbb, 2,         2
 union
select 3 as ccc, 3 as ccc,  3;

 AAA          CCC
 ---          ---          ---
|          1|          1|          1|
|          2|          2|          2|
|          3|          3|          3|
```

7) Разрешено объединение по UNION столбцов следующих типов:
   - CHAR и VARCHAR (результат VARCHAR);
   - BYTE и VARBYTE (результат VARBYTE);
   - NCHAR и NVARCHAR (результат NVARCHAR).
8) Разрешено обновление/удаление/вставка над UNION ALL нескольких запросов, если каждый из них является обновляемым, при этом одним запросом реально могут изменяться несколько таблиц. При переходе по выборке к следующей записи выборки данных может происходить смена таблицы для текущей записи в курсоре. Контроль доступа применяется к каждой отдельной таблице, из которой берет данные конструкция UNION ALL.

```
create or replace table t_1 (i int, b boolean, c char(10));
insert into t_1 values (1,TRUE,'Sample');
insert into t_1 values (2,FALSE,'One more');
insert into t_1 values (3,FALSE,'Extra');

create or replace table t_2 (i int, b boolean, c char(10));
insert into t_2 values (2,FALSE,'Union');
insert into t_2 values (3,FALSE,'Intersect');
insert into t_2 values (4,FALSE,'Except');

create or replace table t_3 (i int, b boolean, c char(10));
insert into t_3 values (3,FALSE,'Insert');
insert into t_3 values (4,TRUE,'Delete');
insert into t_3 values (5,FALSE,'Update');

create or replace view v as
select * from t_1
 union all
select * from t_2
 union all
select * from t_3;

select * from v;

|          1|T|Sample    |
|          2|F|One more  |
|          3|F|Extra     |
|          2|F|Union     |
|          3|F|Intersect |
|          4|F|Except    |
|          3|F|Insert    |
|          4|T|Delete    |
|          5|F|Update    |

insert into v values (6,FALSE,'New');
insert into v values (7,FALSE,'More new');
insert into v values (8,FALSE,'Last new');
delete from v where b;
update v set c = '3rd' where i=3;

select * from v;

|          2|F|One more  |
|          3|F|3rd       |
|          6|F|New       |
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
|           7|F|More new  |
|           8|F|Last new  |
|           2|F|Union     |
|           3|F|3rd       |
|           4|F|Except    |
|           3|F|3rd       |
|           5|F|Update    |

delete from v where c like '%e%';

select * from v;

|           3|F|3rd       |
|           2|F|Union     |
|           3|F|3rd       |
|           3|F|3rd       |
```

9) Сравнительные результаты выполнения комбинированного запроса приведены в таблице 2.

Таблица 2. Сравнение результатов выполнения комбинированного запроса

Конструкция CORRESPONDING		
Не задана	Задана	
	без опции BY…	с опцией BY…
Первый и второй <составные запросы> должны иметь одинаковое количество выбираемых столбцов. Имена столбцов <составных запросов> могут быть разными. Объединение значений столбцов выполняется по порядковым номерам столбцов <составных запросов>.	Первый и второй <составные запросы> могут иметь разное количество выбираемых столбцов. Имена столбцов <составных запросов> и их количество могут быть разными, но, по крайней мере, одно имя столбца должно быть общим для первого и второго <составного запроса>. Столбцы с общими именами в первом и втором <составных запросах> могут иметь разные порядковые номера. Объединение значений столбцов выполняется по общим именам столбцов из <составных запросов> и в том порядке, в каком они перечислены в	Первый и второй <составные запросы> могут иметь разное количество выбираемых столбцов. Имена столбцов <составных запросов> и их количество могут быть разными, но имена столбцов, перечисленные в <списке столбцов>, должны присутствовать во всех <составных запросах>. Если столбец указан в <списке столбцов>, но отсутствует в некоторых <составных запросах>, то в <объединение запросов> он не включается. Столбцы с общими именами в первом и втором <составных запросах> могут иметь разные порядковые номера. Объединение

	первом <составном запросе>.	значений столбцов выполняется по именам столбцов из <списка столбцов> и в том порядке, в каком они перечислены в <списке столбцов>.

10) Разрешено обновление/удаление/вставка над UNION ALL нескольких запросов, если каждый из них является обновляемым, при этом одним запросом реально могут изменяться несколько таблиц. При переходе по выборке к следующей записи выборки данных может происходить смена таблицы для текущей записи в курсоре. Контроль доступа применяется к каждой отдельной таблице, из которой берет данные конструкция UNION ALL.

```
create or replace table t_1 (i int, b boolean, c char(10));
insert into t_1 values (1,TRUE,'Sample');
insert into t_1 values (2,FALSE,'One more');
insert into t_1 values (3,FALSE,'Extra');

create or replace table t_2 (i int, b boolean, c char(10));
insert into t_2 values (2,FALSE,'Union');
insert into t_2 values (3,FALSE,'Intersect');
insert into t_2 values (4,FALSE,'Except');

create or replace table t_3 (i int, b boolean, c char(10));
insert into t_3 values (3,FALSE,'Insert');
insert into t_3 values (4,TRUE,'Delete');
insert into t_3 values (5,FALSE,'Update');

create or replace view v as
select * from t_1
 union all
select * from t_2
 union all
select * from t_3;

select * from v;

|          1|T|Sample    |
|          2|F|One more  |
|          3|F|Extra     |
|          2|F|Union     |
|          3|F|Intersect |
|          4|F|Except    |
|          3|F|Insert    |
|          4|T|Delete    |
|          5|F|Update    |
```

```
insert into v values (6,FALSE,'New');
insert into v values (7,FALSE,'More new');
insert into v values (8,FALSE,'Last new');
delete from v where b;
update v set c = '3rd' where i=3;

select * from v;

|          2|F|One more  |
|          3|F|3rd       |
|          6|F|New       |
|          7|F|More new  |
|          8|F|Last new  |
|          2|F|Union     |
|          3|F|3rd       |
|          4|F|Except    |
|          3|F|3rd       |
|          5|F|Update    |

delete from v where c like '%e%';

select * from v;

|          3|F|3rd       |
|          2|F|Union     |
|          3|F|3rd       |
|          3|F|3rd       |
```

Примеры

```
create or replace table test1(id int, i int, ch char(11), j int,
 nch nchar(13), b boolean);

create or replace table test2(id int, vch varchar(11), k int, nch
 nchar(13), ch char(20), i int);

create or replace table test3(id int, nvch nchar varying(11), l
 int, ch char(13), nch nchar(16), i int, b blob);

insert into test1 values(1,10,'aaa',10,n'naaa',true);
insert into test1 values(2,20,'bbb2',20,n'nbbb',true);
insert into test2 values(2,'bbb1',20,n'nbbb','bbb2',20);
insert into test3 values(3,n'nccc1',30,'ccc',n'nccc2',30,NULL);
```

а) без использования <списка столбцов>.

Общими для всех трех составных запросов являются столбцы id, ch, nch.

В объединенный запрос они попадают в том порядке, в каком перечислены в первом составном запросе.

```
select * from test1
 union all corresponding
select * from test2
 union all corresponding
select * from test3;
```

или

```
table test1
union all corresponding
table test2
union all corresponding
table test3;
```

```
ID          I           CH           NCH
--          -           --           ---
|     1   |   10      |     aaa    |    naaa       |
|     2   |   20      |     bbb2   |    nbbb       |
|     2   |   20      |     bbb2   |    nbbb       |
|     3   |   30      |     ccc    |    nccc2      |
```

б) с использованием <списка столбцов>.

Общими для все трех составных запросов являются столбцы `ch`, `id`. Столбец `i`, хотя и указан в <списке столбцов>, в <объединение запросов> не попадает, так как отсутствует во втором и третьем составном запросе. Порядок столбцов в <объединении запросов> определяется первым составным запросом, а не списком столбцов в опции `BY`....

```
select * from test1
 union all corresponding by (ch,id,i)
select * from test2
 union all corresponding by (id,ch)
select * from test3;
```

или

```
table test1
union all corresponding by (ch,id,i)
table test2
union all corresponding by (id,ch)
table test3;
```

```
 ID          CH
 --          --
|          1|aaa                |
|          2|bbb2               |
|          2|bbb2               |
|          3|ccc                |
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

Разность запросов

Функция

Исключение из результата первого запроса записей, принадлежащих результату второго запроса.

Спецификация

[1] <разность запросов> : : =
[(]<составной запрос> EXCEPT [ALL | DISTINCT]
[CORRESPONDING [BY (<имя столбца> [, …])]] <составной запрос>[)]

Синтаксические правила

1) Все `<составные запросы>` одной `<разности запросов>` должны иметь одинаковое количество совместимых по типу данных столбцов (различными могут быть только имена столбцов).
2) Описание конструкции `CORRESPONDING` приведено в пункте «Объединение запросов».

Общие правила

1) `<Разность запросов>` получается следующим образом:
   - результат инициализируется пустой таблицей;
   - в результат вставляются все строки, которые есть в первом `<составном запросе>` и которых нет во втором `<составном запросе>`.
2) Если указан `DISTINCT`, из результата исключаются дубликаты строк.
3) Если указан `ALL`, каждая запись выдается в том количестве, в котором она содержится в первом множестве, минус суммарное количество в остальных множествах.

   Пусть есть две таблицы:

```
tab1    tab2
  1       2
  2       4
  3       5
  4       7
  3
```

select * from tab1 except all select * from tab2; или table tab1 except all table tab2;	select * from tab1 except distinct select * from tab2; или table tab1 except distinct table tab2;

`|1 |`
`|3 |`
`|3 |`
	`|1 |`
	`|3 |`

4) Отсутствие ALL и DISTINCT равнозначно DISTINCT.
5) В `<разности запросов>` `<GROUP BY-спецификация>` может использоваться только один раз. Она располагается в последнем операторе `SELECT` и применяется ко всему результату. Столбцы упорядочения в этом случае задаются только путем указания их порядковых номеров.

6) I-й столбец результата <разности запросов> будет именован только в том случае, если среди исходных i-х столбцов есть хотя-бы один именованный столбец. В качестве имени результирующего столбца будет взято имя первого именованного столбца из операндов комбинированного запроса.

Пересечение запросов

Функция

Определение пересекающихся результатов двух запросов.

Спецификация

[1] <пересечение запросов> ::=
[(] <составной запрос> INTERSECT [ALL | DISTINCT]
[CORRESPONDING [BY (<имя столбца> [, …])]] <составной запрос>[)]

Синтаксические правила

1) Все <составные запросы> одного <пересечения запросов> должны иметь одинаковое количество совместимых по типу данных столбцов (различными могут быть только имена столбцов).
2) Описание конструкции CORRESPONDING приведено в пункте «Объединение запросов».

Общие правила

1) <Пересечение запросов> получается следующим образом:
   - результат инициализируется пустой таблицей;
   - в результат вставляются все строки, которые есть и в первом <составном запросе>, и во втором <составном запросе>.
2) Если указан DISTINCT, из результата исключаются дубликаты строк.
3) Если указан ALL, каждая запись выдается в наименьшем количестве, в котором она содержится в одном из <составных запросов>.

   Пусть есть две таблицы:

```
tab1    tab2
   1       2
   2       4
   3       5
   4       7
   3       4
   4
```

   select * from tab1 intersect all select * from tab2;

```
|2 |
|4 |
|4 |
```

   select * from tab1 intersect distinct select * from tab2;

```
|2 |
|4 |
```

4) В <пересечении запросов> <GROUP BY-спецификация> может использоваться только один раз. Она располагается в последнем операторе SELECT

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

и применяется ко всему результату. Столбцы упорядочения в этом случае задаются только путем указания их порядковых номеров.

5) I-й столбец результата <пересечения запросов> будет именован только в том случае, если среди исходных i-х столбцов есть хотя-бы один именованный столбец. В качестве имени результирующего столбца будет взято имя первого именованного столбца из операндов комбинированного запроса.

Подзапрос

Функция

Определение подзапроса.

Спецификация

[1] <подзапрос>::= (<запрос выборки>)

Синтаксические правила

1) Если <подзапрос> указан в предикате, отличном от <предиката существования>, то он должен содержать ровно один столбец.

2) Максимально допустимый уровень вложенности SQL-запросов равен 32.

```
table (table (table (table auto))) limit 2;
```

3) В подзапросах поддерживается сортировка ORDER BY.

```
select i from (select i,rownum as j from test1 order by i) where j
 between 2 and 4 order by i;
```

4) В подзапросах поддерживается конструкция FETCH FIRST (в том числе, одновременно с ORDER BY).

```
select sname,
 (select pname from p, sp where s.snum = sp.snum and p.pnum =
 sp.pnum order by qty desc fetch first 1)
from s;
```

Такой подзапрос должен возвращать один столбец и максимум одну строку для каждой комбинации записей таблиц, стоящих во FROM (в данном примере – для каждой записи таблицы P).

Примеры

```
SELECT * FROM Person AS P
 WHERE EXISTS (SELECT * FROM Auto
                 WHERE PersonID =P.PersonID);
  Table Person AS P WHERE EXISTS (table Auto WHERE PersonID
 =P.PersonID);

SELECT * FROM Person WHERE Salary = (SELECT MAX(Salary) FROM
 Person);

select count(*) from auto where (make, year) in (select 'FORD',
 70);
 table person WHERE Salary = (SELECT MAX(Salary) FROM Person);
```

```
select * from (select (select count(*) from sp where
 s.snum=sp.snum) n from s);
```

Тип выборки

Функция

Определение допустимых режимов доступа к выбираемым данным.

Спецификация

[1] <тип выборки>::=
FOR UPDATE [OF <имя столбца> [, …]] | FOR BROWSE

Синтаксические правила

1) Модификатор `FOR UPDATE` применяется к `<запросу выборки>` только с одноперменным `<табличным выражением>` или обновляемым представлением.
2) Модификатор `FOR BROWSE` применяется к `<запросу выборки>` с одно и многопеременным `<табличным выражением>`.

Общие правила

1) При задании модификатора `FOR BROWSE` текущие строки таблиц, использованные для получения данных ответа, в момент их считывания для ответа блокируются от изменения параллельно работающими пользователями БД (каналами БД). Такие строки таблиц можно только читать, т.е. для них задается разделяемая блокировка. На каждую строку таблицы может быть наложено несколько разделяемых блокировок. Блокировка действует только на текущую строку ответа, т.е. если приложение обрабатывает некоторую текущую запись, то эта запись не может быть изменена. После того, как приложение перешло на другую запись, эта запись блокируется, а с предыдущей блокировка снимается.
2) При задании модификатора `FOR UPDATE` все строки таблицы, вошедшие в выборку, сразу же блокируются от изменения параллельно работающими пользователями БД (каналами БД). Такие строки можно читать (для них задается внутренняя блокировка). На каждую строку может быть наложена только одна блокировка `FOR UPDATE`. Если число строк в выборке превышает 1000, то в таком же режиме блокируется вся таблица. Блокировка действует до обработки запроса `COMMIT/ROLLBACK`.
3) При задании модификатора `FOR UPDATE` для необновляемых запросов он игнорируется, никакие блокировки не накладываются. На консоль ядра СУБД ЛИНТЕР выдается сообщение о неверном модификаторе запроса.

Ограничение выборки

Функция

Определение максимального количества выбираемых записей выборки данных.

Спецификация

[1] <ограничение выборки>::=
FETCH FIRST <объем выборки> [PERCENT] [WITH TIES]
| LIMIT [<начало выборки>,] <количество записей>
[2] <объем выборки>::= <количество записей>|<процент выборки>

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

[3] <начало выборки>::= <целочисленный литерал>
[4] <количество записей>::=
неотрицательный целочисленный литерал | -1
[5] <процент выборки>::= неотрицательный целочисленный литерал

Синтаксические правила

1) <Начало выборки> задает относительный номер (отсчет начинается с 0) записи выборки данных, начиная с которой должна выполняться выборка записей.
2) Если <начало выборки> не задано, по умолчанию принимается значение 0. В конструкции FETCH FIRST <начало выборки> равно 0.

Конструкция

```
SELECT personid, model FROM auto LIMIT 0,7;
```

эквивалента

```
SELECT personid, model FROM auto LIMIT 7;
```

3) <Количество записей> задает максимальное количество выбираемых записей выборки данных. <Количество записей>, равное -1, означает выбор записей выборки данных с <начала выборки> и до конца результата (в этом случае <начало выборки> является обязательным).

```
select personid, model from auto limit 998,-1;

|999  |BUICK SKYLARK V8 |
|1000 |CHEVROLET IMPALA |

! Эти конструкции эквивалентны (выбираются все записи):
select * from rank_tst limit -1;
select * from rank_tst limit 0,-1;
select * from rank_tst fetch first -1;
```

4) Если <количество записей> равно 0, возвращается пустое множество записей выборки данных.
5) <Процент выборки> – целочисленное положительное значение в диапазоне [0, 100] Значение <процента выборки> при необходимости округляется с избытком (например, 50% от 3-х записей будет равно 2-м записям).
6) Конструкция LIMIT с одним аргументом задает максимальное количество возвращаемых записей выборки данных, т.е. конструкция LIMIT n эквивалентна LIMIT 0,n.

```
Эти конструкции эквивалентны (выбирается 0 записей);
select * from rank_tst limit 2;
select * from rank_tst limit 0,2;
select * from rank_tst fetch first 2;
```

Общие правила

1) Если <количество записей> превышает объем выборки данных, то выдаются все записи выборки данных.
2) При указании опции PERCENT выдается указанный процент от всех записей с <начала выборки>.

3) Опция WITH TIES заставляет выбрать указанное число или указанный процент записей с <начала выборки>, включая все последние совпадающие записи в отсортированной выборке (требуется опции ORDER BY).

```
create or replace table test (i int);
insert into test values (1);
insert into test values (2);
insert into test values (3);
insert into test values (3);
insert into test values (3);

select * from test order by i
 fetch first 3;
или
select * from test order by i
 fetch first 42 percent;
```

```
 I
 -
|          1|
|          2|
|          3|
```

```
select * from test order by i
fetch first 3 with ties;
или
select * from test order by i
fetch first 42 percent with ties;
```

```
 I
 -
|          1|
|          2|
|          3|
|          3|
|          3|
```

4) Если <начало выборки> превышает реальный объем выборки, то выдаётся пустая выборка данных.

5) Конструкция <ограничение выборки> применима ко всем подзапросам

```
select model from auto where make in (select make from auto fetch
 first 5);
```

6) Рекомендуется использовать ключевое слово LIMIT только в том случае, если предполагается, что выборка для запроса, в котором по сравнению с исходным запросом все внешние соединения заменены внутренними, должна быть непустой. Если же эта выборка может оказаться пустой, то рекомендуется либо не использовать ключевое слово LIMIT, либо использовать хинт /* +NOITER */.

Примеры

1)

```
select count(*) from auto;

|1000 |
```

2)

```
select rowid, make, color from auto fetch first 2;

| 1 | FORD   | BLACK |
| 2 | ALPINE | WHITE |
```

3)

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
select rowid, make, color from auto
  where rowid > 2 fetch first 3;

| 3 | AMERICAN MOTORS | BROWN |
| 4 | MASERATI        | BLACK |
| 5 | CHRYSLER        | WHITE |
```

4)

```
select personid, model from auto limit 900,3;

| PERSONID | MODEL              |
| 901      | CADILLAC DE VILLE  |
| 902      | BUICK SKYLARK V8   |
| 903      | CHEVROLET IMPALA   |

create or replace table test (i int);
insert into table values (1);
insert into table values (2);
insert into table values (3);
insert into table values (4);
insert into table values (5);

select * from test fetch first 75 percent;

 I
 -
|          1|
|          2|
|          3|
|          4|

select distinct make, model, cylnders
  from auto order by cylnders desc fetch first 1;
select distinct make, model, cylnders
  from auto order by cylnders desc fetch first 1 with ties;
select distinct make, model, cylnders
  from auto order by cylnders desc fetch first 10 percent;
select distinct make, model, cylnders
  from auto order by cylnders desc fetch first 10 percent with
 ties;
```

Управление оптимизацией выполнения запросов

В СУБД ЛИНТЕР оптимизация выполнения запросов осуществляется на двух уровнях:

1) внутренний уровень – оптимизация выполнения запросов ядром СУБД на основе синтаксиса SQL-запроса. При использовании этого метода учитывается иерархическое старшинство операций. Если для какой-либо операции существует

более одного пути ее выполнения, то выбирается тот путь, чей ранг выше, так как в большинстве случаев он выполняется быстрее, чем путь с более низким рангом;

2) внешний уровень – с помощью подсказок программиста. Оптимизатор не может использовать индексный путь доступа, основываясь только на существовании индекса; путь доступа должен стать известным при обработке SQL-запроса. Для этого используется механизм «подсказок» (`hints`).

Подсказки могут располагаться как внутри текста запроса (между ключевыми словами и элементами выражений), так и в конце. Текст запроса не может начинаться с подсказки.

Допустимое местоположение подсказок:

```
select   count (*)  from auto /*+ NOCACHE */;
select /*+ NOCACHE */ count (*) from auto;
select count (*) from /*+ NOCACHE */ auto;
select count (* /*+ NOCACHE */ ) from auto;
select sysdate- /*+ NOCACHE */ 7;
```

Недопустимое местоположение подсказок:

```
/*+ NOCACHE */ Select   count (*)  from auto;
select cou /*+ NOCACHE */ nt (*) from auto;
```

Комментарий к подсказкам может быть как строковым ('--'), так и блочным ('/*, */'). Строковый комментарий исключает из выполнения только одну строку, перед которой стоит признак комментария, блочный комментарий исключает из выполнения целый блок команд, заключенный в указанную конструкцию.

```
select * from t1, t2, t3, t4 where
t1.id=t2.id -- +PREDORDER = 3
and
t3.value=t4.value /*+PREDORDER=2*/ and
t2.id=t3.id /*+PREDORDER=4*/ and
t3.id=t4.id /*+PREDORDER=1*/;
```

Подсказка очередности слияния результатов вычисления предикатов

Функция

Управление очередностью объединения результатов вычисления предикатов.

Спецификация

[1] <подсказка очередности>::=
{-- +PREDORDER=<номер> | /* +PREDORDER=<номер> */}
[2] <номер>::= целочисленное значение в диапазоне 1-255

Синтаксические правила

1) Комментарий может быть:

- строковый: исключает из выполнения только одну строку, перед которой стоят символы -- (два минуса);

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

- блоковый: исключает из выполнения целый блок команд, заключенный между символами /* и */.

```
select * from t1, t2, t3, t4 where
t1.id=t2.id -- + PREDORDER = 3
and
t3.value=t4.value /*+PREDORDER=2*/ and
t2.id=t3.id /*+PREDORDER=4*/ and
t3.id=t4.id /*+PREDORDER=1*/;
```

Общие правила

1) Первыми будут объединяться результаты предикатов с наименьшими номерами. Если номер у предиката не задан (например, только у части предикатов группы заданы номера с помощью подсказок), то очередность будет определяться стандартным образом, но результаты «непронумерованных» предикатов будут добавляться в выборку данных после результатов «пронумерованных».
2) Порядок задания приоритетов должен устанавливаться экспериментальным путем.

Пример реального запроса пользователя СУБД ЛИНТЕР, который не мог быть оптимизирован без подсказок:

```
SELECT T_AV."Месяц" as
"Месяц","ФОТ нач.","ФОТ к выд.","$ ФОТ к выд.",
"АВАНС округл." as "Аванс", "ФОТ к выд."-"АВАНС округл." as "Ост.
 выдать",
"Отч. в ФКП","Отч. в ЛОФ","ФСП не менее","Ср. курс $","Сумм. ТЗ",
"Рабоч.ч.",
round ("ФОТ к выд."/"АВАНС", 2) as "Коэфф."
FROM (
select t6.mon_id as "Месяц",
ceil(sum(zp_predl_r)) as "ФОТ нач.",
ceil(sum(zp_na_ruki)) as "ФОТ к выд.",
ceil(sum(PREM_KVART)) as "Отч. в ФКП",
ceil(sum(OTPUSK)) as "Отч. в ЛОФ",
ceil(sum(zp_predl_r)*3.0/97.0) as "ФСП не менее",
ceil(sum(zp_na_ruki)/avg(SR_KURS_DOLL)) as "$ ФОТ к выд.",
round(avg(SR_KURS_DOLL),2) as "Ср. курс $"
from svodn, tab6_time as t6
WHERE
svodn.mon_id=t6.mon_id
group by t6.mon_id ) as T_OTCH,
           /*order by t6.mon_id desc;*/
 (select mon_ID as "Месяц", sum("АВАНС округл.") as "АВАНС
 округл.",
sum("Сумм. ТЗ") as "Сумм. ТЗ", sum("Рабоч.ч.") as "Рабоч.ч.",
sum("АВАНС") as "АВАНС"
from
(
```

```
select t1.MON_ID,
t6."WH" as "Рабоч.ч.", t4."S_MIN_TZ" as "Сумм. ТЗ",
(t2.DO+t1.MLN)*(t4."S_MIN_TZ"/t6."WH")*(0.9-1.0/12.0) as "АВАНС",
floor(
(t2.DO+t1.MLN)*(t4."S_MIN_TZ"/t6."WH")*(0.9-1.0/12.0)/50.0
)*50 as "АВАНС округл."
from TAB1_USERS as t0, TAB1_USERS_INCR as t1, TAB2_DOL_INCR as t2,
(select mon_id, user_id, sum(MIN_TZ) as "S_MIN_TZ"
     from   SVODN, PR_DATA
      where (  SVODN.PR_ID=PR_DATA.PR_ID)  and (PR_DATA.PR_NAME
 not like
'%-Бонус%')  and (PR_DATA.PR_NAME not like '%-бонус%')
  group by mon_id, user_id) as t4,
TAB6_TIME as t6
where

/*  вариант без подсказок - оптимизация не выполняется */
                                        /*
t1.MON_ID=t2.MON_ID and
t1.DOL_ID=t2.DOL_ID and
t0.USER_ID=t1.USER_ID and
t1.MON_ID=t4.MON_ID and t0.USER_ID=t4.USER_ID
and t1.MON_ID=t6.MON_ID
                                        */
/* вариант с подсказками */
t1.MON_ID=t6.MON_ID and -- +PREDORDER=4
t1.MON_ID=t2.MON_ID and /*+PREDORDER=6*/
t1.MON_ID=t4.MON_ID and /*+PREDORDER=3*/
t1.DOL_ID=t2.DOL_ID and /*+PREDORDER=5*/
t0.USER_ID=t1.USER_ID and /*+PREDORDER=2*/
t0.USER_ID=t4.USER_ID /*+PREDORDER=1*/
)
group by mon_id) as T_AV
where T_AV."Месяц"=T_OTCH."Месяц"
order by T_AV."Месяц" desc;
```

Подсказка о вычислении предиката последним

Функция

Оптимизация выполнения поискового запроса путем управления очередностью выполнения предикатов.

Например, пусть:

- в запросе к нескольким таблицам есть предикат "таблица1"."дата" between 'дата1' and 'дата2';
- "таблица1" самая большая по количеству записей в запросе;

- других однопеременных предикатов по "таблице1" в запросе нет;
- но есть предикаты соединения "таблицы1" с другими таблицами по ключевым полям;
- в диапазон между значениями 'дата1' и 'дата2' в "таблице1" попадает большая часть содержащихся в этой таблице записей.

Тогда рекомендуется использовать подсказку /* +LAST */ (обрабатывать предикат последним) для предиката "таблица1"."дата" between 'дата1' and 'дата2'.

В этом случае сначала будут обработаны другие предикаты запроса, которые, возможно, сильно уменьшат объем выборки соединяемых таблиц и только потом к этой уменьшенной выборке будет применен предикат с подсказкой /* +LAST */. В противном случае предикат типа "таблица1"."дата" between 'дата1' and 'дата2' может быть обработан одним из первых и по его условию в выборку может попасть почти вся очень большая таблица, и только потом к этой большой выборке начнется применение остальных предикатов.

Спецификация

[1] <подсказка вычисления очередности предиката>::=
{-- +LAST | /* +LAST*/}

Синтаксические правила

1) Подсказка применима к любому предикату (хотя для некоторых из них применять её нет смысла; в этом случае она бут проигнорирована).
2) Подсказку /* +LAST */ имеет смысл указывать для однопеременного малоселективного предиката по очень большой таблице. Однопеременный малоселективный запрос отбирает почти все (или большинство) записей таблицы. Например, есть таблица с информацией о жителях города Воронеж. Предикат типа "город прописки" = 'Воронеж' – малоселективный, так как почти не уменьшает объем выборки, аналогично предикаты типа "пол" = 'мужской' или "пол" = 'женский' – тоже малоселективные. А, например, предикат типа "город прописки" = 'Липецк' – сильно селективный, объем выборки будет существенно меньше.
3) Подсказка должна указываться после предиката, к которому должна быть применена, например:

```
... and between <значение1> and <значение2> /* +last */ ...
```

Общие правила

1) Подсказка применяется после обработки всех предикатов в группе предикатов без подсказки /* +LAST */, но соединенных по условию AND с предикатом с подсказкой +LAST.
2) Проверка предикатов с подсказкой /* +LAST */ выполняется путем просмотра выборки записей, полученной при обработке остальных предикатов группы, то есть для этих предикатов очередность по отношению друг к другу не имеет большого значения.

```
select distinct make, count(*) from auto
  where auto.year between  60 and 80 /* + last */
    and auto.color in ('BLACK', 'WHITE')
    and model='PANTERA' group by make;

 MAKE
```

```
 ----
|DE TOMASO       |    6|

Помеченный предикат between будет обработан последним.
```

3) При задании логических условий, отличных от AND, подсказка будет действовать только внутри группы предикатов, соединенных по AND.

```
select distinct make, count(*) from auto where
 auto.year between 60 and 80 /* + last */
 and auto.color in ('BLACK', 'WHITE')
 or model='PANTERA' /* + last */  group by make;

 MAKE
 ----
|ALPINE          |    2|
|AMERICAN MOTORS |   32|
|BMW             |    2|
...
```

4) В некоторых случаях использовать подсказку /* +LAST */ нет смысла, например, в таблице создан составной индекс по значениям некоторого столбца и соответствующей этому значению дате, причем в индексе дата указана второй, и отбор по значениям столбца является сильно селективным. Тогда лучше вместо подсказки +LAST использовать такой составной индекс.

Подсказка о раскрытии представления

Функция

«Раскрытие» представления (VIEW) в случае наличия у него только простых предикатов.

Например, пусть создано представление test_view:

```
create or replace table test1( i1 int, j1 int );
insert into test1 values (1, 1);
insert into test1 values (2, 2);
create or replace table test2( i2 int, j2 int );
insert into test2 values (1, 0);
insert into test2 values (2, 1);
insert into test2 values (3, 2);
create or replace view test_view as select i1, i2 from test1,
 test2 where j1=j2;
```

Запрос типа

```
select * from test_view where i2 = 2;
```

после подстановки текста представления будет иметь вид:

```
select i1, i2 from (select i1, i2 from test1, test2 where j1=j2)
 where i2=2;
```

При использовании подсказки запрос будет преобразован к следующему виду:

```
select i1, i2 from test1, test2 where i2=2 and j1=j2;
```

Спецификация

[1] <подсказка раскрытия>::=
{-- +EXPAND([<имя схемы>.]<имя представления> ON |OFF)
| /* +EXPAND([<имя схемы>.]<имя представления> ON | OFF) */}

Синтаксические правила

1) Если <имя схемы> не задано, используется текущая схема (имя пользователя, подавшего запрос).
2) OFF – «раскрытие» представления запрещено.
3) ON – «раскрытие» представления будет произведено, если это возможно (даже если есть сложные предикаты).
4) Подсказка должна размещаться в конце текста запроса (перед точкой с запятой).

Примеры

1)

```
select * from test_view
 where i2 = 2 /*+EXPAND(TEST_VIEW OFF)*/;
```

«Раскрытие» представления запрещено.

2)

```
select * from test_view
 where i2 in (select $$$S11 from $$$SYSRL) /*
+EXPAND(SYSTEM.TEST_VIEW ON)*/;
```

«Раскрытие» представления будет произведено, если это возможно.

Подсказка о сортировке по заданному индексу

Функция

Управление сортировкой записей в выборке данных соответственно указанному индексу без использования конструкции ORDER BY.

Спецификация

[1] <подсказка сортировки по индексу>::=
/* +INDEX [_ASC |_DESC] (<имя таблицы> <имя индекса в таблице>) */

Синтаксические правила

1) Подсказка располагается перед перечислением имен выбираемых столбцов запроса на выборку данных.
2) Подсказка применяется только для именованных индексов.
3) Подсказка применяется только для одного индекса.
4) Подсказки INDEX и INDEX_ASC эквивалентны.

Общие правила

1) В случае невозможности произвести сортировку по указанному индексу подсказка игнорируется.
2) Подсказка задает сортировку с использованием указанного индекса. Сортировка с подсказкой (с использованием индекса) в некоторых случаях будет выполняться быстрее, чем сортировка с помощью конструкции `ORDER BY` (например, в случае, когда выбираемые записи для результирующей выборки хранятся последовательно в нескольких блоках файла данных таблицы).
3) В общем случае запросы с подсказками сортировки будут работать несколько медленнее, чем запросы без подсказок (несортированные) из-за дополнительной сортировки.
4) Использование подсказки вместе с конструкцией `ORDER BY` бессмысленно, так как в этом случае будет проведена повторная сортировка.
5) В ряде случаев подсказка будет игнорироваться (например, когда для обработки запроса ядру СУБД ЛИНТЕР требуется предварительная сортировка).

Пример

```
create or replace table test( id int, ident int );
create or replace table test1( id int, ident int );
insert into test( id, ident) values ( 1, 2 );
insert into test( id, ident) values ( 2, 3 );
insert into test( id, ident) values ( 3, 1 );
insert into test1( id, ident) values ( 1, 1 );
create index "AAA" on test(ident, id);
create or replace table test2( id2 int, ident2 int );
insert into test2( id2, ident2) values ( 1, 3 );
insert into test2( id2, ident2) values ( 2, 1 );
insert into test2( id2, ident2) values ( 3, 2 );
create index "AAA2" on test2(ident2, id2);

select /*+INDEX_DESC(system.test aaa)*/ * from test where ident is
 not null;
select /*+INDEX_DESC(system .test aaa)*/ a.id, a.ident from test
 a, test1 b where b.ident is not null;
select /*+INDEX_DESC(system.test aaa )*/ a.id, a.ident from test
 a, test1 b;
select /*+INDEX(system.test aaa)*/ * from test1 b, test a;
select /*+ INDEX(TEST AAA)*/* from test, test2 where id=id2;
```

Подсказка о типе сортировки

Функция

Управление сортировкой (по индексу или по данным).

Спецификация

[1] <подсказка о типе сортировки> : := /* +SORT {DATA | INDEX} */

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

Синтаксические правила

1) Подсказка располагается в конце SQL-оператора.
2) Конструкция `SORT DATA` принудительно отключает сортировку по индексам (даже если индексы существуют).

```
select i from big_test where i between 1000000 and 1010000
 order by i /* +SORT DATA */
```

3) Конструкция `SORT INDEX` включает сортировку по индексам (даже если требуется отсортировать всего одну-две записи). Соответствующие индексы должны существовать.

```
select i from big_test where i between 1000000 and 1000001
 order by i /* +SORT INDEX */;
```

Общие правила

1) Если подсказка о способе сортировки не задана, СУБД ЛИНТЕР самостоятельно определяет стратегию сортировки. В некоторых случаях выбранный способ может оказаться не оптимальным. Исправить ситуацию можно с помощью данной подсказки.

Пример

Пусть таблица содержит 1 млн. записей.

Если SQL-запрос сформулирован таким образом, что обработке подлежит физически непрерывный набор данных (например, данные, хранящиеся последовательно в первой и второй страницах файла данных таблицы, т.е. внесенные в БД за конкретный месяц), то лучше использовать сортировку по данным.

Если же искомые записи разбросаны случайным образом среди 1 млн. записей таблицы, то оптимальной будет сортировка по индексу.

Подсказки о кэшировании запросов

Для исключения трансляции и выполнения часто повторяющихся SQL-запросов пользователь может подсказать СУБД о необходимости кэширования запросов и результатов их выполнения.

Для поддержки механизма кэширования запросов СУБД ЛИНТЕР создает в оперативной памяти две структуры данных для хранения ссылок (идентификаторов) на:

- тексты кэшируемых запросов (ссылки на исходный и оттранслированный текст SQL-запроса);
- результаты выполнения кэшируемых запросов;
- тексты запросов и результаты их выполнения хранятся в рабочих файлах СУБД.

Управление кэшированием выполняется в утилите `gendb` (см. документ «Создание и конфигурирование базы данных»).

При поступлении на обработку очередного SQL-запроса ядро СУБД ищет текст этого запроса в кэше текстов SQL-запросов.

В случае успешного поиска повторная трансляция запроса отменяется, и, если для этого запроса задан режим кэширования результата, то результат выполнения запроса

извлекается из кэша результатов (в противном случае запрос обрабатывается ядром СУБД).

В случае изменения данных в таблице (таблицах) кэшированного запроса, при удалении используемых синонимов, при изменении прав пользователя и тому подобных изменениях, оказывающих влияние на результат выполнения кэшированного запроса, результат запроса удаляется из кэша результатов. При следующей попытке получить результат выполнения кэшируемого запроса этот запрос заново обрабатывается ядром СУБД.

СУБД ЛИНТЕР поддерживает кэширование только DML-запросов (select, insert, update, delete).

Спецификация

[1] <подсказка об отмене кэширования select-запроса>::=
/* +NOCACHE */
[2] <подсказка о кэшировании запроса>::= /* +CACHE */
[3] <подсказка о кэшировании результата запроса>::=
/* +ANSCACHE */

Синтаксические правила

1) Подсказка NOCACHE отменяет кэширование текста конкретного select-запроса (по умолчанию, если задан режим кэширования текстов запроса, то кэшируются тексты всех select-запросов).
2) Подсказка CACHE заставляет кэшировать текст запроса на модификацию объекта БД (SQL-запросы на модификацию объектов БД по умолчанию не кэшируются).
3) Подсказка ANSCACHE заставляет кэшировать результат выполнения конкретного select-запроса (по умолчанию результаты select-запросов не кэшируются).

Общие правила

1) Кэширование запросов построено на побайтовом сравнении претранслированных запросов: если пришедший от пользователя (или от транслятора SQL) претранслированный запрос совпадает с одним из претранслированных запросов в кэше запросов, то результат выполнения запроса:
   - выбирается из кэша результатов, если в СУБД установлен режим кэширования результатов и для данного запроса использовалась подсказка о кэшировании его результата;
   - формируется ядром СУБД путем выполнения запроса.
2) Запросы, имеющие одинаковый текст, но поданные от имени разных пользователей, являются разными запросами для кэша текстов запросов и для кэша результатов запросов.
3) Запрещается кэшировать следующие запросы:
   - запросы к таблицам на удалённом узле. При попытке кэшировать такой запрос на консоль ядра СУБД будет выдано сообщение: «`INFO: can't place answer of remote query into cache. Ignored.`». Запрос будет обработан корректно, но его ответ в кэш результатов запросов занесён не будет;
   - запросы с модификаторами `FOR UPDATE` и `FOR BROWSE`, поскольку использование этих модификаторов предполагает наложение блокировок на исходную таблицу. В случае обнаружения попытки кэшировать такой запрос на консоль ядра СУБД будет выдано сообщение: «`INFO: can't place answer`

of select query with 'for update' or 'for browse' option into cache. Ignored.». Запрос будет обработан корректно, но его ответ в кэш результатов запросов занесён не будет;

- запросы для событий типа SELECT, например,

```
create event "Select_All_From_Auto_Person" as select * from
 auto,person where auto.personid=person.personid;
```

Это ограничение связано с необходимостью выполнять определённые действия при выдаче ответа на такой select-запрос. При попытке кэшировать такой запрос на консоль ядра СУБД будет выдано сообщение: «INFO: can't place answer of event query into cache. Ignored.». Запрос будет обработан корректно, но его ответ в кэш результатов запросов занесён не будет;

- запросы с модификаторами, ограничивающими выборку FETCH FIRST и LIMIT. При попытке кэшировать такой запрос на консоль ядра СУБД будет выдано сообщение: «INFO: can't place part of query answer into cache. Ignored.». Запрос будет обработан корректно, но его ответ в кэш результатов запросов занесён не будет;
- для запросов, результат которых был взят из кэша результатов запросов, нельзя выполнить команды позиционного обновления и удаления (WHERE CURRENT OF). При попытке выполнить такой запрос будет выдан код завершения 1037 («Запрос необновляемый»), а на консоль ядра СУБД будет выдано сообщение «INFO: can't execute 'current of cursor' query because answer of 'select' query was placed into answer cache. Ignored.»;
- запросы, выбирающие BLOB-столбцы. При попытке кэшировать такой запрос на консоль ядра СУБД будет выдано сообщение: «INFO: can't place answer into cache because answer contains BLOB. Ignored.». Запрос будет обработан корректно, но его ответ в кэш результатов запросов занесён не будет.

4) Запрещено использовать чужие кэшируемые:

- результаты выполнения select-запросов;
- претранслированные запросы.

Примеры

1) Удалить данные недельной давности:

```
delete /* +CACHE */  from tst where d=sysdate-7;
```

2)

```
create or replace table t(i int);
insert into t values(1) /* +CACHE */;
select * from t where i= 1 /* +NOCACHE */;
insert into t values(1) /* +CACHE */;
select * from t where i= 1 /* +NOCACHE */;
alter table t rename to t1;
insert into t values(1) /* +CACHE */;
drop table t1;
```

3)

```
Запустить утилиту gendb и выполнить команду
```

```
set answercache 4;
(4 элемента кэша взято для примера).
Запустить ядро СУБД ЛИНТЕР с ключом «/trace=decomp» для задания
 трассировки исполнения SQL-запросов.
create or replace table test(id int, ch char(10));
insert into test values(1,'a1');
insert into test select id + 1,'a' + to_char(id + 1) from test;
! задаем кэширование результата запроса:
select id, ch from test /* +ANSCACHE */;
!код завершения 1037 – запрос не обновляемый:
update test set id=-1 where current of cursor;
!результат этого запроса берется из кэша результатов – см. файл
 'lintrace.log':
select id, ch from test /* +ANSCACHE */;
!добавление новой записи приводит к очистке кэша результата
 запроса:
insert into test;
!результат выполнения этого запроса берется не из кэша – см. файл
 'lintrace.log':
select id, ch from test /* +ANSCACHE */;
! результат выполнения этого запроса берется уже из кэша – см.
 файл 'lintrace.log':
select id, ch from test /* +ANSCACHE */;
```

Комментарии к трассировочному файлу.

Первый select-запрос с подсказкой `ANSCACHE` выполняет запрос и помещает результат запроса в кэш. Для выдачи результата запроса использует элемент этого кэша. Соответствующий вывод в файле «`lintrace.log`»:

```
C#2 QUERY:
  SELECT
    T_0."ID",
    T_0."CH"
  FROM
    <TABLE "SYSTEM"."TEST" AS T_0>;
C#2 DECOMP.C (Start_Cur_Dec): Now computing derived set #0.
C#2 DECOMP.C (End_Dekart): Derived set #0 is computed, Rows count:
 2.
C#2 FORMOTW.C (FORMOTW): The result of query has been taken from
 answer cache.
C#2 FORMOTW.C (FORMOTW): Read: 18 blocks, write: 33 blocks.
 Additional statistics for read blocks:
  converter 1,index 11,data 5,work 0,sorting 0,blob 0,other 1.
C#2 FORMOTW.C (FORMOTW): Read logical: 82 blocks, write logical:
 65 blocks.
C#2 FORMOTW.C (FORMOTW): Journal read: 1 blocks, written: 17
 blocks.
```

```
C#2 FORMOTW.C (FORMOTW): Time of query execution: 00:00:00:00.02
```

Второй select-запрос с подсказкой ANSCACHE уже не выполняет запрос, он использует результат из кэша:

```
C#2 QUERY:
  SELECT
    T_0."ID",
    T_0."CH"
  FROM
    <TABLE "SYSTEM"."TEST" AS T_0>;
C#2 FORMOTW.C (FORMOTW): The result of query has been taken from
 answer cache.
C#2 FORMOTW.C (FORMOTW): Read: 18 blocks, write: 33 blocks.
 Additional statistics for read blocks:
  converter 1,index 11,data 5,work 0,sorting 0,blob 0,other 1.
C#2 FORMOTW.C (FORMOTW): Read logical: 84 blocks, write logical:
 65 blocks.
C#2 FORMOTW.C (FORMOTW): Journal read: 1 blocks, written: 17
 blocks.
C#2 FORMOTW.C (FORMOTW): Time of query execution: 00:00:00:00.00
```

После выполнения insert-запроса элемент кэша результатов запросов очищается. Последующий select-запрос с подсказкой ANSCACHE заново выполняет запрос и помещает результат запроса в кэш:

```
C#2 QUERY:
  SELECT
    T_0."ID",
    T_0."CH"
  FROM
    <TABLE "SYSTEM"."TEST" AS T_0>;
C#2 DECOMP.C (Start_Cur_Dec): Now computing derived set #0.
C#2 DECOMP.C (End_Dekart): Derived set #0 is computed, Rows count:
 3.
C#2 FORMOTW.C (FORMOTW): The result of query has been taken from
 answer cache.
C#2 FORMOTW.C (FORMOTW): Read: 18 blocks, write: 40 blocks.
 Additional statistics for read blocks:
  converter 1,index 11,data 5,work 0,sorting 0,blob 0,other 1.
C#2 FORMOTW.C (FORMOTW): Read logical: 104 blocks, write logical:
 86 blocks.
C#2 FORMOTW.C (FORMOTW): Journal read: 1 blocks, written: 23
 blocks.
C#2 FORMOTW.C (FORMOTW): Time of query execution: 00:00:00:00.00
```

Четвёртый (последний) select-запрос с подсказкой ANSCACHE не выполняет запрос, он использует результат из кэша:

```
C#2 QUERY:
```

```
  SELECT
    T_0."ID",
    T_0."CH"
  FROM
    <TABLE "SYSTEM"."TEST" AS T_0>;
C#2 FORMOTW.C (FORMOTW): The result of query has been taken from
 answer cache.
C#2 FORMOTW.C (FORMOTW): Read: 18 blocks, write: 40 blocks.
 Additional statistics for read blocks:
  converter 1,index 11,data 5,work 0,sorting 0,blob 0,other 1.
C#2 FORMOTW.C (FORMOTW): Read logical: 106 blocks, write logical:
 86 blocks.
C#2 FORMOTW.C (FORMOTW): Journal read: 1 blocks, written: 23
 blocks.
C#2 FORMOTW.C (FORMOTW): Time of query execution: 00:00:00:00.00
```

Подсказка для отмены итераторной стратегии

Примечание
Поддерживается со сборки 6.0.17.92.

Функция

Отмена итераторной стратегии выполнения запроса.

Спецификация

[1] <подсказка для отмены итераторной стратегии>::=
/* +NOITER */

Общие правила

1) Следует использовать подсказку /* +NOITER */ в том случае, если запрос с конструкцией LIMIT работает медленно (медленнее, чем без LIMIT).
2) Данная подсказка позволяет явно отказаться от итераторной стратегии, несмотря на наличие конструкции LIMIT – на случай, если эта стратегия оказывается неподходящей для какого-то конкретного запроса.

Субъекты БД

Пользователи

Основные понятия

Контроль доступа пользователей к БД основан на следующих понятиях:

1) **Идентификация пользователя**: проверка наличия пользователя с указанным именем (логином) в системной таблице `$$$USR`.
2) **Аутентификация пользователя**: процедура проверки (подтверждения) подлинности пользователя, например: проверка подлинности пользователя путём сравнения введённого им пароля с паролем, сохранённым в БД, или с помощью сервера аутентификации (LDAP, Kerberos).
3) **Авторизация пользователя**: определение и предоставление пользователю, получившему право доступа к БД, его прав доступа к ресурсам БД и управления этим доступом.

Создание пользователя

Функция

Определение оператора создания нового пользователя БД.

Спецификация

[1] <создание пользователя>::=
CREATE [IF NOT EXISTS | OR REPLACE] USER <имя пользователя>
[IDENTIFIED BY {<пароль> | SYSTEM | PROTOCOL | LDAP | KRB}]
[LEVEL (<RAL>, <WAL>)]
[GROUP <имя группы>]
[2] <пароль>::= <символьный литерал>
[3] <RAL>::= <идентификатор>
[4] <WAL>::= <идентификатор>

Общие для всех механизмов аутентификации синтаксические правила

1) Опция OR REPLACE заставляет в случае, если пользователь существует в БД, удалять и пересоздавать его под тем же именем, но с указанными параметрами.
2) Опция IF NOT EXISTS отменяет выполнение оператора, если указанный пользователь уже существует в БД.
3) Для создания нового пользователя необходимо иметь уровень прав DBA.
4) `<Имя пользователя>` должно быть уникальным среди имен пользователей, схем и ролей БД.
5) После отправки клиентским приложением имени и пароля пользователя для его идентификации и аутентификации осуществляется их автоматическое преобразование из кодировки клиентского приложения в кодировку словаря БД. Данная операция критична, например, для кириллических имен пользователей ('Администратор', 'Гость'). В этом случае для словаря СУБД ЛИНТЕР должна быть установлена кодировка, отличная от DEFAULT (кодировки по умолчанию), а в системной таблице `$$$CHARSET` должна присутствовать кодировка, заданная для ОС клиентского приложения.
6) В процессе авторизации пользователю, созданному командой CREATE USER и прошедшему аутентификацию, присваивается CONNECT-категория (то есть доступ

к БД с возможностью подавать SQL-запросы на манипулирование данными с теми объектами БД, владельцы которых предоставили ему такое право).

Примеры

1) это один и тот же пользователь

```
create user GUEST;
create user guest;
create user "GUEST";
```

2) это разные пользователи

```
create user "guest";
create user guest;
```

Синтаксические правила парольной аутентификации

1) <Пароль> не должен превышать 18 символов.
2) <Пароль> регистрозависимый.

Пример

Это разные пароли:

```
create or replace user guest identified by 'qwerty123';
create or replace user guest identified by 'Qwerty123';
```

Общие правила парольной аутентификации

1) Восстановить утерянный <пароль> невозможно.

Примеры

1)

```
create if not exists user GUEST identified by '12345678';
```

2)

```
create user "Гость" identified by 'без пароля';
username "Гость"/"без пароля"
```

Общие правила аутентификации средствами ОС

1) Создание пользователя с аутентификацией средствами ОС (встроенная в ОС аутентификация) возможно только в операционных системах, которые поддерживают эту функциональность. В этом случае в ОС должна быть учетная запись пользователя с таким именем.

Примечание
В ОС Windows пользователь ОС, для которого в базе данных устанавливается режим аутентификации средствами ОС, должен иметь привилегию SE_TCB_NAME («Act as part of the Operating System» – «Работа в режиме операционной системы»).

2) Конструкция устанавливает для пользователя режим встроенной аутентификации операционной системы. Это означает, что при доступе к БД надо указывать имя,

идентичное имени при регистрации в ОС, и тот же пароль, что и при регистрации в ОС. Пароль при этом не хранится в БД. СУБД ЛИНТЕР выполняет только идентификацию пользователя с указанным именем и дает ему и разрешение на доступ к БД, а аутентификацию выполняет ОС.

3) В ОС Linux создание пользователя со встроенной аутентификацией выполняется с помощью разделяемой библиотеки PAM (Pluggable Authentication Modules – подключаемые модули аутентификации). Вследствие этого пользователь ОС, от имени которого запускается ядро СУБД ЛИНТЕР, должен иметь доступ на чтение паролей ОС.

Пример

```
create or replace user "Sakharov" identified by system;
username "Sakharov"/"Password_OS"
```

Общие правила аутентификации BY PROTOCOL

1) Конструкция задает идентификацию и аутентификацию пользователя при проверке доступа к БД по имени пользователя, зарегистрированного в ОС.

2) При запуске клиентских приложений, выполняющих соединение с СУБД ЛИНТЕР, имя пользователя надо задавать пустым (пароль в этом случае запрашиваться не будет).

```
create or replace user "Sakharov" identified by protocol;
inl –u /
```

или

```
inl <Enter>
Имя пользователя   : <Enter>
Пароль пользователя: <Enter>
SQL>
```

3) Пользователь-владелец БД не может идентифицироваться и аутентифицироваться таким способом.

4) Доступ к БД будет проверяться от имени текущего пользователя ОС.



Примечание
Создание пользователя с прозрачной аутентфикацией поддерживается в ОС Windows и ОС Linux, но аутентификация с помощью вышеуказанной конструкции поддерживается только в ОС Linux.

Общие правила аутентификации с помощью LDAP-сервера

1) Конструкция устанавливает для пользователя парольный режим контроля доступа через LDAP-сервер. Для успешной идентификации и аутентификации требуется соответствие предъявленного имени и пароля соответствующим атрибутам одного из пользователей в БД LDAP-сервера. При неудачной попытке аутентификации по любой причине (несоответствие пароля, отсутствие пользователя или недоступность LDAP-сервера) возвращается код завершения 1026 «`Неверный пароль`».

2) При создании пароля на LDAP-сервере необходимо избегать паролей с конечными пробелами, поскольку они интерпретируются СУБД ЛИНТЕР и LDAP-сервером по-разному.

Пример

```
1) Создать в БД LDAP-сервера пользователя, которому необходимо
 предоставить доступ к БД, например:
Имя:  GUEST
Пароль: qwerty123
2) Создать этого же пользователя в СУБД ЛИНТЕР вручную в режиме
 IDENTIFIED BY LDAP, например:
create or replace user GUEST identified by ldap;
3) Для доступа к БД указывать имя и пароль пользователя:
inl –u GUEST/"qwerty123"
```

Общие правила аутентификации с помощью Kerberos-сервера

1) Конструкция устанавливает для пользователя режим идентификации и аутентификации через Kerberos-сервер. Для входа под данным пользователем необходимо указывать пустые имя и пароль пользователя и/или задать специальный флаг в команде соединения с СУБД ЛИНТЕР в интерфейсе нижнего уровня.
2) Для идентификации и аутентификации по Kerberos-протоколу требуется, чтобы ядро СУБД ЛИНТЕР и «Центр распространения ключей» (KDC) Kerberos-сервера находились в ОС одного типа (либо семейства Windows, либо семейства Linux).

Пример

```
1) Ядро СУБД ЛИНТЕР должно быть запущено с аргументом
    /KRBSRVC=<адрес Kerberos-сервиса>
2) Создать в БД Kerberos-сервера пользователя с некими учетными
 данными, например:
Имя:  GUEST
Пароль: qwerty123
3) Создать этого же пользователя в СУБД ЛИНТЕР вручную с
 использованием конструкции IDENTIFIED BY KRB, например:
create or replace user GUEST identified by KRB;
4) Для доступа к БД имя и пароль пользователя не указывать:
  а)
inl –u /
  б)
inl <Enter>
Имя пользователя   : <Enter>
Пароль пользователя: <Enter>
SQL>
  в) утилита «Рабочий стол СУБД ЛИНТЕР»:
В SQL-редакторе
username SYSTEM/MANAGER8
select count(*) from auto;
select count(*) from GUEST.auto; // ошибка – нет доступа

! на компьютере (в ОС) работает пользователь GUEST
username /
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
select count(*) from GUEST.auto; // OK
```

Общие правила предоставления уровней доступа и доверия

1) Назначаемые <идентификаторами> уровень доступа <RAL> и уровень доверия <WAL> (см. документ «Администрирование комплекса средств защиты данных», пункт «Метка субъекта») должны быть определены в системной таблице $$$LEVEL.
2) Если <RAL> и <WAL> не заданы, по умолчанию назначаются пустые метки доступа (это означает, что такой пользователь не будет иметь доступ ни к каким защищаемым метками доступа объектам БД).

Пример

```
select * from $$$level;
$$$ID      $$$NAME          $$$DESCR
----       ----             ----
1          НЕСЕКРЕТНО
2          ДСП
3          СЕКРЕТНО
4          Сов. секретно

create or replace user GUEST identified by '12345678'
level ("Сов. секретно", "СЕКРЕТНО");
```

Общие правила включения в группу пользователей

1) <Имя группы> должно быть определено до добавления в нее пользователя.
2) <Имя группы> задает принадлежность пользователя к указанной группе (с соответствующими этой группе правами доступа к объектам БД).

Пример

```
create user usr1 identified by '12345678';
create user "Гость" identified by '12345678';
create group "Администрация";
create if not exists group "Бухгалтерия"=20;
create group Marketing;
select $$$id, $$$name from $$$group;

$$$ID $$$NAM
---   ---
1     Администрация
2     MARKETING
20    Бухгалтерия

create or replace user GUEST identified by '12345678' group
 "Администрация";
create or replace user USER2 identified by '12345678' group
 Marketing;
```

Удаление пользователя (DROP USER)

Функция

Определение оператора удаления существующего пользователя БД и всех его объектов.

Спецификация

[1] <удаление пользователя>::=
DROP USER <имя пользователя> [CASCADE]

Синтаксические правила

1) Опция `CASCADE` используется для удаления всех объектов пользователя и всех имеющихся в БД ссылок на его объекты.

Общие правила

1) Для удаления пользователя необходимо иметь уровень прав DBA.
2) Удаление создателя БД не допустимо.
3) Запрещено некаскадное удаление пользователя, являющегося владельцем каких-либо объектов БД.
4) Вместе с пользователем удаляются все его привилегии и назначения ролей.
5) При удалении пользователя производится проверка на наличие у него триггеров `after logon/before logoff`. Если триггер (триггеры) есть, и опция `CASCADE`:
   - задана, то пользователь удаляется из БД;
   - не задана, команда отвергается (выдается код завершения 1513 «`Пользователь не может быть удален`»).
6) При удалении пользователя производится проверка на наличие созданных им ролей. Если есть роли, созданные удаляемым пользователем и опция `CASCADE`:
   - задана, то пользователь удаляется из БД;
   - не задана, команда отвергается.

Изменение атрибутов пользователя (ALTER USER)

Функция

Определение оператора изменения атрибутов пользователя БД (пароль, принадлежность к группе, длина пароля, срок действия пароля и другие).

Спецификация

[1] <изменение атрибутов пользователя>::=
<изменение способа аутентификации>
| <изменение характеристик пароля>
| <добавление в группу>
| <блокирование/разблокирование пользователя>
| <количество одновременно открытых каналов>
| <ограничение количества неудачных соединений>
| <мандатный контроль доступа>
| <изменение графика работы пользователя>
| <доступность рабочих станций>

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

| <доступность внешних устройств>
| <кэширование результатов поисковых запросов>

Изменение способа аутентификации

Спецификация

[1] <изменение способа аутентификации>::=
ALTER USER [<имя пользователя>]
IDENTIFIED BY {<пароль> | SYSTEM | PROTOCOL | LDAP | KRB}

Общие правила изменения атрибутов пользователя

1) Изменить свой профиль пользователь может в текущем сеансе. Для изменения профиля другого пользователя необходим уровень привилегий DBA.
2) Если <имя пользователя> не задано, измененный профиль применяется для пользователя, подавшего данную команду.
3) Разрешается указывать одновременно любое количество логически не противоречащих друг другу опций.

```
alter user guest
identified by system
password life unlimited
group "MARKETING"
cursor limit 10
login error limit 3;
```

Синтаксические правила изменения способа аутентификации

1) <Пароль> должен иметь длину не более 18 символов.

Общие правила изменения способа аутентификации

1) Измененный пароль начинает применяться сразу же в текущем сеансе пользователя.

Примечания

1. Изменение администратором БД (или пользователем с привилегией DBA) пароля некоторого пользователя без согласования с самим пользователем не останется незаметным, т.к. сменивший пароль не сможет вернуть обратно старый пароль. Если он (администратор) изменит пароль другого пользователя, то при первой же попытке доступа к БД после изменения пароля этот пользователь не сможет начать работу с СУБД ЛИНТЕР.
2. Если есть несколько пользователей, обладающих правами DBA, то каждый из них может сменить пароль другому (в том числе и создателю БД). Если в БД есть только один пользователь с правами DBA и он забыл свой пароль, то после этого выполнять администрирование БД невозможно.

2) Опция IDENTIFIED BY <пароль> устанавливает новый пароль пользователя. Конструкция используется, как правило, для присвоения пароля пользователям, созданным без аутентификации, или назначения пользователю нового пароля взамен утерянного.

```
alter user "Гость" identified by '12345678';
```

3) Конструкция `IDENTIFIED BY SYSTEM` задает идентификацию и аутентификацию пользователя по имени, зарегистрированному в ОС (см. пункт «Создание пользователя»).

4) Конструкция `IDENTIFIED BY PROTOCOL` задает идентификацию и аутентификацию пользователя по учетным (регистрационным) данным текущего пользователя ОС. В этом случае для доступа к БД необходимо указывать пустые имя и пароль (см. пункт «Создание пользователя»).

5) Конструкция `IDENTIFIED BY LDAP` переводит пользователя в режим аутентификации через LDAP-сервер (см. пункт «Создание пользователя»).

```
alter user USER_NAME identified by LDAP;
grant connect to USER_NAME identified by LDAP;
```

6) Конструкция `IDENTIFIED BY KRB` переводит пользователя в режим идентификации и аутентификации через Kerberos-сервер (см. пункт «Создание пользователя»).

```
alter user USER_NAME identified by KRB;
grant connect to USER_NAME identified by KRB;
```

Изменение характеристик пароля

Спецификация

[1] <изменение характеристик пароля>::=
ALTER USER [<имя пользователя>]
PASSWORD {LENGTH {MIN <длина пароля> | UNLIMITED}
| LIFE {TIME <количество дней> | UNLIMITED}}
[2] <длина пароля>::= целое положительное число
[3] <количество дней>::= целое положительное число

Примечание
Все действия, связанные с измененнием параметров пароля и срока его действия, должны быть согласованы с пользователем, либо администратор БД должен уведомить пользователя о произведенных им изменениях.

Примечание
При задании настроек длины пароля пользователя следует учесть, что в версиях 6.0.20.1 и выше для баз данных по умолчанию включен режим CERTIFIED (см. документ «Создание и конфигурирование базы данных», пункт «Создание БД»), который запрещает задание пароля пользователя длиной менее 8 символов.
Для баз данных, созданных с явно заданной опцией ORDINARY (см. документ «Создание и конфигурирование базы данных», пункт «Создание БД») возможно задание пароля длиной от 0 символов.

Синтаксические правила

1) Опция `PASSWORD LENGTH UNLIMITED` разрешает использовать пароль любой длины в диапазоне от 0 (т.е. без пароля) до 18.

Примечание
При нулевой длине пароля доступ пользователя к БД выполняется без аутентификации (см. пункт «Создание пользователя»).

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

2) Опция PASSWORD LENGTH MIN устанавливает минимальную длину пароля пользователя (от 0 до 18).

```
Изменение длины пароля пользователя, подавшего эту команду:
alter user password length min 6;
```

3) Текущую длину пароля пользователя команда опция PASSWORD LENGTH MIN не проверяет, она влияет только на выполнение последующих команд CREATE USER, ALTER USER или GRANT с ключевым словом IDENTIFIED.

4) Опция PASSWORD LIFE TIME <количество дней> устанавливает длительность действия пароля. <Количество дней> задает время действия пароля (в днях) с момента выполнения данной команды.

```
alter user user1 password life time 30;
```

5) Значение <количество дней> должно быть в диапазоне от 0 до 65534.

6) Длительность действия пароля проверяется только в процессе аутентификации пользователя. Это означает, что текущая сессия пользователя после истечения срока действия пароля прервана не будет и может продолжаться сколь угодно долго.

7) Если обозначена опция PASSWORD LIFE TIME <количество дней>, то заданное значение прибавляется к текущей дате. Полученная дата ограничивает срок действия пароля. Попытка пользователя получить доступ к БД в текущем сеансе после истечения срока действия пароля будет заблокирована – на любую команду, кроме ALTER USER, пользователь будет получать код завершения 1022 («Нарушение привилегий»). После смены пароля по команде ALTER USER доступ пользователя к БД будет возобновлен вплоть до истечения следующего интервала времени действия пароля.

8) Доступ пользователя к БД после окончания срока действия пароля может быть восстановлен в том случае, если администратор БД подаст команду на смену пароля пользователя при помощи оператора GRANT, например:

```
grant dba to TEST identified by '12345678';
```

или если администратор БД (SYSTEM) отменит ограничение на время действия пароля с помощью оператора

```
alter user <имя пользователя> password life unlimited;
```

9) Опция PASSWORD LIFE UNLIMITED отменяет ранее установленный срок действия пароля.

Пример

```
Выполняем блок запросов:
drop user TEST cascade;
create user TEST identified by '12345678';
grant dba to TEST;
alter user TEST password life time 1;
username TEST/12345678
! таблица может быть создана
create or replace table test(i int);

Затем средствами ОС переводим системное время на сутки вперед
 и выполняем блок запросов, в котором первый оператор create
```

```
 table должен быть заблокирован (код завершения 1022 «Нарушение
 привилегий»), а второй – отработать корректно:
! создать таблицу нельзя – истек срок действия пароля
username TEST/12345678
create or replace table test(i int);

username SYSTEM/MANAGER8
! дать пользователю TEST привилегию DBA и установить новый пароль
grant dba to TEST identified by '123456781';
username TEST/123456781
! таблица может быть создана
create or replace table test(i int);
```

Добавление в группу

Спецификация

[1] <добавление в группу>::=
ALTER USER [<имя пользователя>] GROUP <имя группы>

Синтаксические правила

1) Группа должна быть предварительно создана (см. «Администрирование комплекса средств защиты данных», пункт «Создание группы»).

Общие правила

1) Команда включает пользователя в указанную группу. Для её выполнения необходима привилегия DBA.
2) Пользователь может быть включен только в одну группу.
3) Для перемещения пользователя из одной группы в другую необходимо включить его в новую группу.

Пример

```
create group "Бухгалтерия";
alter user user1 group "Бухгалтерия";
```

Блокирование/разблокирование пользователя

Спецификация

[1] <блокирование/разблокирование пользователя>::=
ALTER USER [<имя пользователя>] {LOCK | UNLOCK}

Общие правила блокирования/разблокирования пользователя

1) Команда не применима к создателю БД.
2) Команду может выполнять только создатель БД.
3) Опция `LOCK` блокирует доступ указанного пользователя к БД, опция `UNLOCK` отменяет ранее заблокированный доступ к БД.

```
Пример с использованием утилиты inl:
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
username SYSTEM/MANAGER8
create or replace user TEST identified by '12345678';
username TEST/12345678
username SYSTEM/MANAGER8
alter user TEST lock;

username TEST/12345678
(эта команда не проходит из-за блокировки доступа к БД
 пользователя TEST)
```

4) Блокировка пользователя распространяется на все последующие сеансы СУБД ЛИНТЕР, до его разблокирования по команде `ALTER USER … UNLOCK`.

Количество одновременно открытых каналов

Спецификация

[1] <количество одновременно открытых каналов>::=
ALTER USER [<имя пользователя>] {CONNECT | CURSOR}
{LIMIT <количество каналов> | UNLIMITED}
[2] <количество каналов>::= целое положительное число

Синтаксические правила

1) Опция `CONNECT` управляет количеством доступных пользователю соединений с БД, опция `CURSOR` – количеством доступных пользователю курсоров.

2) Параметр `<количество каналов>` может принимать значения в диапазоне от 1 до 65535. Значение параметра не проверяется на превышение реального количества поддерживаемых БД соединений и курсоров.

Общие правила

1) Конструкция используется для ограничения монопольного использования пользователем соединений или курсоров БД.

```
alter user SYSTEM connect limit 100;
```

2) Опция `UNLIMITED` отменяет ограничение на количество открываемых соединений или курсоров (используется по умолчанию).

Примеры

```
alter user guest connect limit 5;
! узнать максимальное количество доступных открытых соединений с
 БД
select getword($$$s35, 82)
  from LINTER_SYSTEM_USER.$$$USR
 where $$$S32=0 and $$$S34 like 'GUEST';
5

alter user guest cursor limit 12;
select getword($$$s35, 80)
  from LINTER_SYSTEM_USER.$$$USR
```

```
 where $$$S32=0 and $$$S34 like 'GUEST';
12


alter user guest cursor unlimited;
select getword($$$s35, 80)
  from LINTER_SYSTEM_USER.$$$USR
 where $$$S32=0 and $$$S34 like 'GUEST';
0
```

Ограничение количества неудачных соединений

Спецификация

[1] <ограничение количества неудачных соединений>::=
ALTER USER [<имя пользователя>]
LOGIN ERROR {LIMIT <лимит ошибок> | UNLIMITED
| TIMEOUT {NUMBER <длительность блокировки>
| AFTER <количество ошибок>}
| NO TIMEOUT}
[2] <лимит ошибок>::= целое положительное число
[3] <длительность блокировки>::= целое положительное число
[4] <количество ошибок>::= целое положительное число

Синтаксические правила

1) Допустимые значения параметров:
   - <лимит ошибок>: от 1 до 100;
   - <длительность блокировки>: от 1 до 100;
   - <количество ошибок>: от 2 до 100.

Общие правила

1) Конструкция ограничивает попытки несанкционированного доступа к БД (например, злоумышленник под известным ему именем пользователя БД пытается подобрать пароль).
2) Параметры конструкции:
   - <лимит ошибок> задает количество последовательных неудачных попыток соединения с БД, после которых доступ к БД блокируется.

```
alter user TEST login error limit 5;
```

Чтобы разрешить данному пользователю вновь повторить попытки соединиться с БД, администратор СУБД (или любой пользователь с правами DBA) должен разблокировать доступ к БД с помощью команды:

```
ALTER USER <пользователь> UNLOCK;
```

- UNLIMITED – количество неудачных попыток соединения с БД не ограничено (опция по умолчанию);
- TIMEOUT NUMBER <длительность блокировки> задает количество секунд, в течение которых все попытки соединения с БД после <количества ошибок>

будут отвергаться, т.е. удачно соединиться с БД можно будет только по истечении указанного периода времени;

```
alter user guest login error limit 3;
username GUEST/111    // неправильный пароль
username GUEST/222    // неправильный пароль
username GUEST/333    // неправильный пароль
username GUEST/'12345678'// правильный пароль, блокировка доступа
```

- <количество ошибок> задает количество неудачных последовательных попыток соединения с БД, после которых временно блокируется доступ к БД (если задано значение <длительность блокировки>). При достижении лимита неудачных последовательных попыток соединения с БД в журнал linter.out добавляется сообщение вида:

```
SECURITY WARNING: login "TEST" is locked after 4 unsuccessful
 login attempts;
```

- NO TIMEOUT отменяет установленную <длительность блокировки>.

Примечание
Владелец БД не может быть заблокирован конструкцией ALTER USER LOGIN ERROR LIMIT, иначе может оказаться, что некоторые действия по администрированию БД выполнить не сможет уже никто и никогда. При необходимости можно воспользоваться временной блокировкой владельца БД по ALTER USER LOGIN ERROR TIMEOUT. При успешном соединении владельца БД после ряда неудачных последовательных попыток соединения в журнал linter.out добавляется сообщение вида:

```
SECURITY WARNING: database owner's successful login after many
 previous unsuccessful logins
```

3) Изменение текущего значения <лимита ошибок> как в сторону увеличения, так и в сторону уменьшения не сбрасывает (не обнуляет) значение счетчика неудачных попыток доступа к БД.

```
Например, была выполнена команда LOGIN ERROR LIMIT 10. После 5
 последовательных неудачных попыток была выполнена команда
LOGIN ERROR LIMIT 2. Поскольку до этого уже было 5 неудачных
 попыток, то следующая неудачная попытка доступа приведет к
 блокировке пользователя.
```

Мандатный контроль доступа

Спецификация

[1] <мандатный контроль доступа>::=
ALTER USER [<имя пользователя>] LEVEL (<RAL>, <WAL>)

Общие правила

1) Конструкция поддерживается в версии СУБД ЛИНТЕР БАСТИОН (версия с расширенным комплексом средств защиты данных).
2) Назначаемые уровень доступа RAL и уровень доверия WAL должны быть предварительно созданы (см. документ «Администрирование комплекса средств защиты данных», пункт «Метка субъекта»).

Пример

```
select * from $$$level;
$$$ID      $$$NAME          $$$DESCR
----       ----             ----
1          НЕСЕКРЕТНО
2          ДСП
3          СЕКРЕТНО
4          Сов. секретно

alter user GUEST level ("Сов. секретно", "СЕКРЕТНО");
```

Изменение графика работы пользователя

Спецификация

[1] <изменение графика работы пользователя>::=
ALTER USER [<имя пользователя>] {ENABLE | DISABLE} LOGIN
{ALWAYS | <график по времени> | <график по дням>}
[2] <график по времени>::=
FROM <время начала работы>
TO <время окончания работы> [FOR <дни работы>]
[3] <график по дням>::=
[SINCE <дата начала>]|[UNTIL <дата окончания>]
[4] <время начала работы>::= 'HH:MM'
[5] <время окончания работы>::= 'HH:MM'
[6] <дни работы>::= <день недели>{[, <день недели>] ...}
[7] <день недели>::=
'MON' | 'TUE' | 'WED' | 'THU' | 'FRI' | 'SAT' | 'SUN'
[8] <дата начала>::= 'DD.MM.YYYY'
[9] <дата окончания>::= 'DD.MM.YYYY'

Синтаксические правила

1) <График по времени> задается по Гринвичу и округляется в меньшую сторону до ближайшего получасового интервала.
2) Значение '00:00', заданное в качестве конца диапазона времени, рассматривается как окончание суток.

Общие правила

1) Конструкция ENABLE LOGIN разрешает, DISABLE LOGIN запрещает пользователю доступ к БД в указанные периоды времени.
2) Опция ALWAYS устанавливает (ENABLE LOGIN) или запрещает (DISABLE LOGIN) ежедневный круглосуточный период доступа к БД в течение всего времени её существования (при создании пользователя задаётся ENABLE LOGIN ALWAYS).
3) Если опция FROM <время начала работы> не задана, по умолчанию используется начало суток.
4) Если опция TO <время окончания работы> не задана, по умолчанию используется конец суток.
5) Если опция FOR <дни работы> не задана, по умолчанию используется все дни недели.

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

Примеры

```
! 1)
alter user GUEST enable login for 'MON','WED','SUN' since
 '16.06.2010' until '20.06.2010' from '12:50' to '16:45';

! 2) Установка доступности к БД сотрудника GUEST на время
 испытательного срока с 01.04.2018 по 01.06.2018
alter user GUEST enable login since '01.04.2018';
alter user GUEST enable login until '01.06.2018';

! Посмотреть период и длительность испытательного срока
select rowid from $$$usr where $$$s34='GUEST';

44

select cast getraw($$$s35,88,16) as date,
       cast getraw($$$s35,104,16) as date
  from $$$usr where rowid=44;

01.04.2018:00:00:00 01.06.2018:00:00:00

! 3) Посмотреть маску запрещенных для доступа дней недели
alter user U1 disable login for 'SAT','SUN';

! Битовая маска
select 'Доступ запрещен: ' ||
       (case when (GetByte($$$S35,120) & 0x1) = 0
         then '0' else '1' end) ||
       (case when (GetByte($$$S35,120) & 0x2) = 0
         then '0' else '1' end) ||
       (case when (GetByte($$$S35,120) & 0x4) = 0
         then '0' else '1' end) ||
       (case when (GetByte($$$S35,120) & 0x8) = 0
         then '0' else '1' end) ||
       (case when (GetByte($$$S35,120) & 0x10) = 0
         then '0' else '1' end) ||
       (case when (GetByte($$$S35,120) & 0x20) = 0
         then '0' else '1' end) ||
       (case when (GetByte($$$S35,120) & 0x40) = 0
         then '0' else '1' end)
  from LINTER_SYSTEM_USER.$$$USR
  where $$$S32=0 and $$$S34 like 'GUEST';

|Доступ запрещен: 0000011|

! 4) Посмотреть ограничение доступа к БД по дням недели
```

```
create or replace user guest identified by '12345678';

alter user GUEST DISABLE LOGIN
  FROM '01:45' TO '02:15' for 'MON', 'FRI';

select
  case d when 0 then 'MON' when 1 then 'TUE' when 2 then 'WED'
    when 3 then 'THU' when 4 then 'FRI' when 5 then 'SAT'
    when 6 then 'SUN' end, /* день */
  to_char(
    cast (cast (to_date ('30', 'MI') as decimal) * sh as date),
      'HH:MI') /* время начала */ || '-' ||
  to_char(to_date('00:30', 'HH:MI') +
    cast (cast (to_date ('30', 'MI') as decimal) * sh as date),
      'HH:MI') /* время окончания */
from
  (select level-1 d from $$$usr
    start with rowid=1 connect by level < 7), /* дни */
  (select level-1 sh from $$$usr
    start with rowid=1 connect by level < 48),
    /* получасовые промежутки */
  LINTER_SYSTEM_USER.$$$USR
where
  $$$S32=0 and $$$S34 like 'GUEST' and
  ((GetByte($$$S35,18+d*6+sh/8) &
    cast (power(2,mod(sh,8)) as int)) <> 0);

MON 01:30-02:00
FRI 01:30-02:00
```

Доступность рабочих станций

Спецификация

[1] <доступность рабочих станций>::=
ALTER USER [<имя пользователя>]
STATION {LIMIT <лимит станций> | UNLIMITED | <имя станции>}
[2] <лимит станций>::= целое положительное число
[3] <имя станции>::= <идентификатор>

Синтаксические правила

1) Конструкция <доступность рабочих станций> устанавливает ограничение на количество рабочих станций (компьютеров), с которых данный пользователь может работать с БД одновременно:

   - LIMIT <лимит станций> – только с заданного количества станций (значение параметра <лимит станций> в диапазоне от 1 до 100);
   - UNLIMITED – со всех рабочих станций;

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

- <имя станции> – только с указанной станции (или с нескольких конкретных станций, если команда ALTER USER STATION <имя станции> была выполнена несколько раз).

Пример

```
create or replace station "ST1" protocol 'TCPIP' address
 '192.168.0.1';
alter user TEST station "ST1";
alter user TEST station limit 50;
alter user TEST station unlimited;
```

Доступность внешних устройств

Спецификация

[1] <доступность внешних устройств>::=
ALTER USER [<имя пользователя>] DEVICE <имя устройства>
[2] <имя устройства>::= <символьный литерал>

Синтаксические правила

1) Существование <имени устройства> при выполнении команды необязательно – устройство может быть добавлено позднее.

Общие правила

1) Конструкция задает имя устройства, используемого пользователем по умолчанию. Пустое (незаполненное) поле имени устройства в системной таблице $$$USR воспринимается как SY00.
2) Каждое последующее выполнение команды отменяет предыдущее устройство по умолчанию и устанавливает новое.

Примеры

```
alter user GUEST device 'DB00';
! файлы таблицы T1 будут размещены на устройстве DB00
create table T1 …
alter user GUEST device 'DB01';
! файлы таблицы T2 будут размещены на устройстве DB01
create table T2 …

Текущее внешнее устройство по умолчанию

select getstr($$$S35,236,4)
  from LINTER_SYSTEM_USER.$$$USR
 where $$$S32=0 and $$$S34 like 'GUEST';

DB01
```

Кэширование результатов поисковых запросов

Спецификация

[1] <кэширование результатов поисковых запросов>::=
ALTER USER [<имя пользователя>] {ENABLE | DISABLE} ANSWERCACHE

Общие правила

1) Описание механизма кэширования результатов SQL-запросов см. в документе «Архитектура СУБД», пункт «Кэшируемые SQL-запросы» и в пункте «Подсказки о кэшировании запросов».
2) Для управления кэшированием результатов поисковых запросов необходимо иметь привилегию DBA.
3) Опция ENABLE разрешает кэшировать результаты выполнения запросов указанного пользователя <имя пользователя> (в том числе и пользователя SYSTEM).
4) По умолчанию кэширование результатов запросов разрешено всем пользователям, но при выполнении следующих условий:
   - при создании БД задано ненулевое количество кэшируемых результатов запросов (значение параметра ANSWERCACHE не равно нулю – см. документ «Создание и конфигурирование базы данных», пункт «Установка параметров конфигурирования»);
   - в тексте SQL-запроса содержится подсказка о необходимости кэширования результатов запроса (см. пункт «Подсказки о кэшировании запросов»).
5) Опция DISABLE запрещает кэшировать результаты выполнения запросов указанного пользователя <имя пользователя> (в том числе и пользователя SYSTEM).
6) Команда не влияет на кэширование претранслированных запросов (их кэширование выполняется при установке параметра QUERYCACHE конфигурации БД).

Примеры

```
1) Отключение кэширования результатов поисковых запросов
alter user u1 disable answercache;

2) Обратное включение кэширования результатов поисковых запросов
alter user u1 enable answercache;

create or replace table test(id int, ch char(10));
insert into test values(1,'a1');
insert into test select id + 1,'a' + to_char(id + 1) from test;

! Результат этого запроса берется не из кэша (кэш пуст)
select id, ch from test /* +ANSCACHE */;

! Результат этого запроса берется уже из кэша
select id, ch from test /* +ANSCACHE */;

!Запрещаем кэширование результатов запроса
alter user SYSTEM disable answercache;
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
! Результат этого запроса берется не из кэша
select id, ch from test /* +ANSCACHE */;

!Разрешаем кэширование результатов запроса
alter user SYSTEM enable answercache;

! Результат этого запроса берется уже из кэша
select id, ch from test /* +ANSCACHE */;
```

Объекты БД

Схемы

Схема – это именованная группа объектов БД, имеющая владельца.

Каждый пользователь БД может создать одну или несколько схем.

При создании пользователя автоматически создается схема по умолчанию, имя которой совпадает с именем пользователя.

После создания соединения текущей схемой является схема, совпадающая с именем пользователя. При обращении пользователя к объектам, созданным в текущей схеме, имя схемы указывать не обязательно. При обращении к объектам, созданным в отличной от текущей схемы, указание имени схемы объектов является обязательным.

При использовании механизма схем отпадает необходимость жесткой привязки имени объекта к владельцу объекта – пользователю БД (владелец схемы может быть сменен и после этого удален).

Схема позволяет создать два объекта с одинаковым именем, но в разных схемах (например, тестовые таблицы, таблицы с одинаковыми именами для разных структурных подразделений).

Создание схемы (CREATE SCHEMA)

Функция

Определение оператора создания схемы.

Спецификация

[1] <создание схемы>::=
CREATE [IF NOT EXISTS | OR REPLACE] SCHEMA <имя схемы>
[AUTHORIZATION <имя пользователя>]
[DEFAULT CHARACTER SET <имя кодировки>]

Синтаксические правила

1) <Имя схемы> должно быть уникально среди имен существующих схем, пользователей и ролей.
2) Создание схемы допускается только для текущего пользователя, т.е. в опции AUTHORIZATION <имя пользователя> должно быть именем текущего пользователя.
3) Опция DEFAULT CHARACTER SET допустима, но в текущей версии игнорируется. Кодовая страница по умолчанию задаётся либо для БД в целом (set database default character set), либо для каждого активного канала в отдельности (set names).
4) Опция OR REPLACE заставляет удалять существующую в БД схему и создавать её под тем же именем, но с указанными параметрами.



Примечание
Если в схеме существуют объекты, то при выполнении команды OR REPLACE будет выдано сообщение 1513 «Пользователя (или схему) нельзя удалить без

© Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

опции CASCADE из-за существующих объектов». Для удаления схемы совместно с объектами необходимо использовать команду «Удаление схемы» с опцией CASCADE.

5) Опция IF NOT EXISTS отменяет выполнение оператора, если указанная схема уже существует в БД.

Общие правила

1) Указывая имя схемы, пользователь может работать с любым объектом любой из своих схем и с объектами других пользователей при наличии соответствующих прав.
2) Без указания имени схемы допускается работа только с объектами текущей схемы.
3) Рекурсивные схемы не допускаются.
4) Для создания схемы требуется привилегия RESOURCE.

Пример

```
create or replace user TEST identified by '12345678';
grant resource to TEST;
username TEST/12345678
create or replace schema SCHEMA1;
create or replace table TEST(i int);
insert into TEST values(1);

!Должно вернуться значение 1
select * from TEST;

|          1|

set schema SCHEMA1;
create or replace table TEST(i int);
insert into TEST values(2);

!Должно вернуться значение 2
select * from TEST;

|          2|

!Должно вернуться значение 1
select * from TEST.TEST;

|          1|

!Должно вернуться значение 2
select * from SCHEMA1.TEST;

|          2|

username SYSTEM/MANAGER8
drop user TEST cascade;
```

Установка текущей схемы (SET SCHEMA)

Функция

Определение оператора установки текущей схемы.

Спецификация

[1] <установка схемы> : : = SET SCHEMA <имя схемы>

Синтаксические правила

1) <Имя схемы> должно задавать имя существующей схемы.

Общие правила

1) Пользователь может устанавливать текущими только те схемы, владельцем которых он является.
2) После установки текущей схемы, к её объектам можно обращаться без указания имени схемы (т.е. можно, например, использовать в запросе вместо конструкции «имя схемы.TABLE1» просто «TABLE1»).
3) После задания текущей схемы обращение к объектам других схем должно обязательно сопровождаться именем схемы (т.е., например, для пользователя USER_TEST при обращении к таблице «TABLE2», созданной в схеме пользователя по умолчанию, следует писать «USER_TEST.TABLE2»).
4) Текущая схема устанавливается на сессию.
5) Установленная в соединении схема по умолчанию распространяется на все текущие и порождаемые каналы соединения.
6) Если в качестве <имени схемы> задать имя текущего пользователя, то текущая схема будет «сброшена» и текущей схемой будет считаться схема пользователя по умолчанию.

Примечание
Для корректной работы со схемами во вновь создаваемых объектах БД, в которых хранится текст запроса на создание этого объекта (триггеры, процедуры, глобальные события), настоятельно рекомендуется указывать полное имя объекта в формате <имя схемы>| <имя пользователя>.<имя объекта>, т.к. неполное <имя объекта> в один момент времени может указывать на один объект БД, а впоследствии, при смене текущей схемы – на другой. Эта рекомендация не распространяется на представления. В представлении все ссылки к именам таблиц без имен пользователей рассматриваются как ссылки к таблицам создателя представления. Т.е. для того, чтобы корректно работать со схемами, в представлениях надо указывать полные имена таблиц.

Удаление схемы (DROP SCHEMA)

Функция

Определение оператора удаления существующей схемы и всех входящих в неё объектов.

Спецификация

[1] <удаление схемы> : : =
DROP SCHEMA <имя схемы> [RESTRICT] [CASCADE]

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

Синтаксические правила

1) Опция RESTRICT используется для удаления указанной схемы при отсутствии в ней объектов (используется по умолчанию).
2) Опция CASCADE используется для удаления всех объектов указанной схемы и всех имеющихся в БД ссылок на удаляемые объекты схемы.

Общие правила

1) Для удаления схемы необходимо иметь уровень прав DBA.
2) Запрещено некаскадное удаление схемы, содержащей объекты пользователя.
3) Запрещено удаление текущей схемы какого-либо канала.

Роли

В БД, содержащей большое количество (сотни/тысячи) субъектов и объектов (пользователей, таблиц, представлений, хранимых процедур и т.д.), очень сложно определять возможности (или права) каждого субъекта при работе с каждым объектом БД. Для упрощения этой процедуры реализован аппарат ролей.

Роль – это именованный набор прав на множество объектов БД. Введенные роли разбивают всех пользователей БД на группы. Те, кому присвоена определенная роль, образуют естественную группу.

При работе с привилегиями роль рассматривается сначала как пользователь БД. Владелец того или иного объекта БД назначает тот или иной вид доступа для роли как для простого пользователя. После того как роль вобрала в себя все требуемые возможности, она может быть присвоена другому пользователю или группе пользователей.

Примечание
Понятие роли не отменяет возможности назначения или изменения доступа конкретного пользователя к конкретной таблице (минуя роли).

Создание роли (CREATE ROLE)

Функция

Определение оператора создания новой роли.

Спецификация

[1] <создание роли>::=
CREATE [IF NOT EXISTS | OR REPLACE] ROLE <имя роли>

Синтаксические правила

1) <Имя роли> должно быть уникальным среди имен пользователей БД, схем и ролей.
2) Опция OR REPLACE заставляет удалять существующую в БД роль (вместе с назначенными ей привилегиями) и создавать её под тем же именем, но с указанными параметрами (т.е. без назначенных привилегий).
3) Опция IF NOT EXISTS отменяет выполнение оператора, если указанная роль уже существует в БД.
4) Одновременное использование опций IF NOT EXISTS и OR REPLACE запрещено.

Общие правила

1) Для создания роли необходимо иметь уровень прав RESOURCE.

Удаление роли (DROP ROLE)

Функция

Определение оператора удаления существующей роли.

Спецификация

[1] <удаление роли>::= DROP ROLE <имя роли>

Общие правила

1) Удалить роль может только ее создатель.
2) Вместе с ролью удаляются записи обо всех назначенных ей привилегиях и обо всех ее назначениях пользователям и другим ролям.

Назначение роли (GRANT ROLE)

Функция

Определение оператора назначения роли.

Спецификация

[1] <назначение роли>::=
GRANT ROLE <имя роли> TO
{{<имя пользователя> | <имя роли>}[, …] | PUBLIC}

Синтаксические правила

1) Назначаемая роль должна существовать в БД.

```
create user usr1 identified by '12345678';
create user usr2 identified by '12345678';
create user usr3 identified by '12345678';
create role "Отдел системных разработок";
create role "Отдел прикладных разработок";
grant role "Отдел системных разработок" to usr1, usr3;
grant role "Отдел прикладных разработок" to "Отдел системных
 разработок", usr2;
grant role DEMO to public;
```

Общие правила

1) Назначать роль может только ее владелец.

Отмена роли (REVOKE ROLE)

Функция

Определение оператора отмены назначенной роли.

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

Спецификация

[1] <отмена роли>::=
REVOKE ROLE <имя роли> FROM
{{<имя пользователя> | <имя роли>}[, …] | PUBLIC}

Синтаксические правила

1) Отменяемая роль должна существовать в БД.

```
create role "Отдел системных разработок";
create role "Отдел прикладных разработок";
revoke role "Отдел системных разработок" from usr1, usr3;
revoke role "Отдел прикладных разработок" from "Отдел системных
 разработок", usr2;
revoke role DEMO from public;
```

Общие правила

1) Отменять назначение роли может только ее владелец.

Базовые таблицы

Создание таблицы (CREATE TABLE)

Функция

Определение базовой таблицы или глобальной временной таблицы.

Спецификация

[1] <создание таблицы>::=
<создание базовой таблицы>
|<создание глобальной временной таблицы>
|<создание копии базовой таблицы>
[2] <создание базовой таблицы>::=
CREATE {[IF NOT EXISTS | OR REPLACE] TABLE | TABLE [IF NOT EXISTS]}
[<имя схемы>.]<имя таблицы> [<синоним таблицы>]
[<кодировка символьных данных таблицы>]
[<мандатный уровень доступа к таблице>]
[<определение столбца>][, ...]
[<ограничение таблицы>]
[<спецификация параметров таблицы>]
[<загрузка таблицы>]
[3] <кодировка символьных данных таблицы>::=
[CHARACTER SET "<кодировка>"]
[4] <кодировка>::=
{<встроенная кодировка>|<идентификатор кодировки>}
[5] <встроенная кодировка>::= DEFAULT|UTF-8
[6] <идентификатор кодировки>::=
строка системной таблицы $$$CHARSET
[7] <определение столбца>::=
<имя столбца>
<тип данных>
[<подставляемое значение>]
[<значение по умолчанию>]
[<фильтр для фразового поиска>]

[<мандатный уровень доступа к столбцу>]
[<атрибут столбца> [...]]
[8] <тип данных>::=
{
{CHAR | CHARACTER} [(<длина>)] [CHARACTER SET "<кодировка>"]
|{VARCHAR | CHARACTER VARYING | CHAR VARYING} (<длина>)
[CHARACTER SET "<кодировка>"]
|BYTE [(<длина>)]
|{VARBYTE | BYTE VARYING}(<длина>)
|SMALLINT
|{INTEGER | INT}
|BIGINT
|REAL
|DOUBLE [PRECISION]
|FLOAT [(<точность>)]
|{DECIMAL | DEC | NUMERIC}[(<точность>[, <мантисса>])]
|DATE
|BOOLEAN
|{NCHAR | NATIONAL CHAR | NATIONAL CHARACTER} [(<длина>)]
|{NCHAR VARYING | NATIONAL CHAR VARYING|NVARCHAR
|NATIONAL CHARACTER VARYING} (<длина>)
|{BLOB | LONG RAW} [CHARACTER SET "<кодировка>"]
|EXTFILE [ROOT '<каталог>']
}
[9] <подставляемое значение>::=<литерал>
[10] <значение по умолчанию>::=
DEFAULT {[(]<задаваемое значение>[)]|(<вычисляемое значимое>)}
[11] <задаваемое значение>::=
{SYSDATE | NOW}
|{LOCALTIME | LOCALTIMESTAMP}
|CURRENT_DATE
|{CURRENT_TIME | CURRENT_TIMESTAMP}
|NULL
|USER
[12] <вычисляемое значимое>::=
<значимое выражение> | GENERATED BY DEFAULT AS (<значимое выражение>)
[13] <фильтр для фразового поиска>::= DEFAULT FILTER <имя фильтра>
[14] <мандатный уровень доступа к таблице>::= LEVEL (<RAL>, <WAL>)
[15] <мандатный уровень доступа к столбцу>::= LEVEL (<RAL>, <WAL>)
[16] <атрибут столбца>::=
<автоматическое наращивание значений>
|<ограничение столбца>
|<значение из последовательности>
|<генерируемое значение>
[17] <автоматическое наращивание значений>::=
{AUTOROWID | AUTOINC [INITIAL (<начальное значение>)]
| AUTOINC RANGE(<диапазон>)}
[18] <ограничение столбца>::=
<ограничение NULL-значений>
|<ограничение уникальности>
|<ссылочная целостность столбца>
|<контролируемое значение столбца>
[19] <ограничение NULL-значений>::= {NOT NULL | NULL}
[20] <ограничение уникальности>::=
PRIMARY KEY

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

|UNIQUE
|ON UPDATE {SYSDATE | NOW | LOCALTIME | LOCALTIMESTAMP |
CURRENT_DATE
|CURRENT_TIME | CURRENT_TIMESTAMP}

[21] <ссылочная целостность столбца>::=
<спецификация схемы таблицы>[(<имя столбца>[, …])]
[<каскадные ограничения>]

[22] <каскадные ограничения>::=
[ON UPDATE {CASCADE | SET NULL | SET DEFAULT | {NO ACTION | RESTRICT}}]
[ON DELETE {CASCADE | SET NULL | SET DEFAULT | {NO ACTION | RESTRICT}}]

[23] <контролируемое значение столбца>::=
<логическое условие>[<логическое условие> …]

[24] <значение из последовательности>::=
GENERATED [{ALWAYS | BY DEFAULT}] AS IDENTITY (<последовательность>)

[25] <генерируемое значение>::=
GENERATED {ALWAYS | BY DEFAULT} AS {<логическое выражение> | <значимое выражение> | SYS_GUID()}

[26] <начальное значение>::=<целочисленный литерал>

[27] <диапазон>::=<нижняя граница> : <верхняя граница>[, …]

[28] <нижняя граница>::=<целочисленный литерал>

[29] <верхняя граница>::=<целочисленный литерал>

[30] <ссылочная целостность>::=
REFERENCES <спецификация родительской таблицы> [(<имя столбца>[, …])]
[MATCH {SIMPLE | FULL | PARTIAL}] [<спецификация действий>]

[31] <спецификация родительской таблицы>::=
<спецификация схемы таблицы>

[32] <спецификация схемы таблицы>::=<имя схемы>

[33] <спецификация действий>::=
[ON UPDATE {CASCADE | SET NULL | SET DEFAULT | {NO ACTION | RESTRICT}}]
[ON DELETE {CASCADE | SET NULL | SET DEFAULT | {NO ACTION | RESTRICT}}]

[34] <ограничение таблицы>::=
<первичный ключ>
|<уникальный ключ>
|<внешний ключ>
|<логическое условие>

[35] <первичный ключ>::=PRIMARY KEY (<имя столбца>[, ...])

[36] <уникальный ключ>::=UNIQUE (<имя столбца>[, ...]|VALUE)

[37] <внешний ключ>::=
FOREIGN KEY (<имя столбца>[, ...]) <ссылочная целостность>

[38] <логическое условие>::=CHECK (<логическое выражение>)

[39] <спецификация параметров таблицы>::=
<табличный параметр>|<файловый параметр>

[40] <последовательность>::=
[START WITH <начальное значение>] [INCREMENT BY <шаг>]
[MAXVALUE <верхняя граница> | NO MAXVALUE]
[MINVALUE <нижняя граница> | NO MINVALUE]
[CYCLE | NO CYCLE]

[41] <табличный параметр>::=
{MAXROWID | MAXROW | PCTFILL | PCTFREE | BLOBPCT} <беззнаковое целое>

[42] <файловый параметр>::=
{INDEXFILES | DATAFILES | BLOBFILES}
<количество файлов> [(<описатель файла>[, …])]

[43] <описатель файла>::=<имя устройства> [<размер файла>]

[44] <количество файлов>::=<беззнаковое целое>

[45] <размер файла>::=<беззнаковое целое>

[46] <синоним таблицы>::= AS <идентификатор>
[47] <загрузка таблицы>::= AS [()<подзапрос>[)]
[48] <создание глобальной временной таблицы>::=
{CREATE [IF NOT EXISTS | OR REPLACE] GLOBAL TEMPORARY TABLE
| CREATE GLOBAL TEMPORARY TABLE [IF NOT EXISTS]}
[<имя схемы>.]<имя таблицы>
[CHARACTER SET "<кодировка>"]
<список элементов таблицы>
[<действия при завершении транзакции>]
[<мандатный уровень доступа к таблице>]
[49] <список элементов таблицы>::=
(<определение столбца>[, <определение столбца>|<ограничение таблицы>])
[50] <действия при завершении транзакции>::=
ON COMMIT {DELETE | PRESERVE} ROWS
[51] <создание копии базовой таблицы>::=
CREATE [IF NOT EXISTS | OR REPLACE] TABLE <целевая таблица> LIKE <исходная таблица>
[52] <целевая таблица>::= [<имя схемы>.]<имя таблицы>
[53] <исходная таблица>::= [<имя схемы>.]<имя таблицы>

Синтаксические правила

1) Опция `IF NOT EXISTS` отменяет выполнение оператора, если указанная таблица уже существует в БД.
2) Опция `OR REPLACE` заставляет удалять существующую в БД таблицу (вместе со всеми её данными) и создавать новую таблицу под тем же именем, возможно, с новыми параметрами. На удаляемую таблицу не должно быть ссылок из других таблиц/представлений БД.
3) Одновременное использование опций `IF NOT EXISTS` и `OR REPLACE` запрещено.
4) Команда создания таблицы должна включать в себя, по крайней мере, одно `<определение столбца>`.
5) Для создания таблицы пользователь должен иметь уровень прав RESOURCE или DBA.
6) `<Имя таблицы>` должно быть уникальным среди имен существующих в БД таблиц, представлений или их синонимов в пределах `<имени схемы>` либо текущей схемы (при её отсутствии в спецификации).

Примечание
Ядро СУБД при выполнении SELECT-запросов, содержащих группировку или некоторые другие конструкции, создает для своих нужд временные таблицы с именами вида "Table#nnn#", где nnn – системный номер таблицы, поэтому, во избежание конфликтов, не рекомендуется давать пользовательским таблицам похожие имена.

Примеры

```
create or replace table Admin.Auto …
create if not exist table "Петров". Sale …
create or replace table "Ведомость товаров" as Year_Ved …
```

Кодировка символьных данных таблицы

Спецификация

См. спецификацию пункта «Создание таблицы».

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

Синтаксические правила

1) <Идентификатор кодировки> должен быть представлен в системной таблице $$$CHARSET.
2) Для использования <встроенных кодировок> наличие в БД системной таблицы $$$CHARSET не обязательно.

Общие правила

1) <Кодировка> задает имя кодовой страницы – набор символов, в которых будут храниться в БД все символьные данные типа CHAR, VARCHAR таблицы (за исключением столбцов, для которых задана индивидуальная кодировка, см. спецификацию <тип данных>).
2) Если от клиентского приложения символьные данные поступили в иной кодировке, то они будут преобразованы в заданную <кодировку>.
3) При выполнении SQL-запроса выборки данных, хранящиеся в заданной <кодировке> символьные данные результирующей выборки данных преобразуются в кодировку клиентского приложения.
4) При создании системных таблиц для символьных данных всегда используется кодировка системного словаря (по умолчанию это кодировка DEFAULT, она может быть изменена командой SET DATABASE NAMES).
5) Для обычных базовых таблиц, если <кодировка> символьных данных таблицы не задана:
   - используется кодировка, заданная командой SET NAMES;
   - если кодировка не задана командой SET NAMES, то используется кодировка, заданная командой SET DATABASE DEFAULT CHARACTER SET;
   - если обе указанные выше кодировки не заданы, то используется кодировка системного словаря БД, заданная командой SET DATABASE NAMES;
   - если не задана ни одна из кодировок предыдущих пунктов, то используется кодировка DEFAULT (кодировка поддерживает первые 128 символов кодировки "CP437").
6) С помощью кодировки UTF-8 можно обеспечить многоязыковую поддержку пользовательского приложения в СУБД ЛИНТЕР без создания таблицы $$$CHARSET.
7) При задании для столбца многобайтовой кодировки (UTF-8, EUC_JP и т.д.) следует иметь в виду, что размер столбца задаётся в байтах, а каждый символ в многобайтовой кодировке может занимать несколько байтов, т.е. в столбце с многбайтовой кодировкой может поместиться значительно меньше символов, чем размер этого столбца в байтах.

   Например, в кодировке UTF-32 китайские иероглифы представлены 4 байтами, поэтому строка длиной 40 байтов может содержать не более 10 символов-иероглифов.

Примеры

1) Кодировка cp1251 будет преобразована к верхнему регистру и найдена в таблице $$$CHARSET (хранится в таблице как CP1251):

```
create or replace table test character set cp1251 (ch char(10));
```

2) Кодировка cp1251 будет использована как есть и не будет найдена в таблице $$$CHARSET (хранится в таблице как CP1251):

```
create or replace table test character set "cp1251" (ch char(10));
```

3) Кодировка KOI8-R будет использована как есть и будет найдена в таблице $$$CHARSET (кавычки нужны из-за дефиса в имени кодировки) (хранится в таблице $$$CHARSET как KOI8-R):

```
create or replace table test character set "KOI8-R" (ch char(10));
```

Определение мандатного уровня доступа к таблице

Спецификация

См. спецификацию пункта «Создание таблицы».

Синтаксические правила

1) См. документ «Администрирование комплекса средств защиты данных», пункт «Управление уровнем доступа к таблице».

Общие правила

1) См. документ «Администрирование комплекса средств защиты данных», пункт «Управление уровнем доступа к таблице».

Пример

```
create or replace table tb (i int default 0, c char(20))
 level("СЕКРЕТНО", "НЕСЕКРЕТНО");
```

Определение имени столбца

Спецификация

См. спецификацию пункта «Создание таблицы».

Синтаксические правила

1) <Имя столбца> должно отличаться от любого другого <имени столбца> в данной спецификации столбцов таблицы.

```
create or replace table tst(col1 int, col2 char(5), col3 boolean);
```

Определение типа данных столбца

Спецификация

См. спецификацию пункта «Создание таблицы».

Синтаксические правила

1) Характеристика <типов данных столбца> приведена в разделе «Типы данных».
2) Параметр <длина> типов данных CHAR, VARCHAR, BYTE, VARBYTE – положительное целое, не превосходящее 4000 символов. По умолчанию для типов данных CHAR и BYTE значение <длина> считается равным 1.
3) Параметр <длина> типов данных NCHAR, NCHAR VARYING – положительное целое, не превосходящее 2000 символов. По умолчанию для типа данных NCHAR <длина> считается равной 1.
4) По умолчанию точность типа данных REAL – 6 десятичных цифр.
5) По умолчанию точность типа данных DOUBLE – 15 десятичных цифр.

6) Тип данных FLOAT [(<точность>)] определяет приближенный тип данных (синоним типов данных REAL и DOUBLE). <Точность> – целое положительное число в диапазоне от 1 до 53, задающее двоичную точность приближенного значения. Если <точность> <= 24, это воспринимается как тип данных REAL, при значении <точности> в диапазоне от 25 до 53 – как DOUBLE. Если <точность> не задана, по умолчанию принимается DOUBLE.

```
create or replace table test
(col1 real,
 col2 DOUBLE PRECISION,
 col3 DOUBLE,
 col4 float(10),
 col5 FLOAT);
```

7) Тип данных DECIMAL (DEC, NUMERIC) задает знаковое или беззнаковое десятичное число с общим количеством цифр не более 30 (включая лидирующие и завершающие нули, атрибут <точность>) и максимум 10 цифр справа от десятичной точки (включая младшие нули, атрибут <мантисса>).

8) Максимальное количество столбцов в одной таблице не должно превышать 250.

9) В качестве <кодировки> для BLOB-столбцов можно использовать имена однобайтовых (CP866, CP1251, KOI8-R), многобайтовых (UTF-8, EUC-JP, CP932) и UNICODE (UCS2) кодировок.

10) Опция CHARACTER SET <кодировка> применительно к столбцу с символьным типом данных задает кодировку конкретно этого столбца. Данная кодировка является приоритетной по сравнению с кодировкой, заданной для всей таблицы.

```
create or replace table test character set "KOI8-R"
(col1 char(10) character set cp1251,
 col2 nchar(50),
 col3 varchar(200) character set "CP866",
 col4 char(66));
```

11) При создании системных таблиц для символьных данных всегда используется кодировка системного словаря (DEFAULT) или заданная командой SET DATABASE NAMES.

12) Для обычных базовых таблиц, если <кодировка> стоблца типа CHAR/VARCAHAR/BLOB не задана:

- используется кодировка, установленная для таблицы;
- если кодировка для таблицы не задана, то используется кодировка, заданная командой SET NAMES;
- если не задана ни одна из кодировок предыдущих пунктов, то используется кодировка DEFAULT с идентификатором #0 (кодировка поддерживает первые 128 символов кодировки CP437).

13) Опция ROOT <каталог> задает местоположение на диске внешних файлов столбца данных типа EXTFILE (по умолчанию, если опция не задана, используется каталог БД).

```
create or replace table test (col1 extfile, col2 extfile root 'c:
\Program Files\Document');

create or replace table test (col1 extfile, col2 ef root 'c:
\linter\db');
```

Определение подставляемого значения столбца

Спецификация

См. спецификацию пункта «Создание таблицы».

Общие правила

1) Пропущенное в команде добавления данных значение столбца может быть вычислено и подставлено по заданному правилу. Если в команде добавления данных не задано значение столбца или не указано <подставляемое значение столбца>, то по умолчанию используется NULL-значение (если оно допустимо для столбца).

Определение автоматически наращиваемого значения столбца

Спецификация

См. спецификацию пункта «Создание таблицы».

Синтаксические правила

1) Атрибут AUTOROWID применим к типам данных SMALLINT, INT, BIGINT и только к одному столбцу таблицы.
2) В таблице разрешен только один столбец с атрибутом AUTOROWID. Он не может иметь модификатор AUTOINC.
3) Опция AUTOINC INITIAL применима к типам данных SMALLINT, INT, BIGINT.
4) Атрибуты AUTOINC и PRIMARY KEY совместимы.

   Эти конструкции равнозначны:

```
create or replace table tt (i int primary key autoinc);
create or replace table tt (i int autoinc primary key);
```

5) Между ключевыми словами AUTOINC и INITIAL можно вставлять допустимое по смыслу ограничение столбца, например:

```
create table test (i int autoinc not null initial(0));
create table test (i int autoinc primary key initial(0));
create table test (i int autoinc unique initial(0));
```

6) <Целочисленный литерал> в <начальном значении> может быть отрицательной величиной (за исключением максимального отрицательного значения для соответствующего целочисленного типа данных).

   Недопустимые конструкции:

```
create or replace table test (i smallint autoinc initial
 (-32768));
create or replace table test (i int autoinc initial
 (-2147483648));
```

Общие правила

1) Атрибут AUTOROWID определяет <автоматическое наращивание> значения ROWID добавляемой записи.

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

2) <Автоматическое наращивание> генерирует уникальное целочисленное значение столбца добавляемых в таблицу записей в диапазоне от 1 до MAXROW.

```
create or replace table test (col1 int autorowid);
insert into test;
…
select * from test;

|1|
|2|
|3|
…
```

3) Для столбца с атрибутом AUTOROWID явно задаваемое значение столбца должно быть в пределах от 1 до MAXROWID таблицы и отличным от уже существующих ROWID (т.е. дубликаты значений в столбце с атрибутом AUTOROWID недопустимы).
4) Значения AUTOROWID удаленных записей повторно не используются. Для их повторного использования необходимо выполнить сжатие таблицы (операция PRESS), при этом освобождаются все значения AUTOROWID.
5) Для столбца с атрибутом AUTOROWID все значения должны совпадать с ROWID соответствующих записей таблицы.
6) Опция AUTOINC INITIAL определяет <автоматическое наращивание> как автоматически наращиваемое значение, начинающееся со стартового <начального значения>.
7) <Начальное значение> может быть задано в диапазоне от -(N-1) до +N, где N – максимальное значение типа данных соответствующего столбца.
8) Если <начальное значение> не задано, по умолчанию используется 1.
9) Шаг изменения значений AUTOINC равен 1.

```
create or replace table test
(col1 int autoinc, col2 int autoinc initial (-5));
insert into test;
insert into test;
insert into test;
select * from test;

|1|-5|
|2|-4|
|3|-3|
```

10) Значения AUTOINC удаленных записей повторно не используются. Для их повторного использования необходимо выполнить сжатие таблицы (операция PRESS), при этом освобождаются все значения AUTOINC.
11) Операция UPDATE для столбца с атрибутом AUTOINC INITIAL не допустима.
12) В операции INSERT столбцу с атрибутом AUTOINC INITIAL явно можно присваивать только такое значение, которое превышают текущее максимальное значение столбца. После этого автоматическое формирование значений столбца будет выполняться от текущего значения столбца.

```
create or replace table test (col1 int autoinc, col2 int autoinc
 initial (-5));
insert into test;
```

```
insert into test;
insert into test;
select * from test;

|1|-5|
|2|-4|
|3|-1|

insert into test (col2) values(100);
insert into test;
select * from test;

|1|-5|
|2|-4|
|3|-1|
|4|100|
|5|101|
```

13) Стартовым значением автоинкрементного столбца в таблице будет числовой <литерал> из <опции умолчания> автоинкрементного поля (если опция задана) или значение автоинкрементного поля в самой первой добавленной в таблицу записи (по умолчанию 1). Допускается создавать до четырех столбцов с атрибутом AUTOINC.

```
create table tab1 (i int autoinc default 10, si smallint autoinc,
 bi bigint autoinc default -1000);
insert into tab1 default values;
insert into tab1 default values;
insert into tab1 default values;
insert into tab1 default values;
select * from tab1;

| 10 | 1 | -1000 |
| 11 | 2 | -999  |
| 12 | 3 | -998  |
| 13 | 4 | -997  |
```

14) При попытке вставить NULL-значение в столбец с атрибутом AUTOINC ошибка не фиксируется, а вместо NULL-значения вставляется следующее по порядку автоинкрементное значение.

```
create or replace table test( i int autoinc, ch char (10) );
insert into test values(NULL, 'aaa');
insert into test(i) values(NULL);
insert into test(ch) values('bbb');
insert into test(i) values(NULL);
insert into test;
select * from test;

|   1| aaa |
```

```
|    2|      |
|    3|  bbb |
|    4|      |
|    5|      |
```

15) Атрибут AUTOINC RANGE задает диапазоны допустимых значений столбца.
16) Атрибут AUTOINC RANGE применим к типам данных SMALLINT, INT, BIGINT и к любому допустимому количеству столбцов таблицы.
17) Атрибут AUTOINC RANGE определяет <автоматическое наращивание> как автоматически наращиваемое значение из заданных <диапазонов> значений.
18) В атрибуте AUTOINC RANGE <нижняя граница> <диапазона> во всех случаях должна быть меньше <верхней границы> того же <диапазона>, <нижняя граница> следующего <диапазона> больше <верхней границы> предыдущего <диапазона> (т.е. <диапазоны> должны быть упорядочены по возрастанию значений).

```
create table TEST_RANGE (i int autoinc range (1:1000,
 5000:10000,70000:100000));
```

19) Вновь добавляемый <диапазон> не проверяется на отсутствие значений, это должен делать администратор БД, создающий его. При занесении значения по умолчанию в столбец AUTOINC RANGE оно берется наименьшим из тех, которые больше последнего занесенного значения AUTOINC и при этом попадают в один из указанных диапазонов.
20) В атрибуте AUTOINC RANGE количество значений столбца из одного диапазона или суммарное количество значений всех диапазонов столбца не должно превышать значение MAXROW для таблицы.
21) Шаг изменения значений AUTOINC RANGE равен 1.
22) Операция UPDATE для столбца с атрибутом AUTOINC RANGE не допустима.
23) В операции INSERT столбцу с атрибутом AUTOINC RANGE явно можно присваивать только такое значение, которое входит в один из перечисленных <диапазонов>.
24) Если таблица имеет несколько столбцов с атрибутом AUTOINC RANGE, то максимально возможное количество добавляемых записей будет равно самому минимальному из всех декларированных диапазонов.

```
В такую таблицу можно добавить всего 2 записи (фиксируется код
завершения «Переполнение инкрементного столбца»)
create or replace table test (col1 int autoinc range (-100:-10,
 0:500), col2 int autoinc range (1:2));
insert into test;
insert into test;
insert into test;/* ошибка */
select * from test;

|1|-5|
|2|-4|
```

25) Если для столбца задано несколько <диапазонов>, то при добавлении записей сначала используются номера из первого <диапазона>, потом из второго и так далее.
26) В операции INSERT столбцу с атрибутом AUTOINC RANGE явно можно присваивать только такое значение, которое входит в один из перечисленных

<диапазонов>. После этого автоматическое формирование значений столбца будет выполняться от текущего значения этого <диапазона>.

```
create or replace table test (col1 int autoinc range (-100:-10,
 0:500), col2 int autoinc range (1:5));
insert into test;
insert into test;
insert into test(col1) values (333);
insert into test;
select * from test;

|-100|1|
|-99 |2|
|333 |3|
|334 |4|
```

27) Если задана конструкция RANGE, то нельзя задавать опцию INITIAL, и наоборот.

28) При занесении явно указанного значения в столбец AUTOINC со спецификацией RANGE оно должно быть больше последнего занесенного значения AUTOINC лишь в том случае, если попадает в один из указанных диапазонов.

29) В случае AUTOINC без RANGE добавляемое значение может быть любым. Проверка, что вновь добавляемого значения нет в столбце, отсутствует.

30) Запрещен выход значения по умолчанию для AUTOINC с диапазонами за границу последнего диапазона +1.

31) Формирование значения AUTOINC для типа BIGINT отличается от INT AUTOINC и SMALLINT AUTOINC: после удаления всех записей и сжатия таблицы для INT и SMALLINT значение начинает наращиваться от заданного как INITIAL при создании таблицы, а для BIGINT – от 1.

32) Столбцам точных числовых типов разрешается присваивать значения приближенных числовых типов. Приведение приближенного значения к точному значению выполняется путем отсечения дробной части.

```
create or replace table tst(s smallint, i int, b bigint);
insert into tst (s, i, b) values(1.05, 2.45, 3.0009);
update tst set b=678.897;

select * from tst;

|1|2|678|
```

Определение значения по умолчанию столбцов таблицы

Спецификация

См. спецификацию пункта «Создание таблицы».

Синтаксические правила

1) Суммарный размер <значений по умолчанию> столбцов таблицы, задаваемых с помощью конструкций <задаваемое значение>, <вычисляемое значимое> не должен превышать 4000 символов.

2) Если для столбца задано <задаваемое значение>, то оно автоматически приводится к конструкции GENERATED BY DEFAULT AS.

Эти конструкции эквивалентны:

```
create or replace table q1(j int default 1, i int default (j+5));
create or replace table q1(j int default 1, i int generated by
 default as (j+5));
```

3) Разрешено использование функции TO_DATE с константными аргументами в качестве <значения по умолчанию> для столбца типа DATE.

```
create table tab1 (d1 date default
 to_date('01.01.2003'),'dd.mm.yyyy');
```

4) Запрещено одновременное задание для столбца опций DEFAULT и PRIMARY KEY.

5) Значение DEFAULT NULL устанавливается лишь в том случае, если оно не конфликтует ни с какими ранее установленными атрибутами столбца.

6) Значение DEFAULT NULL допускается для столбца с атрибутом NOT NULL, но не устанавливает NULL-значение по умолчанию. Т.е. попытка выполнить ALTER TABLE DROP DEFAULT для такого столбца будет неудачной, т.к. NULL-значение является стандартным значением (а не значением по умолчанию) для тех полей таблицы, которым не присвоено реальное значение при выполнении операции INSERT.

Общие правила

1) Конструкция <значение по умолчанию> задает значение, которое будет присваиваться столбцу при добавлении записи, в которой значение данного столбца не указано. Тип данных <значения по умолчанию> должен соответствовать типу данных столбца или приводиться к нему.

```
create or replace table test
(col1 char(10) character set CP866 default 'Воронеж',
 col2 byte(5) default 0xFF00,
 col3 boolean default 'true',
 col4 real default (+00002.E-5),
 col5 numeric default -28.755,
 col6 date default ('07.11.1917'),
 col7 nchar(12) default n'ъъъ');
```

2) Для BLOB и EXTFILE типов данных <значение по умолчанию> не поддерживается.

3) При отсутствии явного значения столбца или <значения по умолчанию> столбцу присваивается NULL-значение (если оно допустимо) за исключением BLOB-столбца, которому присваивается нулевая длина BLOB-данных.

```
create or replace table test (col1 char(5), col2 byte(10),
  col3 boolean,col4 extfile, col5 real, col6 numeric,
  col7 date, col8 nchar, col9 blob, col10 extfile);
insert into test;
select * from test;

|NULL| NULL| … |Тип:0, Размер:0|NULL|
```

4) <Литерал> в <значении по умолчанию> может иметь максимальное отрицательное значение для соответствующего целочисленного типа данных, уменьшенное на 1.

```
create or replace table test (i smallint autoinc default -32767);
create or replace table test (i int autoinc default -2147483647);
```

5) Максимальное отрицательное значение <литерала> в конструкции AUTOINC INITIAL на единицу меньше максимального отрицательного значения для соответствующего целочисленного типа данных.

```
create or replace table tst (i smallint autoinc initial ( 32767));

create or replace table tst (i smallint autoinc initial (-
32768)); /* ошибка */
```

6) <Значимое выражение> может содержать ссылки на столбцы добавляемой записи.

```
create or replace table test
(col1 int,
 col2 int default (col1+5),
 col3 date default (sysdate),
 col4 int default (abs(-45)+sqrt(16)),
 col5 char(10),
 col6 char(10) default (to_char(length(col5))),
 col7 char(10) default '12345',
 col8 char(10) default (to_char(length(col7))));

insert into test(col1,col5) values (200, '12345');
select * from test;

|200|205| 16.11.2017:14:05:25| 49| 12345|5|
```

Определение значения из последовательности

Спецификация

См. спецификацию пункта «Создание таблицы».

Синтаксические правила

1) Опции <последовательности> аналогичны опциям создания последовательности (см. пункт «Создание последовательности»).

```
create or replace table parts(
part_no integer generated as identity (start with 1 increment by
 -2 minvalue -100000 maxvalue 100000) unique,
part_descr varchar (20),
part_quantity integer);

create or replace table tst (
name char(10),
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
"№ п/п" int GENERATED as identity (start with 1 increment by 1
 maxvalue 1000));
```

2) Если опции ALWAYS, BY DEFAULT не заданы, по умолчанию используется BY DEFAULT.
3) Задание типа данных столбца со <значениями из последовательности> является обязательным.
4) <Значение из последовательности> разрешено задавать только для типов данных SMALLINT, INTEGER, BIGINT.
5) При создании столбца со <значениями из последовательности> неявным образом задается NOT NULL.
6) В таблице можно определить не более четырёх столбцов со <значениями из последовательности>.
7) Для столбца со <значениями из последовательности> нельзя задавать ограничения AUTOINC, AUTOROWID, AUTOINC RANGE, а также значения DEFAULT, в противном случае будет получен код завершения – 1120 («Неверное описание атрибута IDENTITY»).
8) Значения по умолчанию атрибута <последовательность> опции GENERATED [ALWAYS|BY DEFAULT] AS IDENTITY:
   - <начальное значение>: 1;
   - <шаг>: 1;
   - <верхняя граница>: MAXROW;
   - <нижняя граница>: 1;
   - NO CYCLE.

```
create or replace table
  test (col1 int generated always as identity ());
insert into test;
insert into test;
select * from test;

|1|
|2|
```

9) В <значении из последовательности> <нижняя граница> должна быть строго меньше <верхней границы>.
10) Если в <значении из последовательности> задано <начальное значение>, то оно должно быть не больше MAXROW и быть в диапазоне MINVALUE: MAXVALUE (если заданы эти атрибуты), либо в диапазоне допустимых значений соответствующего типа данных столбца.

```
create or replace table test (col1 smallint generated always as
 identity (START WITH 128 NOMAXVALUE));

Ошибочная конструкция
create or replace table test (col1 smallint generated always as
 identity (START WITH 65538 NOMAXVALUE));
```

11) Если в <значении из последовательности> задано CYCLE, то столбец не должен иметь атрибут PRIMARY KEY или UNQUE.

Общие правила

1) Если при определении столбца со <значениями из последовательности> указана опция ALWAYS, то в операциях добавления и модификации записей всегда будет использоваться извлекаемое из <последовательности> значение. Попытка присвоить столбцу собственное значение вызовет ошибку.

```
create or replace table tst (name char(10), "№ п/п" int GENERATED
 always as identity (start with 1 increment by 1 maxvalue 1000));

insert into tst ("№ п/п", name) values (default, '11111');
insert into tst (name) values ('22222');
insert into tst (name) values ('33333');
insert into tst ("№ п/п", name) values (default, '44444');
insert into tst ("№ п/п", name)
  values (100, '55555'); /* ошибка */

select * from tst;

|11111     |         1|
|22222     |         2|
|33333     |         3|
|44444     |         4|
```

2) Для включения столбца со <значениями из последовательности> в список добавляемых (модифицируемых) столбцов без фиксации указанной выше ошибки необходимо использовать опцию DEFAULT.

```
create or replace table tst (name char(10), "№ п/п" int GENERATED
 by default as identity (start with 1 increment by 1 maxvalue
 1000));

insert into tst ("№ п/п", name) values (100, '11111');
insert into tst (name) values ('22222');
insert into tst (name) values ('33333');
insert into tst ("№ п/п", name) values (default, '44444');
insert into tst ("№ п/п", name) values (200, '5555');

select * from tst;

|11111     |       100|
|22222     |         1|
|33333     |         2|
|44444     |         3|
|55555     |       200|
```

3) Если при определении столбца со <значениями из последовательности> задана опция BY DEFAULT, то в операциях добавления и модификации записей выбираемое из <последовательности> значение будет использоваться только в тех случаях, когда не задано явное значение.

Примеры.

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
а)
create or replace table tst (name char(10), "№ п/п" int GENERATED
 by default as identity (start with 1 increment by 1 maxvalue
 1000));

insert into tst ("№ п/п", name) values (100, '11111');
insert into tst (name) values ('22222');
insert into tst (name) values ('33333');
insert into tst ("№ п/п", name) values (400, '44444');

select * from tst;

|11111     |        100|
|22222     |          1|
|33333     |          2|
|44444     |        400|

б)
create or replace table test (col1 int generated by default as
 identity (start with 15 maxvalue 100 minvalue -100));
insert into test;
insert into test;

select * from test

|15|
|16|
```

4) Если ни одна из опций `ALWAYS` и `BY DEFAULT` не задана, по умолчанию используется `ALWAYS`.
5) Конструкция `GENERATED ALWAYS AS IDENTITY (<последовательность>)` подставляет всегда текущее значение `<последовательности>` в столбец добавляемой (модифицируемой) записи, поэтому в операторе добавления/модификации данных нельзя указывать значение добавляемого/модифицируемого столбца; добавляемое/модифицируемое значение должно отсутствовать или заменено фразой `default`, иначе будет фиксироваться ошибка.

```
create or replace table test (col1 int generated always
  as identity (start with 35 maxvalue 100 minvalue -100));
insert into test;
insert into test;
insert into test(col1) values (default);

select * from test;

|35|
|36|
|37|
```

```
! Ошибочная конструкция
insert into test(col1) values (66);
```

6) Если в опции GENERATED [ALWAYS|BY DEFAULT] AS IDENTITY задан атрибут CYCLE, то после исчерпания значений <последовательности> значения столбцов добавляемых (модифицируемых) записей будут повторяться в соответствии с первоначальными параметрами <последовательности>.

```
create or replace table test
(col1 int generated always as identity
(start with 1 increment by  3 maxvalue 5 minvalue 1 cycle));

insert into test default values;
insert into test default values;
insert into test default values;
insert into test default values;

select * from test;

|1|
|4|
|1|
|4|
```

Определение генерируемого значения

Спецификация

См. спецификацию пункта «Создание таблицы».

Синтаксические правила

1) Конструкция <генерируемое значение> применима к типам данных SMALLINT, INT, BIGINT и к любому допустимому количеству столбцов таблицы.

2) Если тип данных <генерируемого значения> столбца указан явно, то он должен быть совместим с типом данных задаваемого <логического выражения> или <значимого выражения>. При отсутствии явного указания типа данных <генерируемого значения> столбца по умолчанию ему назначается тип данных <логического выражения> или <значимого выражения>.

```
create or replace table emp (
emp_no integer,
emp_sal double,
emp_bonus double,
emp_total generated always as (emp_sal + emp_bonus));
```

3) Функция SYS_GUID() в качестве <генерируемого значения> применима только для столбцов с типом данных BYTE(n)/VARBYTE(n), где n должно быть не меньше 16.

```
create or replace table tst(j int default 1, i byte(20) generated
 by default as (sys_guid()));
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

Общие правила

1) При вставке записи в таблицу, содержащую столбец с <генерируемым значением>, вычисляется ассоциированное с ним <логическое выражение> или <значимое выражение>, и полученное значение становится значением этого столбца во вставляемой записи.
2) Конструкция <генерируемое значение> с атрибутом BY DEFAULT AS <значимое выражение> генерирует значение по умолчанию (например, зависящее от текущего значения некоторого столбца). <Значимое выражение> может быть NULL-значением.

```
create or replace table "Товары"
("Дата" date,
 "Курс$" decimal,
 "Товар" char(10),
 "Коэффициент" int,
 "Текущая цена" decimal generated by default as ("Коэффициент" *
 "Курс$"));

insert into "Товары"("Дата","Курс$", "Товар", "Коэффициент")
values (to_date('01-aug-10'),30.2, 'Компьютер',1000);

insert into "Товары"("Дата","Курс$", "Товар", "Коэффициент")
values (to_date('03-aug-10'),27.8, 'Телевизор',500);

select to_char("Дата",'dd.mm.yyyy') "Дата", "Товар", "Текущая
 цена"
  from "Товары";

 Дата          Товар        Текущая цена
|01.08.2010   |Компьютер   |30200.0      |
|03.08.2010   |Телевизор   |13900.0      |
```

3) Конструкция <генерируемое значение> с атрибутом BY DEFAULT подставляет вычисляемое <логическое выражение> или <значимое выражение> только в случае отсутствующего значения столбца в добавляемой (модифицируемой) записи. <Значимое выражение> может быть NULL-значением.

```
create or replace table "Товары"
("Дата" date,
"Курс$" decimal,
"Товар" char(10),
"Коэффициент" int,
"Текущая цена" decimal generated by default as ("Коэффициент" *
 "Курс$"));

insert into "Товары"("Дата", "Курс$", "Товар", "Коэффициент")
values (to_date('01-aug-10'), 30.2, 'Компьютер',1000);

insert into "Товары"("Дата", "Курс$", "Товар", "Коэффициент")
```

```
values (to_date('03-aug-10'), 27.8, 'Телевизор',500);

select to_char("Дата",'dd.mm.yyyy') "Дата", "Товар", "Текущая
 цена"
  from "Товары";

 Дата          Товар        Текущая цена
|01.08.2010   |Компьютер   |30200.0     |
|03.08.2010   |Телевизор   |13900.0     |
```

4) Тип данных <логического выражения> или <значимого выражение> в конструкции <генерируемое значение> должен соответствовать типу данных столбца или быть приводимым к нему.

5) Имена столбцов, задействованных в <логическом выражении> или в <значимом выражении>, должны быть определены до использования их в <генерируемом значении>.

```
Вычисление вклада с учетом процента на заданный месяц
create or replace table test
(col1 real,
col2 decimal default 7.5,
col3 date,
col4 decimal generated by default
as (col1+col1*col2/100/12*datesplit(col3,'M')));

insert into test (col1, col3)
  values (45000, to_date('21.06.2017','dd.mm.yyyy'));

insert into test (col1, col3)
  values (100000.6, to_date('05.07.2017','dd.mm.yyyy'));
select * from test;

|col1    |col2|col3               |col4      |
|45000   | 7.5|21.06.2017:00:00:00|46687.5   |
|100000.6| 7.5|05.07.2017:00:00:00|104375.625|

create or replace table test (col1 char(20), col2 boolean
 generated by default as (col1 is null));
insert into test(col1)  values ('12345');
insert into test(col1)  values (null);

select * from test

|col1    |col2 |
|12345   |false|
|null    |true |
```

6) Если для <генерируемого значения> используется <логическое выражение>, то объявление типа данных <генерируемого значения> является обязательным, например:

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
create or replace table test (b generated always as (2>1));
```

– будет выдан код завершения 2705;

```
create or replace table test (b BOOLEAN generated always as
 (2>1));
```

– завершится без ошибки.

7) Для <генерируемого значения> допустимые операции внутри <логического выражения> те же, что и в <логическом выражении> конструкции CHECK.
8) <Логическое выражение> для <генерируемого значения> должно строиться из констант и ссылок на основные столбцы (без <генерируемых значений>) определяемой таблицы.
9) Особенности добавления и модификации <генерируемых значений> приведены в разделах «Добавление записи» и «Корректировка записи».

Определение фильтра для фразового поиска

Спецификация

См. спецификацию пункта «Создание таблицы».

Синтаксические правила

1) <Фильтр для фразового поиска> можно задавать только для столбцов типа CHAR, VARCHAR, BLOB или EXTFILE.

```
create table test_blob (
  id integer,
  name char(18),
  document blob default filter ascxml2text);
```

2) <Имя фильтра> должно быть прописано в системной таблице $$$FILTER.

Общие правила

1) Конструкция <фильтр для фразового поиска> задает имя фильтра, используемого для столбца при создании фразового индекса (см. документ «Полнотекстовый поиск в базе данных», пункт «Создание фразового индекса»).

Пример

```
create or replace table tst_filter
(col1 char(2000) default filter DOCRTF2TEXT,
 col2 nchar(1000) default filter UTF82TEXT);
```

Определение мандатного уровня доступа к столбцу

Спецификация

См. спецификацию пункта «Создание таблицы».

Синтаксические правила

1) См. документ «Администрирование комплекса средств защиты данных», пункт «Управление уровнем доступа к столбцу таблицы».

Общие правила

1) См. документ «Администрирование комплекса средств защиты данных», пункт «Управление уровнем доступа к столбцу таблицы».

Пример

```
create or replace table tb(i int level("СЕКРЕТНО", "НЕСЕКРЕТНО"),
 c char(20) level("ДСП", "НЕСЕКРЕТНО"));
```

Определение ограничения NULL-значений

Спецификация

См. спецификацию пункта «Создание таблицы».

Синтаксические правила

1) Если <ограничение NULL-значений> явно не указано, по умолчанию используется NULL.
2) Если задана конструкция <имя столбца> NULL, то для того же столбца нельзя задать NOT NULL или PRIMARY KEY.
3) Если NOT NULL не указан и не задано <значение по умолчанию>, то подразумевается DEFAULT NULL.
4) Разрешено сочетание описателей NOT NULL и DEFAULT NULL для столбца. В этом случае используется NOT NULL, а DEFAULT NULL игнорируется.

```
create table tab1 (i int unique default null not null);
```

Общие правила

1) Атрибут столбца NULL разрешает, NOT NULL запрещает присваивать значению столбца NULL-значения. По умолчанию используется NULL.

```
create or replace table test (col1 int not null);
insert into test(col1) values (100);
insert into test(col1) values (null); /*ошибка */
insert into test; /* ошибка */
```

2) Атрибут NULL устанавливает значение по умолчанию для столбцов, значение которых не определено в операции добавления (модификации) записи и для которых не задана опция <подставляемое значение>.

```
create or replace table test (col1 int null);
insert into test(col1) values (100);
insert into test(col1) values (null);
insert into test;

select NVL(col1,-1) from test;

|        100|
|         -1|
|         -1|
```

3) Если для столбца заданы одновременно <ограничение NULL-значений> и <значение по умолчанию>, то приоритет имеет опция <ограничение NULL-значений>.

© Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

Например, в случаях

```
create or replace table test (col1 int not null default null );

create or replace table test (col1 int default null not null);
```

будет применяться правило not null.

4) <Подставляемое значение столбца> не должно конфликтовать с <ограничением NULL-значений>.

```
create or replace table test (col1 int null,
  col2 int generated always as (col1+500) not null);

insert into test(col1) values (100);
insert into test(col1) values (null); // ошибка
insert into test; // ошибка
select * from test;
```

Определение ссылочной целостности столбца

Спецификация

См. спецификацию пункта «Создание таблицы».

Общие правила

1) Конструкция <ссылочная целостность столбца> определяет, что добавляемое (обновляемое) значение данного столбца должно совпадать с одним из значений указанного в REFERENCES столбца или иметь NULL-значение.
2) Для исключения неоднозначности указанный в REFERENCES столбец должен быть столбцом типа PRIMARY KEY.

```
create or replace table test1 (col1 int autorowid primary key);

create or replace table test2 (col1 int references test1(col1));

insert into test1;
insert into test1;
insert into test1;
select * from test1;
1
2
3

insert into test2(col1)  values (2)
insert into test2(col1)  values (2)
select * from test2;
2
2
```

```
insert into test2(col1)  values (5);
//ошибка – не существует значение первичного ключа
```

3) Допускается ссылаться на столбцы в той же самой таблице.

```
create or replace table test
(col1 int primary key,
 col2 int references test(col1),
 col3 char(10) unique,
 col4 char(20) references test(col3));
```

4) Если <имя столбца> не указано, предполагается, что ссылка делается на столбец с атрибутом PRIMARY KEY в <спецификации таблицы>.

```
create or replace table t_pk (c_pk varchar(5) primary key);
create or replace table t_fk (c_fk varchar(5) references t_pk);
```

5) Разрешается не указывать столбец в конструкции REFERENCES, если ссылка производится на первичный ключ таблицы.

```
create or replace table tab1 (i int, c1 char(10), c2 varchar(15),
 primary key (i,c2));
create or replace table tab2 (i int, c2 varchar(20), foreign
 key(i,c2) references tab1);
```

6) Числовые типы данных столбцов, задействованных в <ссылочной целостности столбца>, должны быть идентичными.

7) Длина символьных или байтовых данных определяемого столбца должна быть не больше длины данных столбца, на который делается ссылка.

```
Допустимые команды
create or replace table t_pk (c_pk varchar(5) primary key);
create or replace table t_fk (c_fk varchar(10) references t_pk);

Недопустимые команды
create or replace table t_pk (c_pk varchar(200) primary key);
create or replace table t_fk (c_fk varchar(10) references t_pk);
```

8) <Каскадные ограничения> определяют действия, которые должны выполняться при попытке удалить или обновить значение столбца, на которое есть ссылка:

- ON DELETE SET NULL: при удалении записи, содержащей первичный ключ, на который есть внешние ссылки, значение внешних ключей в подчиненных таблицах заменяется на NULL-значение. Чтобы выполнялось это ограничение, столбец внешнего ключа должен допускать NULL-значение.

```
create or replace table t_pk (c_pk int primary key);
create or replace table t_fk (c_fk int references t_pk(c_pk) on
 delete set null);
insert into t_pk values(10), (20), (30);
select * from t_pk;
10
20
30

insert into t_fk values(20), (30);
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
select * from t_fk;
20
30

delete from t_pk where c_pk=20;
null
30
```

- Опция ON UPDATE указывает, какое значение должно быть присвоено столбцу данных типа DATE при операции обновления записи в случае отсутствия корректируемого значения и спецификации <подставляемое значение>.

Примеры.

а)

```
create or replace table test (col1 int, col2 date on update
 CURRENT_TIMESTAMP);
insert into test(col1) values (1);
insert into test(col1, col2) values(2, to_date('11.11.1111',
 'dd.mm.yyyy'));
select * from test;

|1  |21.11.2017:15:15:50|
|2  |11.11.1111:15:15:50|

update test set col1=200 where col1=2;
select * from test;

|1   |21.11.2017:15:15:50|
|200 |21.11.2017:15:15:50|
```

б)

```
create or replace table test (i int, date1 date, date2 date on
 update sysdate);
insert into test (i, date1, date2) values (1,
 to_date('01.01.2007','dd.mm.yyyy'), to_date('28.04.1950',
 'dd.mm.yyy'));
select * from test;

| 1|01.01.2007:00:00:00.00|28.04.1950:00:00:00.00|

select sysdate;

|27.08.2007:15:58:31.00|

update test set i = 100, date1 = to_date('11.11.1111',
 'dd.mm.yyyy');
select * from test;
```

```
| 100|11.11.1111:00:00:00.00|27.08.2007:15:58:31.00|
```

- ON UPDATE SET NULL: указывает, что при попытке обновить ключевое значение, на которое ссылаются внешние ключи в строках других таблиц, все значения, составляющие эти внешние ключи, должны быть изменены на NULL. Чтобы выполнялось это ограничение, все столбцы внешних ключей целевой таблицы должны допускать значение NULL.

```
create or replace table t_pk (c_pk int primary key);
create or replace table t_fk (c_fk int references t_pk(c_pk) on
 update set null);
insert into t_pk values(10), (20), (30);
select * from t_pk;
10
20
30

insert into t_fk values(20), (30);
select * from t_fk;
20
30

update t_pk set c_pk=500 where c_pk=20
select * from t_pk;
10
500
30

select * from t_fk;
null
30
```

- ON DELETE SET DEFAULT: Указывает, что при удалении значения, на которое ссылается внешний ключ, оно должно быть заменено на значение по умолчанию. Чтобы выполнялось это ограничение, столбец внешнего ключа должен допускать NULL-значение.

  Задаваемое значение по умолчанию должно присутствовать среди значений первичного ключа.

```
create or replace table t_pk (c_pk int primary key);
create or replace table t_fk (c_fk int default -1 references
 t_pk(c_pk) on delete set default);
insert into t_pk values(10), (20), (30),(-1);
insert into t_fk values(20), (30);
delete from t_pk where c_pk=20;
```

  Если столбец допускает NULL-значение и значение по умолчанию явно не задано, то NULL-значение становится неявным значением по умолчанию для данного столбца.

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
create or replace table t_pk (c_pk int primary key);
create or replace table t_fk (c_fk int references t_pk(c_pk) on
 delete set default);
insert into t_pk values(10), (20), (30);
insert into t_fk values(20), (30);
delete from t_pk where c_pk=20;
```

- ON UPDATE SET DEFAULT: указывает, что при изменении значения, на которое ссылается внешний ключ, оно должно быть заменено на значение по умолчанию. Чтобы выполнялось это ограничение, столбец внешнего ключа должен допускать NULL-значение.

  Задаваемое значение по умолчанию должно присутствовать среди значений первичного ключа.

```
create or replace table t_pk (c_pk int primary key);
create or replace table t_fk (c_fk int default -1 references
 t_pk(c_pk) on update set default);
insert into t_pk values(10), (20), (30), (-1);
select * from t_pk;

|10|
|20|
|30|
|-1|

insert into t_fk values(20), (30);
select * from t_fk;

|20|
|30|
C
```

- ON DELETE CASCADE: При удалении в родительской таблице записи с первичным ключом одновременно удаляется все записи в подчиненных таблицах, ссылающиеся на удаляемый первичный ключ.

```
create or replace table t_pk (c_pk int primary key);
create or replace table t_fk1 (c_fk int references t_pk(c_pk) on
 delete cascade);
create or replace table t_fk2 (c_fk int references t_pk(c_pk) on
 delete cascade);
insert into t_pk values(10), (20), (30);
select * from t_pk;

|10|
|20|
|30|
```

```
insert into t_fk1 values(20), (30);
select * from t_fk1;

|20|
|30|

insert into t_fk2 values(10),(20);
select * from t_fk2;

|10|
|20|

delete from t_pk where c_pk=20;
select * from t_pk;

|10|
|30|

select * from t_fk1;

|30|

select * from t_fk2;

|10|
```

- ON UPDATE CASCADE: при изменении в родительской таблице значения первичного ключа одновременно во всех подчиненных таблицах заменяется значение столбца, ссылающегося на изменяемый первичный ключ.

```
create or replace table t_pk (c_pk int primary key);
create or replace table t_fk1 (c_fk int references t_pk(c_pk) on
 update cascade);
create or replace table t_fk2 (c_fk int references t_pk(c_pk) on
 update cascade);
insert into t_pk values(10), (20), (30);
select * from t_pk;

|10|
|20|
|30|

insert into t_fk1 values(20), (30);
select * from t_fk1;

|20|
|30|
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
insert into t_fk2 values(10),(20);
select * from t_fk2;

|10|
|20|

update t_pk set c_pk=500 where c_pk=20;
select * from t_pk;

| 10|
|500|
| 30|

select * from t_fk1;

|500|
| 30|

select * from t_fk2;

| 10|
|500|
```

- ON DELETE NO ACTION|RESTRICT: прекратить операцию, если её выполнение приводит к нарушению ссылочной целостности (с выдачей соответствующего кода завершения). Ограничение RESTRICT является синонимом NO ACTION.

```
create or replace table t_pk (c_pk int primary key);
create or replace table t_fk (c_fk int references t_pk(c_pk) on
 delete no action);

insert into t_pk values(10), (20), (30);
select * from t_pk;

insert into t_fk values(20), (30);
select * from t_fk;
delete from t_pk where c_pk=20;
/* существует соответствующее значение внешнего ключа */
```

9) Разрешается удалять строки со спецификацией действия NO ACTION (RESTRICT) и ссылающиеся на самих себя.

10) Если <спецификация действий> не задана, по умолчанию предполагается NO ACTION.

11) Спецификация действий:

- CASCADE: строка с первичным (уникальным) ключом в родительской таблице удаляется/изменяется, и если в определении ограничения внешнего ключа отсутствует опция MATCH или присутствуют спецификации MATCH SIMPLE или MATCH FULL, то удаляются/изменяются все строки в подчиненной таблице,

ссылающиеся на удаляемую/изменяемую строку родительской таблицы. Если же в определении ограничения внешнего ключа присутствует спецификация MATCH PARTIAL, то удаляются/изменяются только те строки подчиненной таблицы, которые ссылаются исключительно на удаляемую/изменяемую строку родительской таблицы;

- SET NULL: строка с первичным (уникальным) ключом в родительской таблице удаляется/изменяется. Во всех столбцах, которые входят в состав внешнего ключа, и во всех строках подчиненной таблицы, ссылающихся на удаленную/измененную строку родительской таблицы, проставляется NULL-значение. Если в определении внешнего ключа содержится спецификация MATCH PARTIAL, то NULL-значение устанавливается только в тех строках подчиненной таблицы, которые ссылаются исключительно на удаляемую/изменяемую строку родительской таблицы;
- SET DEFAULT: строка с первичным (уникальным) ключом в родительской таблице удаляется/изменяется. Во всех столбцах, которые входят в состав внешнего ключа, и во всех строках подчиненной таблицы, ссылающихся на удаленную/измененную строку родительской таблицы, проставляется заданное при их определении значение по умолчанию. Если в определении внешнего ключа содержится спецификация MATCH PARTIAL, то значение по умолчанию устанавливается только в тех строках подчиненной таблицы, которые ссылаются исключительно на удаляемую/изменяемую строку родительской таблицы;
- NO ACTION: прекратить операцию (с выдачей соответствующего кода завершения), если её выполнение приводит к нарушению ссылочной целостности (ограничение RESTRICT является синонимом NO ACTION).

```
create or replace table test1(i1 int, j1 int, primary key(i1,j1));
insert into test1 values (1,1);
create or replace table test2(i2 int, j2 int, foreign key (i2,j2)
 references test1(i1,j1) match simple on delete cascade);
insert into test2 values(1,NULL);
insert into test2 values(NULL,4);
insert into test2 values(1,1);
insert into test2 values(NULL,NULL);
select * from test2;
```

```
 I2          J2
 --          --
|          1|           |
|           |          4|
|          1|          1|
|           |           |
```

```
delete from test1;
select * from test2;
```

```
 I2          J2
 --          --
|          1|           |
|           |          4|
```

```
|             |             |
```

12) Список столбцов спецификации PRIMARY KEY в <ограничении таблицы> определяет составной первичный ключ таблицы. Значения отдельных столбцов многостолбцового первичного ключа могут повторяться, однако в каждой строке таблицы конкатенация значений столбцов составного ключа должна быть уникальной.

```
create or replace table tab1 (vc varchar(30) not null, primary key
 (vc));
```

Определение контролируемого значения столбца

Спецификация

См. спецификацию пункта «Создание таблицы».

Синтаксические правила

1) Конструкция <контролируемое значение столбца> задает условие, которому должно удовлетворять значение столбца при операции добавления (модификации) записи. Если <контролируемое значение столбца> возвращает значение TRUE, то значение столбца считается корректным, в противном случае – недопустимым.
2) <Логическое выражение> не должно содержать ссылок на другие столбцы данной таблицы, кроме него самого.

```
Проверка кода города в телефонных номерах
create or replace table
  test (col1 char(15) check (substr(col1,1,3)='473'));
insert into test(col1) values ('473-2-711-711');
!недопустимая запись
insert into test(col1) values ('495-274-453-776');
insert into test(col1) values ('473-2-654-300');

select * from test

| 473-2-711-711|
| 473-2-654-300|
```

3) Использование столбцов с атрибутом AUTOROWID в <логическом выражении> запрещено.
4) Использование предиката [NOT] CONTAINS в <логическом выражении> конструкции запрещено (см. документ «Полнотекстовый поиск в базе данных», пункт «Предикат полнотекстового поиска»).
5) Разрешено использование значений SYSDATE, NOW, LOCALTIME, LOCALTIMESTAMP, CURRENT_DATE, CURRENT_TIME, CURRENT_TIMESTAMP в <контролируемом значении столбца>. Однако при этом в БД могут оказаться записи, которые удовлетворяли <контролируемому значению столбца> в момент выполнения операции INSERT/UPDATE, но со временем перестали удовлетворять этому ограничению, т.к. текущее значение этих значений стало другим.

```
select to_localtime(sysdate);
```

```
create or replace table test(i int,
  d date default localtimestamp on update current_time
    check (d < localtimestamp + to_date('01','SS')));

insert into test(i) values(1);
select d from test;
update test set i=2;
select d from test;

! запись не удовлетворяет условию CHECK
insert into test(i, d)
  values(2, localtimestamp + to_date('02','SS'));

! Результат работы примера:

|05.03.2012:16:25:48.47|

 D
 -
|05.03.2012:16:25:48.52|

 D
 -
|00.00.0000:16:25:48.52|

INL : состояние выполнения : 914
Запись не удовлетворяет условию CHECK
```

6) <Логическое выражение> не должно содержать подзапросов, агрегатных функций, конструкций CASE и CAST, функций SECURITY, INDEXTIME, LEAD, LAG, GETTEXTPOS.

Примеры.

```
а)
create or replace table test (col1 int check (col1 between 100 and
 200));

insert into test (col1)  values (100);
insert into test (col1)  values (250); /* ошибка  */

б)
create or replace table test (col1 int check (col1 not in
 (10,20,30)));

insert into test (col1)  values (200);
insert into test (col1)  values (20); /* ошибка  */

в)
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
create or replace table test (col1 char(20) check (col1 not like
 'B%'));

insert into test (col1)  values ('Воронеж'); /* ошибка  */
insert into test (col1)  values ('Москва');
insert into test (col1)  values ('Урюпинск');

г)
create or replace table test (col1 char(20) check (col1 similar to
 'B_W'));

insert into test (col1)  values ('BMW');
insert into test (col1)  values ('MERCEDES'); /* ошибка  */
insert into test (col1)  values ('LADA'); /* ошибка  */
insert into test (col1)  values ('B9W');
д)
create or replace table test (col1 real check (col1 is not null));

insert into test (col1)  values (56.008);
insert into test; /* ошибка  */
```

7) Каждый столбец может иметь произвольное число <контролируемых значений столбца>, но суммарная длина всех оттранслированных <контролируемых значений столбца> не должна превышать 4 Кбайт.

```
create or replace table tst_check(i int check (i>0) check (i<>1)
 check ((i<>18) and (i<>20)), c char(10) default 'Сумма');

insert into tst_check(i) values(2);
insert into tst_check(i) values(1); /* ошибка */
insert into tst_check(i) values(-5); /* ошибка */
insert into tst_check(i) values(18); /* ошибка */
insert into tst_check(i) values(10);
select c, i from tst_check;

|Сумма|    2|
|Сумма|   10|
```

8) Конструкция может задаваться в любом месте среди атрибутов столбца.

```
create or replace table tst_check(i int default 0 check (i>0)
 check (i<>1) check ((i<>18) and (i<>20)), c char(10) default
 'Сумма');

create or replace table tst_check(i int check (i>0) check (i<>1)
 default 0 check ((i<>18) and (i<>20)), c char(10) default
 'Сумма');

Несколько CHECK можно заменить одним
```

```
create or replace table tst_check(i int check ((i>0) and (i<>1)
 and ((i<>18) and (i<>20))), c char(10) default 'Сумма');

create or replace table tst_check(i int check (col_check=1));

create or replace procedure col_check (in i int) result int for
 debug
code
  if i > 1 then
    return 1;
  else
    return 0;
  endif
end;
call col_check(-4);

insert into tst_check(i) values(default);
insert into tst_check(i) values(2);
insert into tst_check(i) values(-3);
select i from tst_check;
delete from tst_check;
```

9) <Контролируемое значение столбца> не должно содержать ссылок на другие столбцы данной таблицы, кроме него самого.

10) <Логическое выражение> может содержать ссылки только на столбцы данной таблицы, которые уже определены в <спецификации столбцов таблицы>.

```
create table t (i int, j int, check (i>j));

Дата поставки товара должна быть не позже, чем через месяц после
 его изготовления
create or replace table pstv (crt date default sysdate,
 sale date, check(divtime(16,to_date(crt,'DD:MM:YYYY'),
 to_date(sale,'DD:MM:YYYY'))<30));

insert into pstv (sale) values
 (to_date('25:02:2018','DD:MM:YYYY'));
insert into pstv (sale) values
 (to_date('10:03:2018','DD:MM:YYYY')); /* ошибка */
select * from pstv;

|02.02.2018:08:36:17| 25.02.2018:00:00:00|

Следующие два оператора эквивалентны:
create table t (i int, j int, check ((i>j) and (i<j+10)));
create table t (i int, j int, check (i>j), check (i<j+10));
```

11) <Контролируемые значения записи> должно задаваться через запятую либо после определения всех столбцов таблицы, либо после определения задействованных в <логическом выражении> столбцов.

Варианты:

В конце списка определяемых столбцов

```
create or replace table tst_check (i int, c char(10), d decimal,
 check((i+d)<to_number(c)));
```

Внутри списка определяемых столбцов

```
create or replace table tst_check (i int, d decimal, check((i
+d)<100.0), c char(10));

create or replace table tst_check(i int default 0 check (i>0)
 check (i<>1) check ((i<>18) and (i<>20)), c char(10) default
 'Сумма');

create or replace table tst_check(i int check (i>0) check (i<>1)
 default 0 check ((i<>18) and (i<>20)), c char(10) default
 'Сумма');
```

12) Несколько <логических условий> можно заменять одним <логическим условием>.

```
create or replace table tst_check(i int check ((i>0) and (i<>1)
 and ((i<>18) and (i<>20))), c char(10) default 'Сумма');
```

Примеры

1)

```
create or replace table test (col1 int check ((col1 between 10 and
 30) and (col1<>15) and (col1>12)));

insert into test (col1) values(11); / ошибка
insert into test (col1) values(15); / ошибка
insert into test (col1) values(37); / ошибка
insert into test (col1) values(20);
```

2)

```
create or replace table test (col1 int check ((col1 between 10 and
 30) and (col1<>15) and (col1>12)),
col2 char(20) check (col2 not like 'A%') check (col2 not like 'Б
%') check (col2 not like 'B%'),
col3 real check (col3*0.5+45.90>0));

insert into test (col1, col2, col3) values (25, 'Москва', 56.89);
insert into test (col1, col2, col3) values (25, 'Воронеж',
 45.99); // ошибка
insert into test (col1, col2, col3) values (25, 'Урюпинск',
 -112.7); // ошибка
```

3) Следующие два оператора делают одно и то же:

```
create table t (i int check ((i>0) and (i<10)));
create table t (i int check (i>0) check (i<10));
```

4)

```
create or replace table test (col1 int check (col1 between 100 and
 200));

insert into test (col1)  values (100);
insert into test (col1)  values (250); /* ошибка */
```

5)

```
create or replace table test (col1 int check (col1 not in
 (10,20,30)));

insert into test (col1)  values (200);
insert into test (col1)  values (20); /* ошибка */
```

6)

```
create or replace table test (col1 char(20) check (col1 not like
 'B%'));

insert into test (col1)  values ('Воронеж'); /* ошибка */
insert into test (col1)  values ('Москва');
insert into test (col1)  values ('Урюпинск');
```

7)

```
create or replace table test (col1 char(20) check (col1 similar to
 'B_W'));

insert into test (col1)  values ('BMW');
insert into test (col1)  values ('MERCEDES'); /* ошибка */
insert into test (col1)  values ('LADA'); /* ошибка */
insert into test (col1)  values ('B9W');
```

8)

```
create or replace table test (col1 real check (col1 is not null));

insert into test (col1)  values (56.008);
insert into test; /* ошибка */
```

Общие правила

1) Конструкция <контролируемые значения записи> задает условие, которому должно удовлетворять значение столбца (столбцов) при операции добавления/изменения записи. Если <контролируемые значения записи> возвращает значение TRUE, то запись считается корректной, в противном случае – недопустимой.

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

2) <Контролируемые значения записи> проверяются в том же порядке, в каком они создавались.

3) Таблица может иметь произвольное число <контролируемых значений записи>, но суммарная длина всех оттранслированных <контролируемых значений записи> не должна превышать 4 Кбайт.

Примеры

1)

```
create or replace table test
(col1 int check ((col1 between 10 and 30) and (col1<>15) and
 (col1>12)));

insert into test (col1) values(11); /* ошибка */
insert into test (col1) values(15); /* ошибка */
insert into test (col1) values(37); /* ошибка */
insert into test (col1) values(20);
```

2)

```
create or replace table test (col1 int check ((col1 between 10 and
 30) and (col1<>15) and (col1>12)), col2 char(20) check (col2 not
 like 'A%') check (col2 not like 'Б%') check (col2 not like 'В%'),
 col3 real check (col3*0.5+45.90>0));

insert into test (col1, col2, col3) values (25, 'Москва', 56.89);
insert into test (col1, col2, col3) values (25, 'Воронеж',
 45.99); /* ошибка */
insert into test (col1, col2, col3) values (25, 'Урюпинск',
 -112.7); /* ошибка */
```

3) Следующие два оператора делают одно и то же:

```
create table t (i int check ((i>0) and (i<10)));
create table t (i int check (i>0) check (i<10));
```

Определение первичного ключа таблицы

Спецификация

См. спецификацию пункта «Создание таблицы».

Синтаксические правила

1) <Имена столбцов> должны быть предварительно определены в <спецификации столбцов таблицы> и не должны повторяться.

2) Нельзя создать первичный ключ на один и тот же столбец (или с одинаковым набором столбцов).

3) Атрибуты AUTOINC и PRIMARY KEY совместимы.

```
create table test(i int autoinc, primary key (i));
```

Общие правила

1) Конструкция <первичный ключ> определяет простой или составной первичный ключ записей таблицы. Столбцы, входящие в состав составного первичного ключа, не допускают NULL-значения.
2) Атрибут PRIMARY KEY определяет столбец, как простой (не составной) первичный ключ таблицы. Простым первичным ключом может быть объявлен только один столбец таблицы.
3) Ограничения столбца с атрибутом PRIMARY KEY:
   - не может содержать NULL-значений;
   - не может содержать дубликаты значений. В связи с этим столбец не должен иметь <значение по умолчанию> (DEFAULT-значения) и константное <подставляемое значение>, если допускается возможность добавления NULL-значения.
4) Значения отдельных столбцов составного первичного ключа могут дублироваться, но комбинация значений всех столбцов составного первичного ключа должна быть уникальной.

```
create or replace table t_pk (c_pk1 int, c_pk2 decimal, c_pk3
 char(10), primary key (c_pk1, c_pk3));
insert into t_pk (c_pk1, c_pk2, c_pk3) values
(10, 25.7,'aaa'),
(20, 67.5,'bbb'),
(10, 219.6, 'bbb'),
(20, 66.8, 'aaa');
10 25.7   aaa
20 67.5   bbb
10 219.6  bbb
20 66.8   aaa
Ошибка - повторное значение ключа
insert into t_pk (c_pk1, c_pk2, c_pk3) values (10, 25.7,'aaa'),

Недопустимое значение ключа
insert into t_pk (c_pk1, c_pk2, c_pk3) values (null, 25.7,'aaa');
insert into t_pk (c_pk1, c_pk2, c_pk3) values (40, 78.7, null);
```

Определение уникального ключа таблицы

Спецификация

См. спецификацию пункта «Создание таблицы».

Синтаксические правила

1) <Имена столбцов> должны быть предварительно определены в <спецификации столбцов таблицы> и не должны повторяться.
2) Для столбцов типа DATE, BOOLEAN, BLOB, EXTFILE атрибут UNIQUE недопустим.

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

3) Нельзя создать уникальный ключ на один и тот же столбец (или с одинаковым набором столбцов).
4) Разрешается одновременное указание ограничений целостности PRIMARY KEY и UNIQUE с одним и тем же набором столбцов (при этом создается только атрибут PRIMARY KEY), однако указание одного из этих атрибутов при существовании другого запрещено.

Примечание
Если ограничения PRIMARY KEY и UNIQUE заданы для одного и того же столбца, останется только PRIMARY KEY.

Пример.

```
create or replace table tst (i int, ch char(10), vch varchar(10),
 primary key(vch), unique(vch));
Столбец vch будет первичным, а не уникальным ключом.
```

5) Для одностолбцового уникального ключа разрешено устанавливать DEFAULT-значения.
6) Количество уникальных ключей таблицы ограничено количеством индексов таблицы.
7) В текущей реализации допускается суммарно максимум 250 простых (одностолбцовых) и 100 составных (многостолбцовых) индексов.

Общие правила

1) Конструкция <уникальный ключ> определяет простой или составной уникальный ключ записей таблицы. Столбцы, входящие в состав уникального составного ключа, могут содержать NULL-значения.
2) Значения столбцов составного уникального ключа могут дублироваться, но комбинация значений всех столбцов составного уникального ключа (без учета NULL-значений) должна быть уникальной.

```
create or replace table t_pk (c_pk1 int autorowid, c_pk2 char(10),
 c_pk3 char(10), c_pk4 char(10), unique (c_pk2, c_pk3, c_pk4));

insert into t_pk (c_pk2, c_pk3, c_pk4) values
  ('Иванов', 'Иван', 'Иванович'),
  ('Иванов', 'Петр', 'Иванович'),
  ('Иванов', 'Иван', 'Петрович'),
  ('Иванов', 'Иван', null),
  (null, null, null),
  (null, null, null);

select * from t_pk;

| 1| Иванов| Иван| Иванович|
| 2| Иванов| Петр| Иванович|
| 3| Иванов| Иван| Петрович|
| 4| Иванов| Иван| null    |
| 5| null  | null| null    |
```

```
| 6| null   | null| null     |
```

3) Конструкция UNIQUE(VALUE) означает, что все явно определенные столбцы (исключая псевдостолбцы типа ROWID) таблицы являются частью составного уникального ключа. Таблица в этом случае не должна иметь столбцы типа BLOB или EXTFILE.

```
Эти конструкции эквивалентны
create or replace table test(i int, j int, unique (value));
create or replace table test(i int, j int, unique (i, j));
```

Определение внешнего ключа таблицы

Спецификация

См. спецификацию пункта «Создание таблицы».

Синтаксические правила

1) <Имена столбцов>, задействованные во <внешнем ключе> должны быть предварительно определены в конструкции <спецификация столбцов таблицы>.

```
create or replace table tab
(id   int primary key,
 name char(10),
 col1 int not null,
 col2 char(5) not null,
 col3 decimal not null,
foreign key (col1, col2, col3) <ссылочная целостность>);
```

2) Столбцы, составляющие внешний ключ, не могут иметь NULL-значение. Для каждой таблицы может быть только один внешний ключ.

3) Нельзя создать два внешних ключа на один и тот же столбец (или с одинаковым набором столбцов).

4) Внешний ключ (как простой, так и составной) может включать AUTOINC-столбцы.

5) Если при задании <внешнего ключа> имя таблицы указывается без имени столбца (списка столбцов), то считается, что ссылка делается на первичный ключ дочерней таблицы.

Общие правила

1) Ссылочная целостность – это согласованность данных между связанными таблицами БД. Она обеспечивается путем комбинирования первичного или уникального ключа и внешнего ключа.

2) Первичный ключ – это столбец или группа столбцов, однозначно определяющие запись таблицы. Первичный ключ уникален: в таблице не может быть двух разных записей с одинаковыми значениями первичного ключа.

3) Уникальный ключ – аналог первичного ключа. Отличие в том, что столбцы уникального ключа могут иметь NULL-значение.

4) Внешний ключ – это столбец или группа столбцов, ссылающиеся на столбец или группу столбцов другой (или этой же) таблицы. Значения внешнего ключа могут

дублироваться. Т.е. внешний ключ обеспечивает соответствие значений полей одной записи родительской таблицы множеству значений полей записей дочерних таблиц.

5) Между двумя или более таблицами БД могут существовать отношения подчиненности, которые определяют, что для каждой записи главной (родительской) таблицы может существовать одна или несколько записей в подчиненной (дочерней) таблице.

6) Таблица, на которую ссылается внешний ключ, называется родительской таблицей, а столбцы, на которые ссылается внешний ключ – первичным (родительским) ключом. Сам внешний ключ создается в дочерней таблице.

7) Таблица, в которой задается ограничение внешнего ключа (опция REFERENCES), называется дочерней таблицей, а столбцы, на которые ссылается внешний ключ – первичным (родительским) ключом.

8) Для соблюдения ссылочной целостности требуется, чтобы любое значение внешнего ключа дочерней таблицы могло содержать только значения из первичного или уникального ключа родительской таблицы.

9) Внешние ключи могут создавать:

   - пользователи – владельцы родительской и дочерней таблиц;
   - пользователи с полномочиями ALTER для дочерней и REFERENCES для родительской таблиц;
   - создатель БД и пользователи с привилегией администратора БД (DBA).

10) Столбец строкового типа во внешнем ключе может ссылаться на столбец того же типа, но другой длины.

```
drop table tfk;
drop table tpk cascade;

create or replace table tpk(c1 char(3), c char(10) primary key, c2
 char(3));
create or replace table tfk(c1 char(3), c char(20) references tpk
 on update cascade, c2 char(3));
insert into tpk values ('111','AAA','222');
insert into tpk values ('abc','AAAAAA','xyz');
insert into tfk values ('111','AAA','222');
insert into tfk values ('111','AAAAAA','222');
select * from tpk;
select * from tfk;
update tpk set c='AAAAAA';
select * from tpk;
select * from tfk;

create or replace table tpk(c1 char(3), c char(10) primary key, c2
 char(3));
create or replace table tfk(c1 char(3), c char(20) references tpk
 on update cascade, c2 char(3));
```

11) Столбец типа DECIMAL во внешнем ключе может ссылаться на столбец того же типа, но с другими значениями масштаба и точности.

Определение ссылочной целостности таблицы

Спецификация

См. спецификацию пункта «Создание таблицы».

Синтаксические правила

1) Столбцы, на которые делается ссылка (<имя столбца>[, …]), должны быть столбцами PRIMARY KEY или UNIQUE.
2) <Спецификация родительской таблицы> определяет таблицу, на которую производится ссылка из данной дочерней таблицы.

Примеры.
1)
Родительская таблица

```
create or replace table
 "Специальности" (id int autoinc initial (100), name char(20));

insert into "Специальности"(name) values('Программист');
insert into "Специальности"(name) values('Инженер');
insert into "Специальности"(name) values('Дизайнер');
select * from  "Специальности";

|ID |NAME            |
|100|Программист     |
|101|Инженер         |
|100|Дизайнер        |
```

Дочерняя таблица

```
create or replace table "Штатное расписание"
 (id_person int autoinc, fio char(20),
 job int references "Специальности"(id_job));

insert into "Штатное расписание"(fio, job)
 values('Иванов И.И.', 102);
insert into "Штатное расписание"(fio, job)
 values('Петров П.П.', 100);
select * from  "Штатное расписание";

|ID_PERSON|FIO          |JOB|
|1        |Иванов И.И.  |100|
|2        |Петров П.П.  |102|
```

либо

```
create or replace table "Штатное расписание"
 (id_person int autoinc, fio char(20),
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
 job int references "Специальности"(id_job));
insert into "Штатное расписание"(fio, job)
 values('Иванов И.И.',
  (select id_job from "Специальности" where name='Программист'));
insert into "Штатное расписание"(fio, job)
 values('Петров П.П.',
  (select id_job from "Специальности" where name='Дизайнер'));

|ID_PERSON|FIO          |JOB|
|1        |Иванов И.И.  |100|
|2        |Петров П.П.  |102|

Ссылка на составной внешний ключ
Для исключения неоднозначности при полном совпадении ФИО вводятся
 дополнительные значения: год и место рождения.

create or replace table "Штатное расписание"
(id_person int autoinc,
 fio char(20),
 year int,
 birth_place char(10),
 job int references "Специальности"(id_job))

insert into "Штатное расписание"(fio, year, birth_place, job)
values('Иванов И.И.', 1998, 'Воронеж', 100);
insert into "Штатное расписание"(fio, year, birth_place, job)
values('Иванов И.И.', 1998, 'Москва', 102);
insert into "Штатное расписание"(fio, year, birth_place, job)
values('Петров П.П.', 1995, 'Воронеж', 100);

select * from  "Штатное расписание";

|ID_PERSON|FIO         |YEAR|BIRTH_PLAC      | JOB  |
|1        |Иванов И.И. |1998|Воронеж         | 100  |
|2        |Иванов И.И. |1998|Москва          | 102  |
|3        |Петров П.П  |1995|Воронеж         | 100  |

create table ref_sub (id int primary key, name char(10), id_ref
 int references par_tab(id_key));

2)
Ссылка на столбец в той же таблице:
create table tab1 (i int primary key, c char(10), ref_i int
 references tab1(i));
```

3) Разрешается не указывать список столбцов в конструкции REFERENCES, если ссылка производится на первичный ключ родительской таблицы.

```
create or replace table tab1 (i int, c1 char(10), c2 varchar(15),
 primary key primary key (i, c2));
create or replace table tab2 (i int, c2 varchar(20), foreign
 key(i, c2) references tab1);
```

4) В спецификации ссылки REFERENCES <спецификация родительской таблицы> и <имена столбцов> определяют имя таблицы и имена столбцов в этой таблице, на которые производится ссылка из таблицы объявляемого столбца (или столбцов).

```
create table ref_sub (id int primary key, name char(10), id_ref
 int references par_tab(id_key));

Ссылка на столбец в той же таблице:
create table tab1 (i int primary key, c char(10), ref_i int
 references tab1(i));
```

5) <Спецификация схемы таблицы> задает имя таблицы, на столбец (или столбцы) которой ссылается <внешний ключ>.

Общие правила

1) Опция MATCH задает ссылочную целостность с расширенными сравнениями ключей. Ссылочная целостность удовлетворяется (результат сравнения TRUE), если внешний ключ определяемой (дочерней) таблицы совпадает по заданному критерию с одним из первичных (уникальных) ключей главной (родительской) таблицы, в противном случае – FALSE. Пусть в описании ниже имя некоторой таблицы БД будет T, имя определяемой таблицы S.
2) Тип сравнения SIMPLE. Ограничение внешнего ключа (ссылочное ограничение) удовлетворяется в том и только в том случае, когда для каждой записи таблицы S выполняется одно из условий:
   - таблица T содержит в точности одну такую запись, что значение внешнего ключа в данной записи таблицы S полностью совпадает со значением соответствующего возможного ключа в этой записи таблицы T.

```
|PK (таблица T)|FK (таблица S)|Ссылочная целость удовлетворяется?|
|10, aaa       |10, aaa       |Да (есть PK= 10, aaa)              |
|20, bbb       |55, bbb       |Нет (нет соответствующего PK)      |
|10, bbb       |10, bbb       |Да (есть PK= 10, bbb)              |
|20, aaa       |20,aaa        |Да (есть PK= 20, aaa)              |
|10, ddd       |              |Нет (нет соответствующего PK)      |

create or replace table t_pk (c_pk1 int, c_pk2 decimal, c_pk3
 char(10), primary key(c_pk1, c_pk3));

create or replace table t_fk (c_fk1 int, c_fk2 decimal, c_fk3
 char(10), foreign key (c_fk1, c_fk3) references t_pk(c_pk1,
 c_pk3) match simple);

insert into t_pk (c_pk1, c_pk2, c_pk3) values
  (10, 25.7,'aaa'),
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
  (20, 67.5,'bbb'),
  (10, 219.6, 'bbb'),
  (20, 66.8, 'aaa');

select * from t_pk;

C_PK1 C_PK2 C_PK3
10    25.7  aaa
20    67.5  bbb
10    219.6 bbb
20    66.8  aaa

insert into t_fk (c_fk1, c_fk2, c_fk3)
values (10, -5.7, 'bbb'), (20, 409.8, 'aaa');
select * from t_fk;

C_FK1 C_FK2 C_FK3
10    -5.7  bbb
20    409.8 aaa

Ошибка: нет совпадения со значениями первичного составного ключа
insert into t_fk (c_fk1, c_fk2, c_fk3) values
(55, -5.7, 'bbb'), (20, 409.8, 'aaa');
```

- какой-либо столбец, входящий в состав внешнего ключа определяемой таблицы, содержит NULL-значение.

```
PK (таблица T) FK (таблица S) Ссылочная целость удовлетворяется?
20, bbb        null, bbb      Да (есть PK= bbb, null в FK)
30, ccc        null, eee      Да (есть null в FK)
               30, null       Да (есть PK= 30, null в FK)
               null, null     Да (есть null в FK)
               30, ccc        Да (есть PK=30, ccc)
               40, ddd        Нет (нет PK=4, ddd и нет null в FK)

insert into t_fk (c_fk1, c_fk2, c_fk3)
values(null, -5.7, 'bbb'l), (20,409.8, 'aaa');
select * from t_fk;

insert into t_fk (c_fk1, c_fk2, c_fk3)
values(null, -5.7, null), (20,409.8, 'aaa');
select * from t_fk;
```

3) Тип сравнения `MATCH  PARTIAL`. Ограничение внешнего ключа (ссылочное ограничение) удовлетворяется в том и только в том случае, когда для каждой записи таблицы `S` выполняется одно из условий:
   - таблица `T` содержит по крайней мере одну такую запись, что значение каждого столбца данной записи таблицы `S`, отличное от NULL-значения, совпадает со значением соответствующего столбца возможного ключа в этой записи таблицы `T`.

```
PK (таблица T)  FK (таблица S)
Ссылочная целость удовлетворяется?
10, aaa, 15.6   10, aaa, 15.6
Да (есть соответствующий первичный ключ)
20, bbb, 34.7   20, abc, 34.7
Нет (нет соответствующего первичного ключа)
30, ccc, 78.3   40, bbb, 77.8
Нет (нет соответствующего первичного ключа)
```

- каждый столбец, входящий в состав внешнего ключа, содержит NULL-значение.

```
PK (таблица T)  FK (таблица S)
Ссылочная целость удовлетворяется?
10, aaa, 15.6   20, null, null
Да (есть частичное совпадение: 20)
20, bbb, 34.7   20, null, -3
Нет (отсутствует частичное совпадение для 20, -3)
30, ccc, 78.3   null, bbb, 45.6
Нет (отсутствует частичное совпадение для bbb, 45.6)
                null, null,null
Да (игнорируется совпадение с первичным ключом)
                20, abc, null
Нет (отсутствует частичное совпадение для 20, abc)
                40, null, null
Нет (отсутствует частичное совпадение для 40)
                40, ddd, null
Нет (отсутствует частичное совпадение для 40, ddd)

create or replace table t_pk (c_pk1 int,
c_pk2 char(10), c_pk3 decimal, primary key(c_pk1, c_pk2, c_pk3));

insert into t_pk (c_pk1, c_pk2, c_pk3) values
(10,'aaa', 15.6),
(20, 'bbb', 34.7),
(30, 'ccc', 78.3);

select * from t_pk;
C_PK1 C_PK2 C_PK3
10    aaa   15.6
20    bbb   34.7
30    ccc   78.3

create or replace table t_fk (c_fk1 int, c_fk2 char(10), c_fk3
 decimal, foreign key (c_fk1, c_fk2, c_fk3) references t_pk(c_pk1,
 c_pk2, c_pk3)  match partial);

insert into t_fk (c_fk1, c_fk2, c_fk3)values(10, 'aaa', 15.6);
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
insert into t_fk (c_fk1, c_fk2, c_fk3)values(20, 'abc' 34.7);
insert into t_fk (c_fk1, c_fk2, c_fk3)values(40, 'bbb' 77.8);
insert into t_fk (c_fk1, c_fk2, c_fk3)values(20, null, null);
insert into t_fk (c_fk1, c_fk2, c_fk3)values(20, null, -3);
insert into t_fk (c_fk1, c_fk2, c_fk3)values(null, null, null);
insert into t_fk (c_fk1, c_fk2, c_fk3)values(20, 'abc', null);
insert into t_fk (c_fk1, c_fk2, c_fk3)values(40, null, null);
insert into t_fk (c_fk1, c_fk2, c_fk3)values(40, 'ddd', null);

select * from t_fk;

C_FK1 C_FK2 C_FK3
10    aaa   15.6
20    NULL  NULL
NULL  NULL  NULL
```

4) Тип сравнения MATCH FULL или тип сравнения не задан (используется по умолчанию). Ограничение внешнего ключа (ссылочное ограничение) удовлетворяется в том и только в том случае, когда для каждой записи таблицы S выполняется одно из условий:

   - ни один столбец, входящий в состав внешнего ключа, не содержит NULL-значение, и таблица T содержит в точности одну такую запись, что значение внешнего ключа в данной записи таблицы S совпадает со значением соответствующего возможного ключа в этой записи таблицы T.

```
PK (таблица T)  FK (таблица S)
Ссылочная целость удовлетворяется?
10, aaa, 15.6 10, aaa, 15.6
Да (полное совпадение с первичным ключом)
20, bbb, 34.7 20, bbb, null
Нет (отсутствует совпадение с первичным ключом)
30, ccc, 78.3 null, null, 78.3
Нет (отсутствует совпадение с первичным ключом)

create or replace table t_pk (c_pk1 int, c_pk2 char(10), c_pk3
 decimal, primary key(c_pk1, c_pk2, c_pk3));

insert into t_pk (c_pk1, c_pk2, c_pk3) values
(10,'aaa', 15.6),
(20, 'bbb', 34.7),
(30, 'ccc', 78.3);

select * from t_pk;
C_PK1 C_PK2 C_PK3
10    aaa   15.6
20    bbb   34.7
30    ccc   78.3
```

```
create or replace table t_fk (c_fk1 int, c_fk2 char(10), c_fk3
 decimal, foreign key (c_fk1, c_fk2, c_fk3) references t_pk(c_pk1,
 c_pk2, c_pk3)  match full);


insert into t_fk (c_fk1, c_fk2, c_fk3)values(10, 'aaa', 15.6);
insert into t_fk (c_fk1, c_fk2, c_fk3)values(20, 'bbb', null);
insert into t_fk (c_fk1, c_fk2, c_fk3)values(null, null, 78.3);

select * from t_fk;
C_FK1 C_FK2 C_FK3
10    aaa   15.6
```

- каждый столбец, входящий в состав внешнего ключа, содержит NULL-значение.

```
PK (таблица T) FK (таблица S)
Ссылочная целость удовлетворяется?
10, aaa, 15.6  null, null, null
Да (совпадение значений внешнего и первичного ключа не требуется)
20, bbb, 34.7
30, ccc, 78.3
```

5) <Спецификация действия> задает действие, которое должно быть выполнено с записью, содержащей внешний ключ (в дочерней таблице) при удалении/изменении первичного (уникального) ключа в родительской таблице:

   - CASCADE: если запись с первичным (уникальным) ключом в родительской таблице удаляется/изменяется, то удаляются/изменяются и все записи в дочерней таблице, ссылающиеся на удаляемую/изменяемую запись родительской таблицы при условии, что в <ссылочной целостности>:

     - отсутствует опция MATCH;
     - либо присутствуют опции MATCH SIMPLE или MATCH FULL.

     Если же в <ссылочной целостности> указана опция MATCH PARTIAL, то удаляются/изменяются только те записи дочерней таблицы, которые ссылаются исключительно на удаляемую/изменяемую запись родительской таблицы.

     То есть при <спецификация действия> CASCADE действия с записью в родительской таблице (удаление/изменение) полностью повторяется с записями в дочерних таблицах.

```
create or replace table "Родительская"
(i int autorowid,
 pk_col1 int autoinc initial (50),
 d decimal default 10.5,
 pk_col2 char(5), primary key (pk_col1, pk_col2));
insert into "Родительская"(pk_col2) values ('11111'), ('22222'),
 ('33333');
select * from "Родительская";

I  PK_COL1   D     PK_COL2
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
1  50        10.5  11111
2  51        10.5  22222
3  52        10.5  33333

create or replace table "Дочерняя"
(vc varchar(10) default 'aaaaa',
 fk_col1 int,
 col2 decimal default 45.8,
 fk_col3 char(5),
foreign key (fk_col1,fk_col3)
references "Родительская"(pk_col1, pk_col2) on delete cascade);

insert into "Дочерняя"(fk_col1, fk_col3)
values (50,'11111'), (51,'22222'), (52,'33333');
select * from "Дочерняя";
VC      FK_COL1   COL2  FK_COL3
aaaaa   50        45.8  11111
aaaaa   51        45.8  22222
aaaaa   52        45.8  33333

delete from "Родительская" where i=2;
select * from "Родительская";

I  PK_COL1  D     PK_COL2
1  50       10.5  11111
3  52       10.5  33333

select * from "Дочерняя";
VC      FK_COL1   COL2   FK_COL3
aaaaa   50        45.8   11111
aaaaa   52        45.8   33333
```

- `SET NULL`: если запись с первичным (уникальным) ключом в родительской таблице удаляется/изменяется, то во всех столбцах, которые входят в состав внешнего ключа, и во всех записях дочерней таблицы, ссылающихся на удаленную/измененную запись родительской таблицы, проставляется NULL-значение. Если в `<ссылочной целостности>` содержится опция `MATCH PARTIAL`, то NULL-значение устанавливается только в тех записях дочерней таблицы, которые ссылаются исключительно на удаляемую/изменяемую запись родительской таблицы.

  То есть записям дочерней таблицы будет присвоено NULL-значение при удалении/изменении соответствующих записей родительской таблицы. Правило будет выполняться, если поля дочерней таблицы допускают NULL-значения.

```
Спецификацию таблицы "Родительская" и её содержимое
см. в описании действия CASCADE.
select * from "Родительская";
```

```
I  PK_COL1   D     PK_COL2
1  50        10.5  11111
2  51        10.5  22222
3  52        10.5  33333

create or replace table "Дочерняя"
(vc varchar(10) default 'aaaaa',
 fk_col1 int,
 col2 decimal default 45.8,
 fk_col3 char(5),
foreign key (fk_col1,fk_col3) references "Родительская"(pk_col1,
 pk_col2) on update set null);

insert into "Дочерняя"(fk_col1, fk_col3)
values (50, '11111'), (51, '22222'), (52, '33333');
select * from "Дочерняя";

update "Родительская" set pk_col2='zzzzz' where i=2;
select * from "Родительская";
I  PK_COL1  D     PK_COL2
1  50       10.5  11111
2  51       10.5  zzzzz
3  52       10.5  33333
select * from "Дочерняя";
VC      FK_COL1   COL2   FK_COL3
aaaaa   50        45.8   11111
aaaaa   NULL      45.8   NULL
aaaaa   52        45.8   33333
```

- `SET DEFAULT`: если запись с первичным (уникальным) ключом в родительской таблице удаляется/изменяется, то во всех столбцах, которые входят в состав внешнего ключа, и во всех записях дочерней таблицы, ссылающихся на удаленную/измененную запись родительской таблицы, проставляется заданное при определении их столбцов значение по умолчанию. Комбинация всех подставляемых по умолчанию значений внешнего ключа должна соответствовать одному из первичных ключей (ключу по умолчанию). Если в определении внешнего ключа содержится опция `MATCH PARTIAL`, то значение по умолчанию устанавливается только в тех записях дочерней таблицы, которые ссылаются исключительно на удаляемую/изменяемую запись родительской таблицы.

  То есть в столбец (столбцы) записей дочерней таблицы заносятся значения по умолчанию, указанные при `<спецификации определении столбцов>` в родительской таблице. Если значение по умолчанию не определено, то операция удаление/изменения записи отменяется и возвращается соответствующий код завершения.

```
Спецификацию таблицы "Родительская" и её содержимое
см. в описании действия CASCADE.
select * from "Родительская";
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
I   PK_COL1  D      PK_COL2
1   50       10.5   11111
2   51       10.5   22222
3   52       10.5   33333

Комбинация значений по умолчанию 52, '33333' должна быть одним
из значений первичного ключа (ключа по умолчанию)
create or replace table "Дочерняя"
(vc varchar(10) default 'aaaaa',
 fk_col1 int default 52,
 col2 decimal default 45.8,
 fk_col3 char(5) default '33333',
foreign key (fk_col1,fk_col3) references "Родительская"(pk_col1,
 pk_col2) on update set default);

insert into "Дочерняя"(fk_col1, fk_col3)
values (50, '11111'), (51,'22222'), (52, '33333');
select * from "Дочерняя";
VC      FK_COL1   COL2  FK_COL3
aaaaa   50        45.8  11111
aaaaa   51        45.8  22222
aaaaa   52        45.8  33333

update "Родительская" set pk_col2='zzzzz' where i=2;
select * from "Родительская";

I   PK_COL1   D      PK_COL2
1   50        10.5   11111
2   51        10.5   zzzzz
3   52        10.5   33333

select * from "Дочерняя";
VC      FK_COL1   COL2   FK_COL3
aaaaa   50        45.8   11111
aaaaa   52        45.8   33333
aaaaa   52        45.8   33333
```

- NO ACTION: прекратить операцию (с выдачей соответствующего кода завершения), если её выполнение приводит к нарушению ссылочной целостности (ограничение RESTRICT является синонимом NO ACTION).

  То есть <спецификация действия> NO ACTION запрещает удаление/изменение записи в родительской таблице при наличии связанных с ней записей в дочерней таблице. Опция NO ACTION действует по умолчанию, если опции ON UPDATE и ON DELETE не заданы.

```
create or replace
  table test1(i1 int, j1 int, primary key(i1,j1));
```

```
insert into test1 values (1,1);
create or replace
  table test2(i2 int, j2 int, foreign key (i2,j2)
    references test1(i1,j1) match simple on delete cascade);
insert into test2 values(1,NULL);
insert into test2 values(NULL,4);
insert into test2 values(1,1);
insert into test2 values(NULL,NULL);

select * from test2;

 I2          J2
 --          --
|          1|           |
|           |          4|
|          1|          1|
|           |           |

delete from test1;
select * from test2;

 I2          J2
 --          --
|          1|           |
|           |          4|
|           |           |
```

6) Разрешается удалять записи со спецификацией действия `NO ACTION (RESTRICT)` и ссылающиеся на самих себя.

7) Определить `<внешний ключ>` можно как непосредственно при создании дочерней таблицы, так и позднее, с помощью оператора `ALTER TABLE … ADD FORIEGN KEY`. В обоих случаях родительская таблица должна быть предварительно создана.

Спецификация параметров таблицы

Спецификация

См. спецификацию пункта «Создание таблицы».

Синтаксические правила

1) Значение параметра `MAXROWID` – целое положительное число, не превосходящее `2^30-1`. Значение по умолчанию равно `1022`.

2) Значение параметра `MAXROW` – больше нуля и не должно превышать значение `MAXROWID`. По умолчанию равно `MAXROWID`.

3) Значение параметров `PCTFILL`, `PCTFREE` – целое положительное число, не превышающее `100`. По умолчанию значение `PCTFILL` равно `100`.

Примечание
Для таблиц с размером записи более 4Кб значения параметров PCTFILL, PCTFREE игнорируются, так как широкие записи хранятся в BLOB-файлах. Поэтому указанные при создании таблицы значения параметров PCTFREE и PCTFILL фактически не применяются, а применяется значение параметра BLOBPCT.

4) Значение параметра BLOBPCT – целое положительное число, не превышающее 100. По умолчанию равно 50.
5) <Количество файлов> таблицы любого вида не может быть больше 63.
6) <Имя устройства> – <символьный литерал> длиной не более четырех символов.
7) Максимальный <размер файла> данных/индексов/BLOB составляет 64 Гбайт.
8) Если <файловый параметр> не задан и в таблице отсутствуют BLOB-столбцы, то по умолчанию СУБД определит два файла минимальных размеров: файл данных и индексный файл и разместит их в каталоге БД.

Общие правила

1) Параметр MAXROWID задает планируемое максимальное количество записей в создаваемой таблице. При создании таблицы будет зарезервировано на диске столько страниц конвертера данных таблицы, чтобы их хватило для размещения информации о MAXROWID записей, т.е. параметр MAXROWID влияет только на количество страниц конвертера.
2) Параметр MAXROW задает число записей, для хранения которых будет зарезервировано место на диске сразу же при создании таблицы. При создании таблицы будет создан первоначальный файл данных для одновременного хранения в нем MAXROW записей (размер рассчитывается исходя из декларируемого размера записей).
3) Хранящиеся записи могут не занимать все ROWID, т.к. обычно MAXROWID задается «с запасом». Это делается в связи с тем, что ROWID удаленных записей начинают переиспользоваться только после того, как исчерпан соответствующий «запас». Подобное переиспользование приводит к некоторому замедлению процесса добавления записей в таблицу. С параметром MAXROW связаны и другие параметры таблицы: размеры файлов (в зависимости от MAXROW СУБД вычисляет значения этих размеров по умолчанию).
4) Если при добавлении записи значения параметров MAXROW и MAXROWID превышены, таблица будет автоматически расширяться, что приведет к замедлению работы СУБД ЛИНТЕР.

```
CREATE TABLE auto
  (personID int not NULL,
   make char(20) default '',
   model char(20))
   maxrowid 4096
   maxrow 1000
   pctfill 70;
```

5) Параметр PCTFREE задаёт порог, по которому СУБД определяет, что страница файла данных свободна для добавления в неё данных: а именно, страница файла считается свободной, если в ней есть не менее PCTFREE процентов свободного места (см. документ «Архитектура СУБД», раздел «Файлы данных»). В случае, если этот параметр равен 0 (не установлен), считается, что порог, задаваемый параметром PCTFREE, совпадает с порогом, задаваемым параметром PCTFILL. Значение по

умолчанию параметра PCTFREE – 0 (не установлен). Значение параметра PCTFREE зависит от значения PCTFILL (см. алгоритм вычисления PCTFREE от значения PCTFILL в документе «Рекомендации по настройке СУБД ЛИНТЕР», раздел «Предварительная разметка дискового пространства для таблицы»).

6) Параметр PCTFILL определяет порог занятости для страниц файла данных. Если свободного места в странице недостаточно для размещения заданного PCTFILL процента упакованной записи, то страница считается занятой (закрытой) для добавления (см. документ «Архитектура СУБД», раздел «Файлы данных»).

Параметр вычисляется по формуле

```
PCTFILL=L_Compress_Rec/L_Declare_Rec*100
  где:
  L_Compress_Rec – средняя длина упакованной записи;
  L_Declare_Rec – декларированная в конструкции CREATE TABLE длина
 записи.
  Упаковке (сжатию) подвергаются:
  – символьные данные (усекаются правосторонние концевые пробелы)
 в типах данных char, varchar, nchar, nvarchar и extfile;
  – байтовые данные (усекаются правосторонние концевые двоичные
 нули) в типах данных byte, varbyte.
Числовые и остальные типы данных не упаковываются.
```

7) Значение параметра PCTFILL может быть критичным, когда в таблице есть строковые поля большого размера, обычно заполняемые значительно более короткими значениями. Оптимальный процент сжатия в этом случае находится в пределах от 75% до 95% (PCTFILL от 5 до 25).

```
CREATE TABLE auto
  (personID int not NULL,
  make char(20) default '',
  model char(20))
  maxrowid 4096
  maxrow 1000
  pctfill 70;
```

8) Для таблиц с максимальным декларируемым размером записи, близким к 4 Кбайт, использовать значение PCTFILL по умолчанию категорически не рекомендуется, иначе размер файлов данных увеличится на несколько порядков при том же объеме данных.

```
CREATE TABLE BIG(C CHAR(4000));
INSERT INTO BIG SELECT CAST NULL AS CHAR FROM BIG1;
…
(всего 130000 записей)
Размер файла данных таблицы BIG получается равным 130 Мбайтам – по
 4 записи на страницу, хотя все записи пустые.
```

9) Параметр BLOBPCT задает процент заполнения BLOB-страниц создаваемой таблицы. BLOB-страница, заполненная до этого процента, будет использоваться только для расширения уже содержащихся в ней BLOB-данных, новые BLOB-данные размещаться в эту страницу не будут.

10) Опции INDEXFILES, DATAFILES, BLOBFILES задают количество, расположение и размер зарезервированных для таблицы файлов соответствующего типа.

11) Файлы таблицы могут быть размещены только на устройствах, <RAL>-уровень которых не ниже <RAL>-уровня таблицы.

12) <Имя устройства> в <описателе файла> указывает переменную операционной системы (среды окружения), содержащую спецификацию каталога, в котором будет создаваться файл таблицы или логическое имя устройства из таблицы $$$DEVICE (если таблица $$$DEVICE присутствует в БД). <Имя устройства> должно включать не более четырех символов. При размещении файла применяется следующий алгоритм:
    - если таблица $$$DEVICE имеется в БД, и значение <имя устройства> в ней найдено, файл создается в соответствии со спецификацией из этой таблицы;
    - если таблица $$$DEVICE отсутствует в БД, и пользователю разрешена работа с неописанными устройствами, то СУБД ЛИНТЕР будет рассматривать в качестве переменной среды окружения значение <имя устройства>;
    - если <имя устройства> не определено, фиксируется ошибка;
    - если <имя устройства> не задано в <определении таблицы>, то в качестве переменной среды окружения будет рассматриваться переменная SY00;
    - в случае, когда SY00 не определена, СУБД ЛИНТЕР будет создавать файл таблицы в текущем каталоге;
    - <размер файла> указывает число страниц (по 4096 байт) в файле. <Размер файла> должен быть не менее двух страниц. Если <размер файла> не задан, то система вычислит его, исходя из предыдущих параметров и установок по умолчанию.

```
CREATE TABLE Integer_Table
  (Integer_Column integer not NULL)
  pctfill 50
  indexfiles 1(4)
  datafiles 1('DB' 2);
```

13) Допускается создавать не более 63 файлов данных/индексов/BLOB каждого типа.

14) По умолчанию максимальная длина записи таблицы 65535 байт. Для установки другого максимального размера записи необходимо выполнить команду ALTER DATABASE SET RECORD SIZE LIMIT.

Создание таблицы с загрузкой данных

Спецификация

См. спецификацию пункта «Создание таблицы».

Синтаксические правила

1) Конструкция <загрузка таблицы> должна быть последней в конструкции <создание базовой таблицы>.

```
CREATE or replace TABLE tst
(personID int not NULL,
 make char(20) default '',
 model char(20))
 maxrowid 4096
 maxrow 1000
```

```
pctfill 70 as (select rowid, make, model
                  from auto);
```

2) Количество столбцов и типы данных столбцов создаваемой таблицы должны быть согласованы с результатами <подзапроса>, например:

```
create or replace table tab2
as (select make, to_date(to_char(year+1900), 'yyyy') from auto);
```

или

```
create or replace table tab3 as (select cast make as char(20) as
 c, to_date(to_char(year+1900), 'yyyy') as d from auto where  make
 like 'F%');
```

Создание таблицы, содержащей столбцы из нескольких таблиц

```
create or replace table tab1 (col1 int, col2 char(10));
create or replace table tab2 (col1 int, col2 char(10));

select tab1.col1, tab2.col2 from tab1, tab2
 where tab1.col1=tab2.col1;

insert into tab1  values(100, '12345');
insert into tab1  values(200, '67890');
insert into tab2  values(300, 'abcde');
insert into tab2  values(200, 'fdegh');
create or replace table tab (a int, b char(10))
as (select tab1.col1, tab2.col2
      from tab1, tab2
     where tab1.col1 = tab2.col1);
select * from tab;

|200| fdegh|
```

Общие правила

1) Конструкция <загрузка таблицы> создает таблицу с указанными параметрами и загружает в неё данные, возвращаемые <подзапросом>. Количество столбцов и типы данных столбцов создаваемой таблицы должны быть согласованы с результатами <подзапроса>, например:

```
create or replace table tab1 (c char(20), d date) as (select make,
 to_date(to_char(year+1900), 'yyyy') from auto);
```

или

```
create or replace table tab1 as (select cast make as char(20) as
 c, to_date(to_char(year+1900), 'yyyy') as d from auto);
```

2) Конструкция <создание базовой таблицы> с одновременной загрузкой в неё данных выполняется как одна транзакция, поэтому команда ROLLBACK отменяет создание таблицы вместе с загруженными в неё данными.

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

3) Если в создаваемой таблице для столбца задано подставляемое <значение из последовательности>, то в нем необходимо указывать опцию default, т.к. по умолчанию действует опция always, которая заставляет генерировать значения столбца автоматически (а не загружать их из <подзапроса>), например:

```
create or replace table tst (name char(20), i int generated by
default as identity (start with -100 increment by 1 maxvalue 0))
as (select make, rowid from auto limit 10);
```

4) Особенности конструкции <загрузка таблицы> при загрузке данных:

- атрибут AUTOROWID игнорируется (подставляется значение из <подзапроса>);

```
create or replace table tst (name char(20), i int autorowid)
as (select make, personid + 1000 from auto limit 10);

select name, i from tst limit 2;

|FORD   |1001|
|ALPINE |1002|
```

- атрибут AUTOINC INITIAL должен допускать вставку значения из <подзапроса>, например:

```
1) допустимый запрос (подставляемое значение 9 входит в диапазон
 от 1 до 9)
create or replace table tst (name char(20), i int autoinc initial
 (1)) as (select make, 9 from auto limit 1);

2) недопустимый запрос (декларированные подставляемые значения
 начинаются с 10, а возвращаемые подзапросом значения начинаются с
 1)
create or replace table tst (name char(20), i int autoinc initial
 (10)) as (select make, personid from auto limit 1);
```

- атрибут AUTOINC RANGE игнорируется (подставляется значение из <подзапроса>).

```
create or replace table tst (name char(20),
i int autoinc range (100:400, 500:1000))
as (select make, personid from auto limit 2);

insert into tst (name) values ('1111');

select name, i from tst limit 2;

|FORD   |  1|
|ALPINE |  2|
|1111   |100|
```

Создание глобальной временной таблицы

Спецификация

См. спецификацию пункта «Создание таблицы».

Синтаксические правила

1) Создание глобальной временной таблицы выполняется аналогично созданию базовой таблицы, с учетом запрета следующих спецификаций:
   - работа триггеров (за исключением триггеров DELETE для таблиц, созданных с предложением ON COMMIT DELETE ROWS). В этом случае временная таблица будет очищаться так же, как и при выполнении оператора TRUNCATE TABLE;
   - создание ссылочной целостности (ON UPDATE CASCADE, ON DELETE CASCADE);
   - поддержка BLOB-данных;
   - поддержка длинных записей (когда длина записи вместе с длиной заголовка превышает 4096 байт);
   - создание фразового индекса;
   - создание <идентификационных столбцов>;
   - создание внешних ссылок на глобальную временную таблицу;
   - создание конструкции ON UPDATE {…}.

Общие правила

1) Глобальные временные таблицы предназначены для хранения промежуточных результатов обработки запросов выборки данных или для использования в качестве рабочего пространства.
2) Данные в глобальных временных таблицах не сохраняются в БД. Они автоматически удаляются в конце сеанса работы с СУБД или при завершении транзакции, в которой они были созданы. При повторном использовании они должны заполняться новыми данными.
3) Глобальные временные таблицы имеют постоянно сохраняемые в схеме БД определения (до тех пор, пока временная таблица, подобно любой другой, не будет удалена с помощью оператора DROP TABLE). Работа с такими таблицами выполняется с помощью SQL-операторов манипулирования данными.
4) К глобальным временным таблицам можно применять DDL-операторы такие, как ALTER TABLE, CREATE INDEX, но только в те моменты, когда сессия не обращается к временной таблице и не связана с ней (например, не выполняет DML-операторы).
5) Данные в глобальных временных таблицах удаляются в конце каждого сеанса работы клиентского приложения с СУБД ЛИНТЕР. Данные глобальной временной таблицы одного сеанса клиентского приложения не доступны сеансам других клиентских приложений (и, следовательно, другим пользователям), данные в глобальных временных таблицах доступны для любой программы или модуля только в пределах сеанса.
6) Конструкция <действия при завершении транзакции> определяет те действия, которые должны быть выполнены при завершении транзакции, в рамках которой была создана временная таблица. При выполнении команды COMMIT

опция DELETE ROWS «очищает» таблицу, а PRESERVE ROWS – оставляет данные для выполнения следующей транзакции текущего сеанса СУБД («очищает» таблицу только после завершения сеанса). Если осуществляется откат транзакции (ROLLBACK) или прерывается ее выполнение, таблица возвращается к состоянию на конец предыдущей транзакции. При этом записи удаляются или сохраняются согласно следующему условию: сохранить состояние БД, соответствующее состоянию до отмененной транзакции. Если прерванная транзакция была первой в текущем сеансе, то таблица становится пустой. Значением по умолчанию является опция DELETE ROWS.

7) Данные временной таблицы не видны из других сеансов. Т.е. пользователи могут одновременно использовать одну и ту же временную таблицу, не пересекаясь данными, поэтому блокировка таблиц с помощью команды LOCK TABLE не требуется.

8) Рекомендации по использованию глобальных временных таблиц:

- сохранение результатов вызванной хранимой процедуры;

```
create or replace procedure proc_test (in i int)
result cursor(
  MAKE char(20),
  MODEL char(20))
declare
  var d typeof(result);//
code
  open d for direct
    "select make, model from auto where personid = " + itoa(i) +
 ";";//
  return d;//
end;

create or replace global temporary table tmp_tab
(make char(20), model char(20)) on commit delete rows
as (select * from proc_test(5));
```

- объединение данных из различных табличных объектов (базовых таблиц, представлений или других временных таблиц).

9) Глобальная временная таблица удаляется с помощью оператора DROP TABLE.

10) Внешние ссылки на глобальную временную таблицу запрещены.

11) Внутри глобальной временной таблицы внешние ссылки разрешены только на базовые таблицы.

```
create table parent_tab (id int primary key);
create global temporary table glb_tmp (id int primary key,name
 char(10),id_ref1 int, foreign key (id_ref1) references
 parent_tab(id) on delete cascade) on commit preserve rows;
```

Создание копии базовой таблицы

Спецификация

См. спецификацию пункта «Создание таблицы».

Синтаксические правила

1) <Исходная таблица> должна ссылаться на имя базой таблицы (представления запрещены).
2) <Исходная таблица> может ссылаться на глобальную временную таблицу. В этом случае <целевой таблицей> может быть как глобальная временная таблица, так и базовая.

```
create or replace global temporary table glb_tmp (id int primary
 key,name char(10),id_ref1 int on commit delete rows);
create or replace global temporary table glb_tmp_copy like
 glb_tmp;
create or replace table glb_tmp_copy like glb_tmp;
```

Общие правила

1) Конструкция <определение копии таблицы> используется для создания пустой <целевой таблицы> на основе определения <исходной таблицы>, включая все имена столбцов, их типы данных, их ненулевые ограничения и индексы, определенные в <исходной таблице>.

```
create or replace table t1 (i int);
create or replace table t2 like t1;
create or replace table "Тестер".t2 like t1;
```

2) Для создания таблицы пользователь должен иметь уровень прав RESOURCE или DBA.
3) Для создания копии таблицы на <исходную таблицу> требуется привилегия SELECT.
4) В момент выполнения копии таблицы <исходная таблица> не должна быть заблокирована.
5) При копировании таблицы файловые параметры <исходной таблицы> применяются для <целевой таблицы>.

```
create or replace table tst1(i int, j int) datafiles 2 ('SY00' 10,
 'SY00' 11) indexfiles 3 ('SY00' 10, 'SY00' 11, 'SY00' 25);
create or replace table tst2 like tst1;
inl> show tst1
…
* Файлов индексов            : 3 ("SY00" 10,"SY00" 11,"SY00" 25)
* Файлов данных              : 2 ("SY00" 10,"SY00" 11)

inl>show tst2
* Файлов индексов            : 3 ("SY00" 10,"SY00" 11,"SY00" 25)
* Файлов данных              : 2 ("SY00" 10,"SY00" 11)
```

6) При копировании таблицы внешние ключи не копируются.
7) При копировании таблицы информация о генерируемых столбцах <исходной таблицы> сохраняется в <целевой таблице>, поэтому в <целевой таблице> <генерируемые значения> формироваться будут.

```
create or replace table tst1(i int, j int, k int generated as (i
+j));
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
create or replace table tst2 like tst1;
inl>show tst1
…
K                   INTEGER GENERATED ALWAYS AS (("I"="J"))
inl>show tst2
…
K                   INTEGER GENERATED ALWAYS AS (("I"="J"))
insert into tst2(i,j) values(1,2);
insert into tst2(i,j) values(3,4);
select * from tst2;
I J K
1 2 3
3 4 7
```

Удаление таблицы (DROP TABLE)

Функция

Определение оператора удаления таблицы.

Спецификация

[1] <удаление таблицы>::=
DROP TABLE [<имя схемы>.]<имя таблицы> [CASCADE]

Синтаксические правила

1) <Имя таблицы> должно задавать базовую таблицу.
2) При удалении таблицы, созданной в текущей схеме, допускается не указывать <имя схемы>.

```
DROP TABLE Auto;
DROP TABLE System.Auto;
```

Общие правила

1) Удалить таблицу может только ее владелец. Администратор БД имеет возможность удалить одновременно все объекты некоторого пользователя с помощью команды DROP USER <имя пользователя> CASCADE.
2) Удалить можно только незаблокированную таблицу (т.е. такую таблицу, с которой в данный момент не работают другие пользователи).
3) При запуске СУБД ЛИНТЕР с ключом /COMPATIBILITY=STANDARD запрещается обычное удаление таблицы, на которую ссылаются представления (VIEW). При этом на консоль ядра СУБД и в файл linter.out выдаётся список объектов, ссылающихся на удаляемый объект и не позволяющих его удалить. Пример списка:

```
INFO Can't delete relation '"SYSTEM"."TEST1" (#187)' because
 exists reference(s):
  "SYSTEM"."TESTV" (#188)
  "SYSTEM"."TESTV2" (#190)
```

4) Каскадное удаление таблицы, на которую ссылаются представления (VIEW), разрешено всегда.

5) Если на удаляемую таблицу наложено ограничение целостности `ON DELETE CASCADE`, то удаление таблицы возможно только после предварительного удаления всех ссылающихся на нее таблиц. Если задан модификатор `CASCADE`, ссылки на таблицу удаляются автоматически.

6) Удаление системных таблиц запрещено.

7) Если создана таблица с именем, совпадающим с именем системной таблицы, но с другой структурой, то ее можно удалить.

```
create user S identified by '12345678';
grant DBA to S;
create table $$$SYSRL (I int);
insert into $$$SYSRL values (100);
drop table $$$SYSRL;
```

8) При удалении таблицы связанные с нею хранимые события не удаляются.

9) Для удаления хранимых событий при удалении таблицы необходимо в команде удаления таблицы указать опцию `cascade`:

```
drop table <имя таблицы> cascade;
```

10) Одновременно с удалением таблицы, для которой создано правило репликации, удаляется и само правило.

11) При каскадном удалении таблицы удаляется и связанное с ней глобальное (хранимое) событие.

Модификация таблицы (ALTER TABLE)

Функция

Определение оператора изменения структуры и/или атрибутов базовой таблицы.

Спецификация

[1] <модификация таблицы>::=
ALTER TABLE [<имя схемы>.]<имя таблицы>
{
<переименование таблицы>
|<изменение функциональности таблицы>
|<изменение уровня мандатного контроля доступа таблицы>
|<модификация триггеров>
|<модификация файловых параметров таблицы>
}

[2] <переименование таблицы>::= RENAME TO <новое имя таблицы>

[3] <изменение функциональности таблицы>::=
{
<добавление первичного ключа>
|<добавление уникального ключа>
|<добавление внешнего ключа>
|<добавление CHECK-условия таблицы>
|<удаление первичного ключа>
|<удаление уникального ключа>
|<удаление внешнего ключа>
|<удаление CHECK-условий таблицы>
|<установка режима циклической таблицы>
|<отмена режима циклической таблицы>
}

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

[4] <изменение уровня мандатного контроля доступа таблицы> : : =
SET LEVEL (<RAL>, <WAL>)
[5] <модификация триггеров> : : = REBUILD ALL TRIGGERS
[6] <модификация файловых параметров таблицы> : : =
{
<добавление файла>
|<модификация файла>
|<удаление файла>
}
[7] <добавление первичного ключа> : : =
ADD PRIMARY KEY (<имя таблицы>[, …])
[<режим модификации индекса>]
[8] <добавление уникального ключа> : : =
ADD UNIQUE (<имя таблицы>[, …] | VALUE) [<режим модификации индекса>]
[9] <добавление внешнего ключа> : : =
ADD FOREIGN KEY (<имя столбца>[, …]) <ссылочная целостность>
[<режим модификации индекса>]
[10] <добавление CHECK-условия таблицы> : : =
ADD CHECK <логическое выражение>
[11] <удаление первичного ключа> : : =
DROP PRIMARY KEY [WITHOUT INDEX]
[12] <удаление уникального ключа> : : =
DROP UNIQUE (<имя столбца>[, …] | VALUE) [WITHOUT INDEX]
[13] <удаление внешнего ключа> : : =
DROP FOREIGN KEY (<имя столбца>[, …]) [WITHOUT INDEX]
[14] <удаление CHECK-условий таблицы> : : = DROP CHECK
[15] <установка режима циклической таблицы> : : =
SET RECORDS LIMIT <количество записей>
[16] <отмена режима циклической таблицы> : : = CANCEL RECORDS LIMIT
[17] <добавление файла> : : =
ADD {INDEXFILE | DATAFILE | BLOBFILE} [([<имя устройства>] [<размер файла>][, …])]
[18] <модификация файла> : : =
MODIFY {INDEXFILE | DATAFILE | BLOBFILE} <номер файла> [([<имя устройства>]
[<размер файла>][, …])]
[19] <удаление файла> : : = DROP {INDEXFILE | DATAFILE | BLOBFILE}
[20] <ссылочная целостность> : : =
REFERENCES [<имя схемы>.]<имя таблицы> [(<список столбцов>)]
[MATCH {SIMPLE | FULL | PARTIAL}]
[<спецификация действий>]
[21] <режим модификации индекса> : : =
[INDEXFILE <номер файла>] [BY APPEND]
[22] <новое имя таблицы> : : = <идентификатор>

Общие правила

1) Для выполнения команды необходимы уровень доступа RESOURCE к БД и привилегия доступа ALTER к изменяемой таблице.

Синтаксические правила переименования таблицы

1) При переименовании таблицы нельзя изменить ее схему.

Общие правила переименования таблицы

1) Переименование системных таблиц не допускается.

2) Операция переименования таблицы не проверяет возможное использование заменяемого имени таблицы в других объектах БД (представлениях, синонимах, хранимых процедурах), поэтому, если такие объекты существуют, они перестанут работать.

Примеры

```
alter table tab rename to "Заказы";
alter table user1.sale rename to "Продажи";
```

Синтаксические правила добавления первичного ключа

1) Запрещено добавление атрибута PRIMARY KEY для псевдостолбцов (типа ROWID, ROWTIME).
2) Запрещено создание PRIMARY KEY на ту же комбинацию столбцов, на которую уже есть уникальный ключ (UNIQUE) с тем же порядком столбцов (верно как для простых, так и для составных ключей).
3) Комбинации атрибутов PRIMARY KEY и NULL недопустима.
4) Для столбцов типа DATE, BOOLEAN, BLOB, EXTFILE атрибут PRIMARY KEY недопустим.

Общие правила добавления первичного ключа

1) Изменение первичного ключа с помощью его переопределения не допускается. В этом случае необходимо сначала его удалить (см. конструкцию <удаление первичного ключа>), а потом создать заново.

```
create or replace table tst (i smallint primary key);
alter table tst add column c char(10);
alter table tst add primary key (i,c); // ошибка
alter table tst drop primary key;
alter table tst add primary key (i,c); // нормальное завершение
```

Примеры

```
create or replace table tst (i smallint);
alter table tst add primary key (i);

create or replace table tst (i smallint, ch char(10));
alter table tst add primary key (i,ch);
```

Синтаксические правила добавления уникального ключа

1) Запрещено добавление атрибута UNIQUE для псевдостолбцов (типа ROWID, ROWTIME).
2) Запрещено создание уникального ключа на ту же комбинацию столбцов, на которую уже есть первичный ключ (PRIMARY KEY) или уникальный ключ с тем же порядком столбцов. Привило распространяется для простых и для составных ключей.
3) Для столбцов типа DATE, BOOLEAN, BLOB, EXTFILE атрибут UNIQUE недопустим.

Общие правила добавления уникального ключа

1) Конструкция UNIQUE (VALUE) создаёт уникальный ключ из всех столбцов таблицы (за исключением псевдостолбцов).

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
// создание уникального ключа из столбцов i, ch, d
create or replace table tst (i smallint, ch char(10), d float);
alter table tst add unique (value);
```

2) Конструкция UNIQUE (VALUE) создаёт уникальный ключ только из тех столбцов, которые существуют в таблице на текущий момент. Если в таблицу будут добавлены новые столбцы, они автоматически не войдут в составной уникальный ключ – надо будет повторить операцию UNIQUE (VALUE).

3) Опция INDEXFILE <номер файла> задает номер индексного файла, в котором должен храниться индекс добавляемого столбца. Если опция не задана, то для таблиц, имеющих более одного индексного файла, по умолчанию используется значение 2.

```
create or replace table tst (i int unique, j int) indexfiles 3;
create or replace index i on tst indexfile 2;
alter table tst add unique(j) indexfile 3;
```

4) Опция BY APPEND задает режим создания индекса не через сортировку данных (как делается по умолчанию), а через последовательное добавление элементов.

```
create or replace table tst (i int unique, j int) indexfiles 3;
alter table tst add unique(j) by append;
```

Примеры

```
create or replace table tst (i smallint, ch char(10));
alter table tst add unique (ch);

create or replace table tst (i smallint, ch char(10));
alter table tst add unique (i, ch);
```

Синтаксические правила добавления внешнего ключа

1) Описание опции MATCH и конструкции <спецификация действий> приведено в пункте «Создание таблицы. Внешний ключ».

2) Если <список столбцов> ссылается на первичный ключ таблицы, то этот <список столбцов> можно не указывать.

```
create or replace table tab1 (i1 int);
create or replace table tab2 (i2 int primary key);

alter table tab1 add foreign key (i1) references tab2;
```

эквивалентная конструкция:

```
alter table tab1 add foreign key (i1) references tab2(i2);
```

Общие правила добавления внешнего ключа

1) В таблицу можно добавить несколько внешних ключей.

```
create or replace table tab1 (i1 int, c1 char(10));
create or replace table tab2 (i2 int primary key, c2 char(10)
 unique);
```

```
alter table tab1 add foreign key (i1) references tab2;
alter table tab1 add foreign key (c1) references tab2(c2);
```

Примеры

```
alter table ref_sub1a add foreign key (id_ref) references
 ref_main(id) on delete no action;

alter table ref_sub1a_1b add foreign key (id_ref1) references
 ref_sub1a(id) on delete cascade;

alter table ref_sub1a_1b add foreign key (id_ref2) references
 ref_sub1b(id) on delete set null;
```

Синтаксические правила добавления CHECK-условия таблицы

1) Использование столбцов с атрибутом AUTOROWID, псевдостолбцов ROWID, ROWTIME и DBROWTIME и предиката [NOT] CONTAINS (см. документ «Полнотекстовый поиск в базе данных», пункт «Предикат полнотекстового поиска») в <логическом выражении> запрещено.

2) В <логическом выражении> разрешено использование псевдозначений SYSDATE, NOW, LOCALTIME, LOCALTIMESTAMP, CURRENT_DATE, CURRENT_TIME, CURRENT_TIMESTAMP. Однако при этом в БД могут оказаться записи, которые удовлетворяли ограничению CHECK в момент выполнения операции INSERT/UPDATE, но со временем перестали удовлетворять этому ограничению, т.к. текущее значение этих псевдозначений стало другим.

```
create or replace table tst(i int);
alter table tst add check (i<datesplit(sysdate, 'Y')-2000);
insert into tst values(5);
insert into tst values(15);
insert into tst values(250); // ошибка
```

3) Таблица может иметь произвольное число ограничений целостности типа CHECK, но суммарная длина всех оттранслированных CHECK-условий не должна превышать 4 Кбайт.

```
create or replace table tst(i int, ch char(10), d float);
alter table tst add check (i<(datesplit(sysdate, 'Y')-2000));
alter table tst add check (to_number(ch) > i);
alter table tst add check (i+d > 100.78);
```

или эквивалентная конструкция

```
alter table tst add check (i<(datesplit(sysdate, 'Y')-2000)
  and to_number(ch) > i and (i+d > 100.78));
```

4) Ограничение целостности CHECK для таблицы может содержать ссылки на любые столбцы этой же таблицы.

5) Ограничение целостности типа CHECK не может содержать подзапросов, агрегатных функций, конструкций CASE и CAST, функций SECURITY, INDEXTIME, LINIDXINFO, LEAD, LAG, GREATEST, LEAST, RANK, GETTEXTPOS, GET_EVENTS_STATE.

Общие правила добавления CHECK-условия таблицы

1) Ограничения целостности типа CHECK для таблицы проверяются в том же порядке, в каком они создавались.
2) Добавленное CHECK-условие присоединяется по логическому «И» ко всем ранее заданным условиям CHECK.
3) При выполнении операции DELETE CHECK-условия не проверяются.

Общие правила изменения уровня мандатного контроля доступа таблицы

1) <Идентификатор> задает имя соответствующего уровня доступа (на чтение данных из таблицы (RAL) или на добавление/корректировку данных (WAL)). Допускается назначать только существующие в БД уровни доступа.
2) Указание мандатного доступа запрещено для системных таблиц.
3) Более подробно работа с уровнями мандатного контроля доступа описана в документе «Администрирование комплекса средств защиты данных», пункт «Управление уровнем доступа к таблице».
4) Отменить (изменить) мандатный уровень доступа к таблице невозможно (например, нельзя хранящиеся в БД секретные данные объявить несекретными).

Примеры

```
create or replace table RT (ID INT AUTOINC PRIMARY KEY,IDR INT,
 NNAME CHAR(25));
create level LEVEL3=3;
ALTER table RT SET LEVEL(LEVEL3,LEVEL3);
ALTER table RT SET COLUMN ID LEVEL(LEVEL3,LEVEL3);
```

Синтаксические правила удаления первичного ключа

1) Опция WITHOUT INDEX задает режим отмены атрибута столбца без удаления построенного по этому столбцу индекса (например, был создан столбец, для него создан индекс, потом столбец был объявлен как PRIMARY KEY, а затем этот атрибут столбца отменяется с данной опцией – индекс останется).
2) По умолчанию (опция WITHOUT INDEX не задана) индекс удаляется.

Общие правила удаления первичного ключа

1) Таблица, к которой применяется конструкция <удаление первичного ключа>, должна иметь первичный ключ.
2) Удалить первичный ключ можно только в том случае, если на него нет внешних ссылок из других таблиц, иначе надо предварительно удалить все внешние ссылки на него (см. конструкцию <удаление внешнего ключа>).

Пример

Удаление первичного ключа, на который есть ссылки внешних ключей.

```
create or replace table tab1 (i int, c1 char(10), c2 varchar(15),
 primary key (i,c2));
```

```
create or replace table tab2 (i int, c2 varchar(20), foreign
 key(i,c2) references tab1);

1) Получить список таблиц, у которых есть внешние ключи,
 ссылающиеся на удаляемый первичный ключ (например, в таблице
 TAB1). Это можно сделать с помощью представления FOREIGN_KEYS
 (создается в демонстрационной БД)
select fktable_name, fkcolumn_name
  from foreign_keys where pktable_name='TAB1';

FKTABLE_NAME  FKCOLUMN_NAME
TAB2      I
TAB2      C2

2) Удалить внешний ключ
alter table tab2 drop foreign key (i, c2);

3) Удалить первичный ключ
alter table tab1 drop primary key;
```

Синтаксические правила удаления уникального ключа

1) Опция `WITHOUT INDEX` задает режим отмены атрибута столбца без удаления построенного по этому столбцу индекса (например, был создан столбец, для него создан индекс, потом столбец был объявлен как UNIQUE, а затем этот атрибут столбца отменяется с данной опцией – индекс останется).
2) По умолчанию (опция `WITHOUT INDEX` не задана) индекс удаляется.

Общие правила удаления уникального ключа

1) Таблица, к которой применяется конструкция <удаление уникального ключа>, должна иметь соответствующий уникальный ключ.

```
create or replace table tst (i int, ch char(10), unique (i, ch));
alter table tst drop unique (i, ch);
```

2) Если для удаления уникального ключа используется опция `VALUE`, то удаляемый уникальный ключ тоже должен быть создан с помощью этой опции.

Примеры

```
1)
create or replace table
  tst (i int, ch char(10), d float, unique (i, ch));

alter table tst drop unique (value); // ошибка
2)
create or replace table tst (i int, ch char(10), d float);

alter table tst add unique (value);
alter table tst drop unique (value); // OK
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

Синтаксические правила удаления внешнего ключа

1) Опция WITHOUT INDEX задает режим отмены атрибута столбца без удаления построенного по этому столбцу индекса (например, был создан столбец, для него создан индекс, потом столбец был объявлен как FOREIGN KEY (или вошел в состав внешнего ключа), затем этот атрибут столбца отменяется данной опцией – индекс останется).
2) По умолчанию (опция WITHOUT INDEX не задана) индекс удаляется.

Общие правила удаления внешнего ключа

1) Таблица, к которой применяется конструкция <удаление внешнего ключа>, должна иметь соответствующий внешний ключ.

```
create or replace table tab1 (i1 int, c1 char(10));
create or replace table tab2 (i2 int primary key, c2 char(10)
 unique);


alter table tab1 add foreign key (i1) references tab2;
alter table tab1 add foreign key (c1) references tab2(c2);

alter table tab1 drop foreign key (c1);
```

Общие правила удаления CHECK-условий таблицы

1) Удаляются сразу все CHECK-условия таблицы. Выборочное удаление CHECK-условия не поддерживается.
2) CHECK-условия для столбцов остаются неизменными

Пример

```
alter table tst drop check;
```

Синтаксические правила установки режима циклической таблицы

1) Значение параметра <количество записей> должно быть в интервале от 1 до 1073741823 (0x3FFFFFFF).

Общие правила установки режима циклической таблицы

1) Конструкция SET RECORDS LIMIT устанавливает режим циклической таблицы для указанной базовой таблицы.
2) Параметр <количество записей> задает максимальное количество записей в циклической таблице. В случае превышения указанного значения новые записи будут размещаться на месте самых старых.

```
create or replace table ct(i int);
insert into ct values(1),(2),(3), (4),(5);
select * from ct;
I
1
2
3
```

```
4
5
alter table ct set records limit 3;
insert into ct values(10),(20),(30), (40),(50);
select * from ct;
I
40
50
30
```

3) Если на момент установки циклического режима таблица содержала большее количество записей, чем задано в параметре <количество записей>, то все записи, номера (ROWID) которых превышают значение <количество записей>, будут удалены. При их удалении будут срабатывать, если созданы, триггеры «before delete .. for each row» и «after delete .. for each row».

```
create or replace table ct(i int);
insert into ct values(1),(2),(3), (4),(5);
select * from ct;
I
1
2
3
4
5
alter table ct set records limit 3;
select * from ct;
I
1
2
3
```

4) Для циклических таблиц поддерживаются все типы триггеров, кроме «instead of delete». Попытка преобразовать таблицу с имеющимися триггерами «instead of delete» в циклическую или создать такой триггер для уже существующей циклической таблицы будет неуспешной.

5) В случае автоматического удаления старых записей при добавлении новых записей в циклическую таблицу поддерживается следующий порядок срабатывания триггеров (если соответствующие триггеры созданы):
   - триггеры «before insert .. for each statement» для добавляемой записи;
   - триггеры «before delete .. for each row» для удаляемой записи;
   - триггеры «after delete .. for each row» для удаляемой записи;
   - триггеры «before insert .. for each row» для добавляемой записи;
   - триггеры «after insert .. for each row» для добавляемой записи;
   - триггеры «after insert .. for each statement» для добавляемой записи.

6) Особенности добавления новых записей в переполненную циклическую таблицу из подзапроса (INSERT FROM SELECT) см. в разделе «Добавление записи» (оператор INSERT).

7) Для системных таблиц циклический режим запрещен.

Общие правила отмены режима циклической таблицы

1) Конструкция CANCEL RECORDS LIMIT отменяет ранее установленный режим циклической таблицы. Манипулирование записями таблицы будет выполняться в соответствии с ранее установленными параметрами MAXROWID, MAXROW.

Пример

```
alter table ct cancel records limit;
```

Общие правила модификации триггеров

1) Команду REBUILD ALL TRIGGERS может выполнить только владелец таблицы с уровнем прав не менее RESOURCE.
2) Для повторной трансляции всех триггеров, настроенных на модифицированную таблицу, необходимо выполнить команду вида:

```
ALTER TABLE <имя таблицы> REBUILD ALL TRIGGERS;
```

3) При выполнении команды REBUILD ALL TRIGGERS триггеры перетранслируются последовательно.

   Если после модификации структуры таблицы код триггера оказался некорректным, то при выполнении команды REBUILD ALL TRIGGERS будет выдано сообщение «ошибка трансляции запроса» из-за получения кода завершения 2210. При этом в консоль ядра будет выведено сообщение вида:
   ```
   ERROR: cannot rebuild trigger "SYSTEM"."T_AI" (#36) with procedure #129
   ```

   и дальнейшая повторная трансляция триггеров будет прекращена.
4) Для получения списка триггеров, для которых повторная трансляция завершилась не успешно, необходимо использовать следующий запрос:

```
select $$$NAME from $$$trig where 1 = (select count(*) from $$
$proc where $$$id = $$$proc);
```

5) После коррекции исходного кода триггера необходимо повторно выполнить команду REBUILD ALL TRIGGERS для повторной трансляции прочих триггеров таблицы.

Пример

```
create or replace table t_stat (c varchar(10));
create or replace table t_at(i int);
create or replace trigger t_ai after insert on t_at for each
statement execute
code
execute direct ("INSERT INTO T_STAT VALUES ('INSERT');");
end;
alter table t_at add column j int;
alter table t_at rebuild all triggers;
insert into t_at values (100, 1);
select * from t_at;
select * from t_stat;
```

Синтаксические правила добавления файла

1) <Символьный литерал> задает идентификатор устройства, на котором должен расплагаться добавляемый файл.
2) Устройство должно быть предварительно создано с помощью команды CREATE DEVICE, например:

```
create device "DB00" directory 'c:\linter\db\DEMO';
```

3) Длина <символьного литерала> должна быть 4 символа.
4) Если <имя устройства> не задано, то по умолчанию используется устройство, выделенное пользователю, выполняюшему команду, а если и оно не назначено, то SY00.

Примечание
В момент выполнения команды не проверяется наличие в ОС переменной окружения SY00.

Общие правила добавления файла

1) Суммарное количество файлов таблицы не должно превышать 63.
2) <Размер файла> задается в страницах (страница равна 4 Кбайтам).

Примеры

```
SY00=c:\linter\db\DEMO
DB00=d:\db_market

create or replace table tst1 (i int) datafiles 3;
SQL>show tst1
…
Файлов данных: 3 ("SY00" 7, "SY00" 2, "SY00" 2)

! файл на DB00, размер файла 30 страниц
alter table tst1 add datafile ('DB00' 30);
SQL>show tst1
…
Файлов данных: 4 ("SY00" 7, "SY00" 2, "SY00" 2, "DB00" 30)

! файл на DB00, размер файла 2 страницы
alter table tst1 add datafile ('DB00');
SQL>show tst1
…

Файлов данных: 5 ("SY00" 7, "SY00" 2, "SY00" 2, "DB00" 30, "DB00"
 2)

! файл на SY00, размер файла 100 страниц
alter table tst1 add datafile (100);
SQL>show tst1
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
…
Файлов данных: 6 ("SY00" 7, "SY00" 2, "SY00" 2, "DB00" 30, "DB00"
 2, "SY00" 100)

! файл на SY00, размер файла 2 страницы
alter table tst1 add datafile;
SQL>show tst1
…
Файлов данных: 7 ("SY00" 7, "SY00" 2, "SY00" 2, "DB00" 30, "DB00"
 2, "SY00" 100, "SY00" 2)

alter table tst1 add datafile ();
SQL>show tst1
…

Файлов данных: 8 ("SY00" 7, "SY00" 2, "SY00" 2, "DB00" 30, "DB00"
 2, "SY00" 100, "SY00" 2, "SY00" 2)
```

Общие правила модификации файла

1) <Номер файла> должен ссылаться на существующий файл таблицы.
2) <Размер файла> может быть изменен только в сторону увеличения.
3) Узнать текущий размер файла можно с помощью команды show утилиты inl или SQL-функции LINTER_FILE_SIZE, например,

```
! узнать идентификатор таблицы MOD_FILE:
select $$$s11 from $$$sysrl where $$$s13='MOD_FILE';
729
! узнать размер 3-го datafile таблицы MOD_FILE
select LINTER_FILE_SIZE (729, 1, 3);
5
```

4) <Размер файла> задается в страницах (страница равна 4 Кбайтам).

Пример

```
create or replace table mod_file (i int) DATAFILES 5;
alter table mod_file modify datafile 3 (20);
```

Общие правила удаления файла

1) Конструкция удаляет из таблицы последний файл указанного типа (т.е. файл с максимальным номером). Первый файл удалить нельзя.
2) Удаляемый файл не должен содержать данных и не использоваться в момент выполнения данной команды.



Примечание
Проверить заполненность файлов можно с помощью ключей -fd, -fi, -fb утилиты testdb (см. документ «Тестирование базы данных», пункты «Вывод ROWID по экстенту файла данных», «Вывод ROWID по экстенту файла индексов», «Вывод ROWID по экстенту BLOB-файла»).

Пример

```
create or replace table mod_file (i int) DATAFILES 5;
alter table mod_file drop datafile;
```

Модификация столбцов таблицы

Функция

Определение оператора изменения структуры и/или атрибутов столбцов базовой таблицы.

Спецификация

[1] <модификация столбцов таблицы>::=
ALTER TABLE [<имя схемы>.]<имя таблицы>
{
<переименование столбца>
|<изменение длины столбца>
|<добавление столбцов>
|<удаление одиночного столбца>
|<удаление группы столбцов>
|<изменение типа данных столбца>
|<добавление CHECK-условия столбца>
|<удаление CHECK-условия столбца>
|<установка значения по умолчанию>
|<удаление значения по умолчанию>
|<установка фильтра по умолчанию>
|<добавление диапазона автоинкрементных значений>
|<установка каталога внешних файлов>
|<установка последовательности значений>
|<отмены/разрешения NULL-значений>
|<изменение уровня мандатного контроля доступа столбца>
}

[2] <переименование столбца>::=
ALTER [COLUMN] <текущее имя столбца>
RENAME TO <новое имя столбца>

[3] <изменение длины столбца>::=
ALTER [COLUMN] <имя столбца> SIZE <длина>

[4] <добавление столбцов>::=
ADD {<добавление столбца> | <добавление группы столбцов>}

[5] <добавление столбца>::= COLUMN <определение столбца>

[6] <добавление группы столбцов>::= (<определение столбца>[, …])

[7] <удаление одиночного столбца>::=
DROP [COLUMN] <имя столбца> [CASCADE]

[8] <удаление группы столбцов>::=
DROP (<имя столбца>[, …]) [CASCADE]

[9] <изменение типа данных столбца>::=
ALTER [COLUMN] <имя столбца> SET DATA TYPE <тип данных>

[10] <добавление CHECK-условия столбца>::=
ALTER [COLUMN] <имя столбца> ADD CHECK <логическое выражение>

[11] <удаление CHECK-условия столбца>::=
ALTER [COLUMN] <имя столбца> DROP CHECK

[12] <установка значения по умолчанию>::=
ALTER [COLUMN] <имя столбца>
SET DEFAULT {[()<значение>[)] | (<значимое выражение>)}

© Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

[13] <удаление значения по умолчанию>::=
ALTER [COLUMN] <имя столбца> DROP DEFAULT
[14] <установка фильтра по умолчанию>::=
ALTER [COLUMN] <имя столбца> SET DEFAULT FILTER <идентификатор>
[15] <добавление диапазона автоинкрементных значений>::=
ALTER [COLUMN] <имя столбца>
ADD RANGE (<нижняя граница>:<верхняя граница>[, …])
[16] <установка каталога внешних файлов>::=
ALTER [COLUMN] <имя столбца> SET ROOT <символьный литерал>
[17] <установка последовательности значений>::=
ALTER [COLUMN] <имя столбца> <опция последовательности>[…]
[18] <опция последовательности>::=
[SET] [{START | RESTART} WITH <начальное значение>]
[SET] [INCREMENT BY <шаг>]
[SET] [MAXVALUE <верхняя граница> | NO MAXVALUE]
[SET] [MINVALUE <нижняя граница> | NO MINVALUE]
[SET] [CYCLE | NO CYCLE]
[19] <отмены/разрешения NULL-значений>::=
ALTER [COLUMN] <имя столбца> {ENABLE NULL | DISABLE NULL}
[20] <изменение уровня мандатного контроля доступа столбца>::=
SET COLUMN <имя столбца> LEVEL (<RAL>, <WAL>)
[21] <текущее имя столбца>::= <идентификатор>
[22] <новое имя столбца>::= <идентификатор>

Синтаксические правила переименования столбца таблицы

1) Переименование предопределенных столбцов таблицы (`ROWID`, `ROWTIME`, `DBROWTIME`) не допускается.

Общие правила переименования столбца таблицы

1) Переименование столбцов системных таблиц не допускается.
2) Операция переименования столбца не проверяет возможное использование заменяемого имени столбца в других объектах БД (представлениях, хранимых процедурах), поэтому, если такие объекты существуют, они перестанут работать.

Пример

```
create or replace table tst (i int, ch char(20));
alter table tst alter i rename to Id;
alter table tst alter column ch rename to "Наименование";
```

Синтаксические правила изменения длины столбца

1) Изменение длины столбца допускается только для строковых и байтовых типов данных.
2) Изменение длины столбца допустимо только в сторону увеличения в пределах допустимой размерности соответствующего типа данных.
3) Изменение длины столбцов типов данных `CHAR`, `BYTE`, `NCHAR` допускается только в том случае, если им не назначена опция умолчания (`DEFAULT`).
4) Не допускается изменение длины столбца, имеющего DEFAULT-значение, кроме столбцов переменной длины (`VARCHAR`, `VARBYTE`, `NCHAR VARYING`). В этом случае необходимо сначала удалить значение по умолчанию, потом изменить длину значений столбца и затем вернуть старое или установить новое DEFAULT-значение.

```
create or replace table tst (c char(10) default 'qwerty123');

alter table tst alter c drop default;
alter table tst alter c size 20;
alter table tst alter c set default 'qwerty123';
```

Общие правила изменения длины столбца

1) Изменение длины значений столбца допускается только для столбца, который не индексирован и не входит в составной индекс. Если столбец индексирован (или не индексирован, но объявлен как PRIMARY KEY, UNIQUE или FOREIGN KEY) или входит в составной индекс, то сначала надо удалить индекс (или удалить свойство PRIMARY KEY, UNIQUE или FOREIGN KEY), затем изменить размер значений столбца и снова проиндексировать столбец (или добавить удаленное свойство).
2) Запрещено выполнение оператора в том случае, если столбцы, расположенные в таблице после расширяемого столбца, входят в составные первичные, уникальные, внешние ключи и составные индексы.
3) Допускается как явное изменение длины значений столбца, так и косвенное (через присвоение столбцу нового типа данных, см. конструкцию <изменение типа данных столбца>).

```
create or replace table tst (i int);

alter table tst alter column i size 8;
```

эквивалентная конструкция:

```
alter table tst alter column i set data type bigint;
```

Синтаксические правила добавления столбцов

1) См. <определение столбца> в пункте «Создание таблицы».
2) Запрещено добавлять столбцы типа BLOB, EXTFILE.
3) Столбец с атрибутом AUTOROWID добавить в таблицу нельзя.

Общие правила добавления столбцов

1) Добавляемые столбцы присоединяются в конец таблицы (становятся ее последними столбцами).
2) Все существующие записи после операции ADD COLUMN всегда получают во всех добавленных столбцах NULL-значения.
3) Сразу после добавления нового столбца с default можно выполнить запрос

```
update имя_таблицы set имя_столбца=default;
```

тогда в столбце все значения примут значения по умолчанию.

Пример

```
create or replace table tst (i int);

alter table tst add column c char(10) default '*****';
alter table tst add (dt boolean, d date not null);
```


© Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

Синтаксические правила изменения типа данных столбца

1) Допустимые изменения типов данных столбца приведены в таблице 3.

Таблица 3. Допустимые изменения типов данных

Текущий тип данных	Допустимы преобразования	
char(n)	char(m), где m > n	varchar(m), где m >= n
varchar(n)	char(m), где m >= n	varchar(m), где m > n
nchar(n)	nchar(m), где m > n	nvarchar(m), где m >= n
nvarchar(n)	nchar(m), где m >= n	nvarchar(m), где m > n
byte(n)	byte(m), где m > n	varbyte(m), где m >= n
varbyte(n)	byte(m), где m >= n	varbyte(m), где m > n
smallint	int	bigint[1)]
int	bigint[1)]	
[1)]Среди текущих значений столбца не должно быть отрицательных значений.		

Общие правила изменения типа данных столбца

1) Параметр <тип данных> определяет новый <тип данных> столбца, например, вместо текущего <типа данных> int устанавливается <тип данных> bigint.
2) Операция <изменения типа данных столбца> не проверяет возможное использование заменяемого типа данных столбца в других объектах БД (например, в представлениях, триггерах), поэтому, если такие объекты существуют, то работа с ними может быть некорректной.
3) При попытке изменения типа данных столбца на точно такой же (совпадают и название типа и длина) выдается ошибка.

Пример

```
create or replace table tst (ch char(10));
alter table tst alter column ch set data type varchar(20);
alter table tst alter ch set data type varchar(30);
```

Синтаксические правила добавления CHECK-условия столбца

1) Синтаксис <логического выражения> приведен в описании команды CREATE TABLE.
2) Использование столбцов с атрибутом AUTOROWID, псевдостолбцов ROWID, ROWTIME и DBROWTIME и предиката [NOT] CONTAINS (см. документ «Полнотекстовый поиск в базе данных», пункт «Предикат полнотекстового поиска») в <логическом выражении> запрещено.

Общие правила добавления CHECK-условия столбца

1) Добавленное CHECK-условие присоединяется по логическому «И» ко всем ранее заданным условиям CHECK-условиям.
2) <Логическое выражение> в конструкции ADD CHECK может содержать LIKE-выражение, например:

```
alter table tab1 alter column ch add check(ch like 'aaa/_%' escape
 '/');
```

Пример

```
ALTER TABLE "Банк" ALTER COLUMN "Корр. счет" ADD CHECK
 (length("Корр. счет")=20);
```

Синтаксические правила установки значения по умолчанию

1) Параметр <значение> должен быть литералом, тип данных которого соответствует типу данных модифицируемого столбца.

```
create or replace table tst (i int, ch char(10));
insert into tst(i) values (100);
alter table tst alter column i set default 394006;
alter table tst alter column ch set default 'РЕЛЭКС';
insert into tst;
alter table tst alter column i set default (999);
insert into tst;
select * from tst;
I       CH
100     NULL
394006  РЕЛЭКС
999     РЕЛЭКС
```

2) Параметр <значение> не должен быть NULL-значением (хотя синтаксически NULL-значение допустимо, оно не будет восприниматься как значение по умолчанию). NULL-значение является стандартным значением для отсутствующих при операции добавления данных значений.
3) <Значимое выражение> должно содержать только константные значения и/или ссылки на столбцы, не имеющие значений по умолчанию.

```
create or replace table tst (i int, ch char(10));
alter table tst alter column i set default (25+sqrt(100));
insert into tst;
select * from tst;
I   CH
35  NULL

create or replace table tst (i int, ch char(10));
alter table tst alter column ch set default (to_char(i)+' руб.');
insert into tst(i) values(500);
select ch from tst;
CH
500 руб.
```

Общие правила установки значения по умолчанию

1) Конструкция устанавливает значение по умолчанию (если оно ранее не было установлено) либо меняет ранее установленное значение по умолчанию на новое значение. Ранее добавленные записи остаются без изменений.

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
Установка значения по умолчанию:
create or replace table tst (i int);
insert into tst(i) values (100);
alter table tst alter column i set default (800);
insert into tst;
select * from tst;
I
100
800

Изменение ранее установленного значения по умолчанию:
create or replace table tst (i int);
insert into tst(i) values (100);
alter table tst alter column i set default (800);
insert into tst;
alter table tst alter column i set default (999);
insert into tst;
select * from tst;
I
100
800
999
```

2) Столбцу присваивается стандартное значение (NULL-значение) если при добавлении записи в таблицу для столбца:
   - не задано значение;
   - он не имеет значения по умолчанию;
   - для него не установлено ограничение NOT NULL.

Синтаксические правила установки фильтра по умолчанию

1) <Идентификатор> задает имя внутреннего фильтра СУБД (конвертера, преобразующего символьные данные разных форматов в стандартный вид, пригодный для выполнения в нем полнотекстового поиска). Имя фильтра должно быть определено в системной таблице $$$FILTER (см. документ «Полнотекстовый поиск в базе данных», раздел «Фильтры»).

```
select $$$name from $$$filter;
$$$NAME
ASCTEXT2TEXT
ASCXML2TEXT
UNITEXT2TEXT
UNIXML2TEXT
DOCRTF2TEXT
NOTEXT2TEXT
ANSI2TEXT
KOI8R2TEXT
RUSTEXT2TEXT
```

```
UTF82TEXT
```

Общие правила установки фильтра по умолчанию

1) Указанный фильтр будет использоваться по умолчанию при индексировании данных данного столбца.

Пример

```
create or replace table full_search (doc char(4000) default filter
 "DOCRTF2TEXT", utf blob default filter "UTF82TEXT");

alter table full_search alter column doc set default filter
 "KOI8R2TEXT";
```

Синтаксические правила добавления диапазона автоинкрементых значений

1) Добавлять новые диапазоны можно только для столбцов с атрибутом AUTOINC RANGE.

Общие правила добавления диапазона автоинкрементых значений

1) Атрибут столбца `AUTOINC RANGE` задает список диапазонов допустимых значений. После исчерпания диапазонов значений добавление новых записей прекращается.

```
create or replace table tst (i int autoinc range
(1:3, 10:12));
insert into tst; // 1
insert into tst; // 2
insert into tst; // 3
insert into tst; // 10
insert into tst; // 11
insert into tst; // 12
insert into tst; // переполнение автоинкрементного столбца

select * from tst;
I
1
2
3
10
11
12
```

2) Добавляемые диапазоны границы должны быть расположены за последним существующим диапазоном.

```
create or replace table tst (i int autoinc range (10:100,
 500:1000, 2000:5000));
alter table tst alter column i add range (6000:7000);
create or replace table tst (i int autoinc range (1:2, 3:4));
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
alter table tst alter column i add range (5:6,10:15,100:500);
```

3) Вновь добавляемый диапазон не проверяется на отсутствие в столбце таблицы значений из этого диапазона.

4) Явно можно задавать любые значения вне границ диапазонов.

Синтаксические правила установки каталога внешних файлов

1) Конструкция применима только к столбцам типа данных EXTFILE.

2) <Символьный литерал> определяет каталог, в котором размещаются внешние файлы для данного столбца.

Общие правила установки каталога внешних файлов

1) Если при создании столбца с типом данных EXTFILE каталог внешних файлов не указан, то по умолчанию предполагается, что внешние файлы находятся в каталоге БД. При этом функция DEFAULT, предоставляющая информацию о каталогах внешних файлов, в этом случае возвращает NULL-значение. Однако функция FILESIZE, предоставляющая информацию о размере внешних файлов, ищет внешний файл в каталоге БД и, если находит, возвращает его длину.

```
create or replace table tst ("Слова" extfile, "Музыка" extfile
 root 'd:\Program Files\Music\shanson');

insert into tst("Музыка", "Слова") values (ef('Чубчик.mp4'),
 ef('Чубчик.txt'));

select default ("Музыка"), default ("Слова") from tst;
d:\Program Files\Music\shanson  NULL

select filesize ("Музыка"), filesize ("Слова") from tst;
325482353 669
```

2) Новый каталог внешних файлов заменяет ранее установленный (старый) каталог. Файлы, хранящиеся в старом каталоге внешних файлов, в новый каталог не перемещаются. Чтобы иметь доступ к ним, необходимо вручную переместить их в новый каталог.

```
create or replace table tst ("Слова" extfile, "Музыка" extfile
 root 'd:\Program Files\Music\shanson');

insert into tst("Музыка", "Слова") values (ef('music.mp4'),
 ef('readme.txt'));

select default ("Музыка"), default ("Слова") from tst;
  d:\Program Files\Music\shanson  NULL
alter table tst alter column "Слова" set root 'd:\Program Files
\Text\shanson';
select default ("Музыка"), default ("Слова") from tst;

d:\Program Files\Music\shanson  d:\Program Files\Text\shanson
```

Синтаксические правила установки последовательности значений

1) Конструкция <установка последовательности значений> применима только к <идентификационным столбцам>.

```
create or replace table tst (i int generated always as identity
 (start with 1));
insert into tst;
select * from tst; // 1
alter table tst alter column i set start with 99;
insert into tst;
select * from tst; // 99
I
1
99
```

2) Модификатор SET можно не указывать.

```
create or replace table tst (i int GENERATED ALWAYS AS IDENTITY
 (start with 1));

// все опции последовательности заданы с модификатором SET
alter table tst alter column i set restart with 10 set increment
 by 10 set minvalue 10 set maxvalue 100000 set no cycle;

// все опции последовательности заданы без модификатора SET
alter table tst alter column i restart with 10 increment by 10
 minvalue 10 maxvalue 100000 no cycle;

// с модификатором SET  заданы некоторые опции последовательности
alter table tst alter column i restart with 30 set increment by 10
 minvalue 10 set maxvalue 100000 no cycle;
```

Общие правила установки последовательности значений

1) При указании новых параметров <идентификационного столбца> ранее введенные значения столбца не меняются.

```
create or replace table tst (i int GENERATED ALWAYS AS IDENTITY
 (start with 1));
insert into tst;
insert into tst;
select * from tst;
I
1
2

alter table tst alter column i set restart with 99 set increment
 by 100;

insert into tst;
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
insert into tst;
select * from tst;
I
1
2
99
199
```

Общие правила отмены/разрешения NULL-значений

1) <Разрешение NULL-значений> допускается для столбцов с атрибутом NOT NULL.

```
create or replace table tst (i int not null);
insert into tst; // ошибка
alter table tst alter column i enable null;
insert into tst; // нормальное завершение
```

2) <Запрет NULL-значений> допускается для столбцов с атрибутом NULL и только в том случае, если к моменту запрета NULL-значений в столбце нет NULL-значений.

```
create or replace table tst (i int null);
alter table tst alter column i disable null; //нормальное
 завершение

create or replace table tst (i int null);
insert into tst;
select * from tst;
I
NULL
alter table tst alter column i disable null; // ошибка
```

Общие правила изменения уровня мандатного контроля доступа столбца

1) Устанавливаемые уровни доступа должны быть декларированы в БД.

Пример

```
select * from $$$level;
$$$ID $$$NAME    $$$DESCR
1     НЕСЕКРЕТНО
2     ДСП
3     СЕКРЕТНО

create or replace user "USER1"
  identified by '12345678' level ("СЕКРЕТНО","СЕКРЕТНО");

grant dba to USER1;
username "USER1"/12345678
create or replace table tst (i int);
alter table tst set column i level ("СЕКРЕТНО", "СЕКРЕТНО");
```

```
insert into tst;
insert into tst(i) values(100);
insert into tst(i##"СЕКРЕТНО"#"СЕКРЕТНО") values(200);
insert into tst(i##3#3) values(300);
select * from tst;

 I
 -
|           |
|         100|
|         200|
|         300|

select security(i, 'R'), security(i, 'W')  from tst;

|           3|           3|
|           3|           3|
|           3|           3|
|           3|           3|
```

Синтаксические правила удаления столбцов

1) <Имя столбца> не должно ссылаться на псевдостолбец таблицы (ROWID, ROWTIME и т. п.).

Общие правила удаления столбцов

1) Для удаления столбцов необходимы привилегии, аналогичные привилегиям для удаления всей таблицы (см. конструкцию DROP TABLE).
2) При удалении столбца удаляются сам столбец и все содержащиеся в нем данные.
3) Столбец с указанным <именем столбца> не должен быть:
   - единственным столбцом таблицы;
   - первичным ключом;
   - внешним ключом;
   - индексированным столбцом;
   - элементом составного индекса;
   - имеющим CHECK-условие.
4) Для удаления столбца с меткой мандатного доступа необходимо иметь соответствующий уровень доступа.
5) После того как столбец удален, его невозможно восстановить (например, по команде ROLLBACK). В этом случае необходимо создавать столбец заново и загружать в него удаленные данные (если они предварительно были сохранены).
6) При удалении столбца типа EXTFILE удаляются только ссылки на внешние файлы, сами файлы не удаляются.
7) Запрещено удаление столбцов:
   - в БД с мандатной защитой (кроме случая, когда таблица не содержит записей);

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

- из системной таблицы;
- из глобальной временной таблицы, которая находится в чьем-либо использовании в момент удаления;
- из таблицы, которая содержит расширенные описания файлов;
- из таблиц-представлений, таблиц-синонимов или таблиц «в памяти»;

Примечание
Если необходимо удалить столбец из таблицы «в памяти» ее необходимо предварительно преобразовать в базовую таблицу.

- если для столбца (столбцов) добавлен комментарий.

Примечание
Для удаления столбца (столбцов) с комментарием необходимо использовать опцию CASCADE.

- всей таблицы.

8) Разрешено удаление столбцов при указании модификатора CASCADE:
   - входящих в составной индекс;
   - являющихся первичным или уникальным ключом;
   - являющихся внешним ключом (направленным и «к» и «от» таблицы);
   - входящим в триггер с условием срабатывания «update of <имя столбца>»;
   - имеющим ограничение целостности CHECK;
   - имеющим именованный одностолбцовый индекс.

   В этом случае происходит автоматическое удаление всех атрибутов столбца и/или установленных для него ограничений и удаление любых столбцов той же таблицы (в которой находится удаляемый столбец) с номерами, превышающими минимальный номер в наборе удаляемых столбцов, которые мешают удалению.

Примечание
При обнаружении перечисленных конструкций, относящихся к удаляемым столбцам, и при отсутствии модификатора CASCADE выдается код завершения 1530 «Столбец не может быть удален (возможно, он имеет зависимости)».

9) Одностолбцовые неименованные и фразовые индексы у удаляемых столбцов удаляются автоматически.

Примеры

```
create or replace table tst(i int, ch char(10), d float);
alter table tst drop column i;
alter table tst drop d;

create or replace table tst(i int, ch char(10), d float);
alter table tst drop (i, d);
```

Синтаксические правила удаления значения по умолчанию

1) Конструкция применима только для столбцов с атрибутом GENERATED BY DEFAULT AS или DEFAULT.

```
create or replace table tst (i int default 0);
! OK
alter table tst alter column i drop default;
! Нет значения по умолчанию
alter table tst alter column i drop default;
```

Общие правила удаления значения по умолчанию

1) Конструкция отменяет ранее установленное значение по умолчанию.

Пример

```
alter table tst alter column i drop default;
```

Синтаксические правила удаления CHECK-условий столбца

1) <Имя столбца> должно ссылаться на столбец, у которого есть хотя бы одно CHECK-условие.

Общие правила удаления CHECK-условий столбца

1) Удалить можно только сразу все CHECK-условия, привязанные к столбцу. Выборочное удаление CHECK-условий не поддерживается.

Пример

```
alter table "Банк" alter column "Корр. счет" drop check;
```

Временная модификация таблицы (ALTER TABLE ENABLE/DISABLE)

Функция

Определение оператора временного изменения атрибутов базовой таблицы.

Спецификация

[1] <временный запрет действия первичного ключа>::=
ALTER TABLE [<имя схемы>.]<имя таблицы> DISABLE PRIMARY KEY

[2] <временный запрет действия уникального ключа>::=
ALTER TABLE [<имя схемы>.]<имя таблицы> DISABLE UNIQUE (<список столбцов>)

[3] <временный запрет действия внешнего ключа>::=
ALTER TABLE [<имя схемы>.]<имя таблицы> DISABLE FOREIGN KEY (<список столбцов>)

[4] <временное отключение всех триггеров таблицы>::=
ALTER TABLE [<имя схемы>.]<имя таблицы> DISABLE ALL TRIGGERS

[5] <отмена временного запрета действия первичного ключа>::=
ALTER TABLE [<имя схемы>.]<имя таблицы> ENABLE PRIMARY KEY

[6] <отмена временного запрета действия уникального ключа>::=
ALTER TABLE [<имя схемы>.]<имя таблицы> ENABLE UNIQUE (<список столбцов>)

[7] <отмена временного запрета действия внешнего ключа>::=
ALTER TABLE [<имя схемы>.]<имя таблицы> ENABLE FOREIGN KEY (<список столбцов>)

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

[8] <отмена временного отключения всех триггеров таблицы> : : =
ALTER TABLE [<имя схемы>.]<имя таблицы> ENABLE ALL TRIGGERS
[9] <список столбцов> : : = <имя столбца>[, <имя столбца> …]

Синтаксические правила

1) <Имя таблицы> должно быть именем базовой таблицы или синонимом, указывающим на базовую таблицу (прямо или через цепочку синонимов).

Общие правила

1) Временный запрет действия ключа (PRIMARY KEY, UNIQUE или FOREIGN KEY) должен устанавливаться в случаях, когда соответствующее ограничение ссылочной целостности нарушено для некоторых записей таблицы в результате логического повреждения целостности БД.
2) На протяжении действия введенного запрета количество нарушений целостности не должно увеличиваться.
3) Перед отменой временного запрета действия ключей имеющиеся в БД нарушения целостности должны быть устранены, при этом новых нарушений в процессе устранения появляться не должно.
4) На период временного запрета действия ключей запрещаются операции над первичным/уникальным ключом (когда запрещен сам этот ключ или какой-либо ссылающийся на него внешний ключ).
5) Если нет ссылающихся на него внешних ключей, то разрешены любые действия. Если такие ключи есть, то:
   - добавление разрешается (при этом уникальность значения по-прежнему проверяется при добавлении в индекс);
   - удаление разрешается только для значений-дубликатов первичного (уникального) ключа (а также NULL-значений для уникального ключа);
   - обновление разрешается только для значений-дубликатов первичного (уникального) ключа, а также NULL-значений для уникального ключа. Уникальность нового значения проверяется при добавлении в индекс;
   - проверки ссылочной целостности не выполняются (это касается удаления и обновления значений-дубликатов первичного (уникального) ключа);
   - запрещается добавление новых внешних ключей, ссылающихся на запрещенный первичный (уникальный) ключ.
6) На период временного запрета действия ключей запрещаются операции над внешним ключом (когда запрещен сам этот ключ либо первичный/уникальный ключ, на который он ссылается):
   - удаление разрешается;
   - разрешается добавление только NULL-значений;
   - разрешается обновление только в ссылках на NULL-значения.
7) Правом на временное отключение/включение всех триггеров таблицы обладает ее владелец с уровнем прав доступа к БД – RESOURCE.

Пример

Пусть есть первичный (уникальный) ключ, на который ссылается множество внешних ключей. В момент отказа компьютера испорчен индекс на первичный (уникальный)

ключ, в результате чего в столбце оказались дубликаты первичного (уникального) ключа. Перестроить этот индекс, не снимая атрибута «первичный (уникальный) ключ», нельзя, так как ядро СУБД обнаружит отсутствие уникальности значений ключа. Чтобы снять атрибут «первичный (уникальный) ключ», следует предварительно удалить все записи, ссылающиеся на этот ключ.

Использование команд временной модификации таблицы позволяет минимизировать количество действий, требуемое для восстановления БД.

Восстановление таблицы (REBUILD TABLE)

Функция

Восстановление (насколько возможно) таблицы с нарушенной структурой данных.

Спецификация

[1] <восстановление таблицы> : : =
REBUILD TABLE [<имя схемы>.]<имя таблицы>
[WITH {INDEXES | INDEX FILES | ALL FILES} | INDEXES ONLY]
[IGNORE LOCK] [WAIT | NOWAIT]

Синтаксические правила

1) <Имя таблицы> должно ссылаться на базовую таблицу.
2) <Имя таблицы> не должно ссылаться на системную таблицу.
3) По умолчанию используется опция WAIT.
4) Правом на восстановление таблицы обладает ее владелец или любой другой пользователь с привилегией на модификацию структуры таблицы (ALTER) и уровнем прав доступа к БД RESOURCE.

Общие правила

1) Исходной информацией в процессе восстановления таблицы является файл данных восстанавливаемой таблицы. На его основе происходит построение нового конвертера таблицы.
2) После построения нового конвертера дальнейшие действия зависят от заданного модификатора команды:
   - WITH INDEXES – выполняется перестройка индексов;
   - WITH INDEX FILES – выполняется перестройка индексов и индексных файлов;
   - WITH ALL FILES – выполняется перестройка индексов, индексных файлов и файлов данных (в файлах данных очищаются страницы, в которых длина записи выходит за обозначенный размер страницы, длина записи нечетная и другие случаи).
3) На время восстановления таблицы ядро СУБД блокирует доступ к ней других пользователей. Если блокировка восстанавливаемой таблицы невозможна (есть активные каналы, которые подали связанный с таблицей запрос и не завершили его), необходимо использовать модификатор IGNORE LOCK.
4) Опция INDEXES ONLY заставляет перестраивать только простые и составные индексы таблицы (за исключением фразовых индексов) без восстановления соответствия адресного конвертера файлам данных (например, если утилита testdb диагностировала для этой таблицы только проблемы, относящиеся к индексам). Опция полезна тем, что для её применения не требуются знания о структуре и имени поврежденного индекса.

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

Сжатие таблицы (PRESS TABLE)

Функция

Сжатие (уплотнение) таблицы и упорядочивание её ROWID после многочисленных операций удаления из нее записей.

Спецификация

[1] <сжатие таблицы> : : =
PRESS TABLE [<имя схемы>.]<имя таблицы>
[WITH TRUNCATE] [WAIT | NOWAIT]

Синтаксические правила

1) <Имя таблицы> должно ссылаться на базовую таблицу.
2) <Имя таблицы> не должно ссылаться на системную таблицу.
3) Правом на сжатие таблицы обладает ее владелец или любой другой пользователь с привилегией на модификацию структуры таблицы (ALTER) и уровнем прав доступа к БД RESOURCE.

Общие правила

1) Если таблица имеет столбец AUTOROWID, операция <сжатие таблицы> не допускается.
2) Если не задан модификатор WAIT (NOWAIT), по умолчанию используется WAIT.
3) После удаления всех записей и сжатия таблицы для столбцов AUTOINC INT и AUTOINC SMALLINT значение начинает наращиваться от заданного как INITIAL при создании таблицы, а для AUTOINC BIGINT – от 1.
4) После сжатия таблицы выполняется перестройка индекса.
5) Если задан модификатор WITH TRUNCATE сжатие пространства БД выполняется по следующим правилам:
   - для файлов данных происходит перенос данных (в пределах своего файла) в начало файла с последующим усечением освободившегося пространства;
   - для файлов индексов происходит удаление индексов путём усечения файлов индексов до минимального размера, затем пересоздание индексов;
   - файлы с BLOB-данными не сжимаются.

Тестирование таблицы (TEST TABLE)

Функция

Проверка структуры таблицы.

Спецификация

[1] <тестирование таблицы> : : =
TEST TABLE [<имя схемы>.]<имя таблицы> [<операция тестирования>] [FULL]
[WAIT | NOWAIT] [NO LONG {QUANT | QUANTUM}]
[2] <операция тестирования> : : =
<тестирование одностолбцового индекса>
| <тестирование именованного индекса>
| <тестирование элементов структуры>
[3] <тестирование одностолбцового индекса> : : =
COLUMN <имя столбца> INDEX

[4] <тестирование именованного индекса>::=
INDEX {<имя индекса> | <имя столбца>}
[5] <тестирование элементов структуры>::=
(<элемент структуры>[, ...])
[6] <элемент структуры>::=
DESCRIPTION | BITMAP | DATA | INDEX | BLOB | INTEGRITY

Синтаксические правила

1) QUANTUM является синонимом QUANT.
2) При указании опции NO LONG QUANT тестирование таблицы выполняется в фоновом режиме (даже при отсутствии других активных процессов в ядре СУБД). Если опция не задана, по умолчанию используется конкурирующий (квантуемый с другими процессами обработки данных) режим тестирования.
3) Модификатор WAIT заставляет СУБД ЛИНТЕР ожидать завершения процедуры тестирования.
4) Если модификатор WAIT (NOWAIT) не задан, по умолчанию действует WAIT.
5) При указании опции FULL без указания <операции тестирования> выполняется полное тестирование индексов указанной таблицы (соответствует уровню проверки 3 утилиты testdb). Также включает проверки отсутствия дубликатов в AUTOINC-столбцах (если они не PRIMARY KEY и не UNIQUE) – по умолчанию не выполняются.

   Без опции FULL тестирование индексов указанной таблицы соответствует уровню проверки 2 утилиты testdb.
6) При указании конструкции COLUMN <имя столбца> INDEX FULL выполняется полное тестирование индекса на указанный столбец (соответствует уровню проверки 3 утилиты testdb), без FULL соответствует уровню проверки 2 утилиты testdb.
7) При указании конструкции INDEX {<имя индекса> | <имя столбца>} FULL выполняется полное тестирование индекса с указанным именем (соответствует уровню проверки 3 утилиты testdb), без FULL соответствует уровню проверки 2 утилиты testdb.
8) Опцию FULL нельзя указывать для тестирования <элемента структуры>.
9) <Имя таблицы> должно ссылаться на базовую таблицу или представление.
10) <Имя таблицы> не должно ссылаться на системную таблицу.
11) <Имя таблицы> не должно ссылаться на виртуальную таблицу ($$$CHAN, $$$SYSINFO, $$$EVENTS_INFO).
12) Правом на тестирование таблицы обладает ее владелец или любой другой пользователь с привилегией на модификацию структуры таблицы (ALTER) и уровнем прав доступа к БД RESOURCE.
13) На время тестирования таблица блокируется в режиме SHARE.

```
test table auto column personid index;
```

14) Опции тестирования элементов структуры:
    - DESCRIPTION – тестирование системного описания таблицы;
    - BITMAP – тестирование битовых карт файлов таблицы;
    - DATA – тестирование файла данных;
    - INDEX – тестирование файла индексов;
    - BLOB – тестирование BLOB-файлов;
    - INTEGRITY – тестирование логической целостности;

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

- если конкретный <элемент структуры> тестирования не задан, по умолчанию тестируются все элементы таблицы:

```
test table auto;
```

эквивалентно:

```
test table auto (description, bitmap, data, index, blob,
 integrity);
```

- Результат выполнения тестирования:
  а) при отсутствии ошибок в физической и логической структуре таблицы диагностические сообщения выведены не будут;
  б) при наличии ошибок в физической и логической структуре таблицы будет выведена диагностическая информация в формате:

```
|SEVERITY|TYPE|DESCRIPTION|
```

Например,

```
|SEVERITY | TYPE                                                    |
 DESCRIPTION
|WARNING  | Table #2503 data file #1 bitmap, page #1, bit #6216 |
 Bit is set for page which is not filled, has only 0 bytes
```

Восстановление таблицы необходимо выполнять с помощью утилиты testdb (см. документ «Тестирование базы данных», пункт «Исправление БД»).

Предварительная загрузка таблицы (TEST TABLE PRELOAD)

Функция

Предварительная загрузка таблицы с диска в пул ядра СУБД.

Спецификация

[1] <предварительная загрузка таблицы>::=
TEST TABLE [<имя схемы>.]<имя таблицы>
PRELOAD [WITH BLOBS] [WAIT | NOWAIT] [NO LONG {QUANT | QUANTUM}]

Синтаксические правила

1) QUANTUM является синонимом QUANT.
2) При указании опции NO LONG QUANT предварительная загрузка таблицы выполняется в фоновом режиме (даже при отсутствии других активных процессов в ядре СУБД). Если опция не задана, по умолчанию используется конкурирующий (квантуемый с другими процессами обработки данных) режим предварительной загрузки.
3) Опция WAIT заставляет СУБД ЛИНТЕР ожидать завершения предварительной загрузки таблицы.
4) Если модификатор WAIT (NOWAIT) не задан, по умолчанию действует WAIT.
5) Если задана опция WITH BLOBS, то в пул ядра СУБД загружаются и страницы файлов с BLOB-данными (при наличии таких файлов). По умолчанию страницы BLOB-файлов не загружаются.

Общие правила

1) Команда выполняет предварительную загрузку страниц файла данных и файла индексов указанной таблицы в пул ядра СУБД (подробности см. в документе «Архитектура СУБД», раздел «Предварительная загрузка таблиц»).

Примечание
Для выполнения предварительной загрузки таблицы необходимы права DBA.

2) Предварительная загрузка таблицы всегда выполняется успешно (если пул ядра СУБД в момент загрузки полностью занят, то ранее загруженные в него страницы других таблиц будут вытесняться).

Усечение таблицы (TRUNCATE TABLE)

Функция

Ускоренная очистка таблицы и усечение файлов таблиц.

Спецификация

[1] <усечение таблицы>::=
TRUNCATE TABLE [<имя схемы>.]<имя таблицы>
[DROP STORAGE | REUSE STORAGE | REUSE FILES]
[<новые параметры таблицы>] [WAIT | NOWAIT]
[2] <новые параметры таблицы>::=
INDEXFILES <описание файлов>
| DATAFILES <описание файлов>
| BLOBFILES <описание файлов>
| MAXROW <количество записей>
| MAXROWID <количество записей>
| PCTFILL <процент заполнения>
| PCTFREE <процент заполнения>
| BLOBPCT <процент заполнения>
[3] <описание файлов>::=
<количество файлов> [(<описатель файла> [, …])]
[4] <процент заполнения>::= <беззнаковое целое>

Синтаксические правила

1) <Имя таблицы> должно ссылаться на базовую таблицу.

Примечание
Команда TRUNCATE TABLE для глобальных временных таблиц игнорируется.

2) <Имя таблицы> не должно ссылаться на системную таблицу.
3) Правом на усечение таблицы обладает ее владелец или любой другой пользователь с привилегиями на модификацию структуры таблицы (ALTER), на удаление данных (DELETE) и уровнем прав доступа к БД RESOURCE.
4) Слова STORAGE, REUSE и FILES являются ключевыми только в команде TRUNCATE.
5) Описание параметров INDEXFILES, DATAFILES, BLOBFILES, MAXROWID, MAXROW, PCTFILL, PCTFREE и BLOBPCT см. в пункте «Создание таблицы».
6) Если модификатор WAIT (NOWAIT) не задан, по умолчанию действует WAIT.

Общие правила

1) В БД с секретностью усекаемая таблица не должна содержать записей.

2) При усечении таблицы проверяется наличие триггеров на таблицу: если есть триггер на удаление из этой таблицы, то TRUNCATE не выполняется.
3) На усекаемую таблицу не должно быть ссылок из непустых таблиц БД.
4) Указания по размерам создаваемых файлов после усечения таблицы:
   - REUSE STORAGE – использовать те размеры файлов, которые были до усечения;
   - REUSE FILES – использовать минимальные размеры файлов;
   - DROP STORAGE – оставить по одному файлу минимального размера каждого типа.

   По умолчанию используется опция DROP STORAGE.
5) Выполняется выравнивание значения MAXROWID на 1022. Т.е. если указать только значение MAXROW (например, 2000) и не указывать MAXROWID, то значение MAXROWID будет браться из MAXROW с выравниванием (т.е. оно будет равно 2044). Если, наоборот, указать MAXROWID (например, 2000), но не указывать MAXROW, то сначала будет выровнен MAXROWID (до значения 2044), а затем это значение будет присвоено MAXROW (2044).
6) Параметры INDEXFILES, DATAFILES, BLOBFILES, если они заданы явно, имеют приоритет над опциями DROP STORAGE, REUSE STORAGE, REUSE FILES.
7) Команда является DDL-операцией, то есть перед ее обработкой выполняется команда COMMIT (фиксация в БД всех накопленных изменений). Откат операции, которая уже началась по данной команде, не возможен.
8) Вся таблица блокируется на запись до окончания операции.
9) Удаляются все файлы таблицы: данных, индексов, BLOB-данных.
10) Новые файлы таблицы создаются по минимальному количеству (один файл данных; один файл индексов, если нет ссылок из описаний индексов на несколько индексных файлов; один файл BLOB-данных, если есть BLOB-столбцы или возможны длинные записи) и минимальному размеру: одна страница (4 Кбайта) битовой карты и по одной странице файла данных, индексов, BLOB-данных.



Примечание
Команда TRUNCATE TABLE не эквивалентна последовательности команд DROP TABLE и CREATE TABLE с новыми параметрами таблицы, т.к. при использовании TRUNCATE TABLE все системные ссылки на усекаемую таблицу остаются неизменными. В ином случае ссылки на эту таблицу из других таблиц, правила репликации, права доступа к таблице и т.д. будут потеряны.

Пример

```
create or replace table test(i int);
insert into test values(1);
insert into test values(2);
insert into test values(3);
select count(*) from test;

|          3|

truncate table test;
select count(*)from test;

|          0|
```

Таблицы «в памяти»

Создание таблицы «в памяти» (CREATE TABLE IN-MEMORY)

Функция

Определение оператора создания таблицы «в памяти».



Примечание
По умолчанию СУБД ЛИНТЕР не предусматривает работу с таблицами «в памяти». Для поддержки этой функциональности необходимо:

- сконфигурировать СУБД с помощью утилиты `gendb` (см. документ «Создание и конфигурирование базы данных», пункт «Установка параметров конфигурирования», команды SET IN-MEMORY TABLES, SET IN-MEMORY COLUMNS, SET IN-MEMORY FILES);
- при каждом запуске ядра СУБД ЛИНТЕР указывать ключ `/INMEMPOOL=<размер>`, где `<размер>` задает максимально допустимое количество страниц в пуле страниц ядра СУБД ЛИНТЕР, выделяемых для размещения таблиц «в памяти». Если этот ключ не задан, то использование таблиц «в памяти» запрещено.
- в случае активной работы с таблицами «в памяти» рекомендуется увеличить размер памяти канала, выполнив команду

```
ALTER DATABASE SET CHANNEL MEMORY LIMIT 1048576;
```

Спецификация

[1] <определение таблицы «в памяти»>::=
CREATE [IF NOT EXISTS | OR REPLACE]
TABLE [<имя схемы>.]<имя таблицы>(<атрибуты таблицы>)
IN-MEMORY [[NO] AUTOSAVE] [[NO] AUTOLOAD]

Синтаксические правила

1) Параметры (`<атрибуты таблицы>`) аналогичны параметрам создания базовой таблицы (см. конструкцию `<создание таблицы>`), за исключением атрибутов, информация о которых хранится не только в описании таблицы и ее столбцов, но и в других компонентах БД. Например: значения по умолчанию, диапазоны значений столбцов, правила ограничения целостности (`CHECK`), внешние ключи, фразовые индексы и т.п.
2) Для таблиц «в памяти» запрещено создавать триггеры.
3) Для таблиц «в памяти» запрещено использование ссылочной целостности.
4) `<Имя таблицы>` должно быть отлично от имени любой другой таблицы (базовой или «в памяти»), представления и синонима в пределах <имени схемы> или текущей схемы.

```
create table in_mem (i int unique) in-memory;
```

5) Опция `AUTOSAVE` задаёт автоматическое сохранение таблицы «в памяти» на диск при завершении работы СУБД. Значение по умолчанию – `NO AUTOSAVE`.
6) Опция `AUTOLOAD` задаёт автоматическое активизирование таблицы «в памяти» при старте СУБД ЛИНТЕР (в противном случае таблица «в памяти» активизируется при первом обращении к ней). Значение по умолчанию – `NO AUTOLOAD`. Обращение к

таблице «в памяти», которая не была активизирована при старте СУБД или командой RESTORE (например, из-за недостаточности ресурсов) вызывает ошибку.

7) Опция OR REPLACE заставляет удалять существующую в БД таблицу (со всеми её данными) и создавать её под тем же именем, но с указанными параметрами.
8) Опция IF NOT EXISTS отменяет выполнение оператора, если указанная таблица уже существует в БД.
9) Одновременное использование опций IF NOT EXISTS и OR REPLACE запрещено.

Общие правила

1) При старте ядра СУБД выполняется активизация всех таблиц «в памяти», созданных с атрибутом AUTOLOAD.
2) При останове ядра СУБД выполняется сохранение всех таблиц «в памяти», созданных с атрибутом AUTOSAVE.

Сохранение таблицы «в памяти» (SAVE TABLE)

Функция

Определение оператора сохранения таблицы «в памяти».

Спецификация

[1] <сохранение таблицы «в памяти»>::=
SAVE TABLE [<имя схемы>.]<имя таблицы>

Синтаксические правила

1) <Имя таблицы> должно ссылаться на таблицу «в памяти».

Общие правила

1) Вся информация о таблице сохраняется на диск (в системные таблицы БД, в файлы данных, индексные файлы и т.п.). При каждом сохранении таблицы «в памяти» её текущая версия заменяет ранее сохраненную.
2) Если таблица «в памяти» создана без опции AUTOSAVE, на диске сохраняются только те данные, которые были в ней на момент выполнения оператора SAVE TABLE.

Пример

```
create or replace table in_mem (i int unique) in-memory no
 autosave;
insert into in_mem(i) values(1);
insert into in_mem(i) values(2);
insert into in_mem(i) values(3);

// сохраняем промежуточный вариант таблицы
save table in_mem;

// продолжаем работать с таблицей «в памяти»
insert into in_mem(i) values(4);
insert into in_mem(i) values(5);
select * from in_mem;
```

```
1
2
3
4
5
// завершаем работу  ядра СУБД.
…
// запускаем  ядро СУБД.
select * from in_mem;
1
2
3
```

Восстановление таблицы «в памяти» (RESTORE TABLE)

Функция

Определение оператора восстановления таблицы «в памяти».

Спецификация

[1] <восстановление таблицы «в памяти»>::=
RESTORE TABLE [<имя схемы>.]<имя таблицы>

Синтаксические правила

1) <Имя таблицы> должно ссылаться на сохранённую ранее таблицу «в памяти».

Общие правила

1) При каждом восстановлении таблицы «в памяти» она заменяет ранее восстановленную таблицу.

Пример

```
create or replace table in_mem (i int unique) in-memory no
 autosave;
insert into in_mem(i) values(1);
insert into in_mem(i) values(2);
insert into in_mem(i) values(3);
select * from in_mem;
1
2
3
// сохраняем в контрольной точке текущую версию таблицы

save table in_mem;

// выполняем модификацию таблицы

update in_mem set i=20 where rowid=2;
update in_mem set i=30 where rowid=3;
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
select * from in_mem;
1
20
30

// модификация признана ошибочной, делаем «откат»  к сохранённой
 версии // таблицы

restore table in_mem;
select * from in_mem;
1
2
3
```

Модификация таблицы «в памяти»

Функция

Определение оператора модификации таблицы «в памяти».

Спецификация

[1] <модификация таблицы «в памяти»>::=
ALTER TABLE [<имя схемы>.]<имя таблицы>
[NOT] [IN-MEMORY] [[NO] AUTOSAVE] [[NO] AUTOLOAD]

Синтаксические правила

1) Опция `IN-MEMORY` применима только к базовой таблице, то есть `<имя таблицы>` должно ссылаться на базовую таблицу.
2) Конструкция `NOT IN-MEMORY` применима только к таблице «в памяти», то есть `<имя таблицы>` должно ссылаться на таблицу «в памяти».
3) Отсутствие `NOT IN-MEMORY` или `IN-MEMORY` равнозначно конструкции `NOT IN-MEMORY`.

Общие правила

1) Если задана опция `IN-MEMORY`, базовая таблица становится таблицей «в памяти» и активизируется. Если одновременно заданы опции `AUTOLOAD` и/или `AUTOSAVE`, то они применяются к созданной таблице «в памяти».
2) Если задана опция `NOT IN-MEMORY`, таблица «в памяти» сохраняется на диск и становится базовой таблицей.
3) Перевод таблицы из категории «в памяти» в категорию «базовая» выполняется только для предварительно сохранённой таблицы. Изменения в таблице «в памяти», выполненные после её сохранения, но перед выполнением оператора `ALTER TABLE`, будут утеряны.

```
create or replace table tst (i int unique) in-memory;
insert into tst(i) values(1);
insert into tst(i) values(2);
insert into tst(i) values(3);
save table tst;
```

```
// после модификации таблицы эти изменения будут утеряны
insert into tst(i) values(4);
insert into tst(i) values(5);

alter table tst not in-memory;

// эти изменения будут сохранены в базовой таблице
update tst set i=10 where rowid=1;
```

4) Базовая таблица, которая преобразуется в таблицу «в памяти», не должна иметь препятствующих этой операции ограничений (например, фразовых индексов, ссылок на себя).

5) Если опция IN-NEMORY не задана, и указанная таблица уже является таблицей «в памяти», то указание опции AUTOLOAD и/или AUTOSAVE устанавливает для таблицы «в памяти» эти свойства.

Пример

```
// создаём базовую таблицу
create or replace table tst (i int unique);
insert into tst(i) values(1);
insert into tst(i) values(2);
insert into tst(i) values(3);
select * from tst;
1
2
3
// пробуем применить к базовой таблице операцию сохранения
save table tst;
// получаем код завершения о недопустимости данной операции
// заменяем базовую таблицу на таблицу «в памяти»
alter table tst in-memory;
update tst set i=10 where rowid=1;
select * from tst;
10
2
3
// пробуем применить к модифицированной таблице операцию
 сохранения
save table tst;
// получаем нормальный код завершения
```

Представления

Создание представления (CREATE VIEW)

Функция

Определение представления.

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

Спецификация

[1] <создание представления>::=
CREATE [IF NOT EXISTS | OR REPLACE] [MATERIALIZED]
VIEW [<имя схемы>.] <имя представления>
[(<имя столбца> [, …])] AS <запрос выборки>
[WITH [CASCADED | LOCAL] CHECK OPTION]

[2] <имя представления>::= <идентификатор>

Синтаксические правила

1) <Имя представления> должно быть уникальным среди имен таблиц, представлений и их синонимов в рамках <имя схемы> или текущей схемы, при отсутствии <имя схемы>.
2) Опция OR REPLACE заставляет удалять существующее в БД представление и создавать его под тем же именем, но с указанными параметрами.
3) Опция IF NOT EXISTS отменяет выполнение оператора, если указанное представление уже существует в БД.
4) Одновременное использование опций IF NOT EXISTS и OR REPLACE запрещено.
5) Опция MATERIALIZED заставляет создавать «материализованное» представление.
6) Если <запрос выборки> обновляемый, то и представление считается обновляемой таблицей. В противном случае – это считываемая таблица.
7) При отсутствии спецификации столбцов представления они получат имена столбцов из <запроса выборки>. Если некоторые столбцы <запроса выборки> имеют одинаковое имя или считаются неименованными, то список имен столбцов представления должен быть задан.

```
CREATE VIEW Owner_Auto AS
  SELECT Name,FirstNam,SerialNo,Make,Model
FROM Person P,Auto A WHERE P.PersonID=A.PersonID;

select * from Owner_Auto;

 NAME     FIRSTNAM         SERIALNO           MAKE     MODEL
|WAGNER  |PHIL            |J003180026998205  |ALPINE  |A-310  |
…
```

8) Имена в списке (<имя столбца>[, …]) должны быть уникальны.
9) Число столбцов в списке (<имя столбца>[, …]) должно быть равно количеству столбцов в <запросе выборки>.

```
CREATE VIEW Black_Auto(PersonID,Make,Model,SerialNo) AS
SELECT PersonID,Make,Model,SerialNo
  FROM Auto WHERE Color ='BLACK';
```

10) Разрешено опускать явное преобразование типа при выборке NULL-значения как столбца. По умолчанию тип данных создаваемого столбца будет int. Три нижеследующих конструкции эквивалентны:

```
create or replace view vtst (ID, N1, V1) as
SELECT ID, cast(NULL as int), cast(NULL as int) FROM tst2;
```

```
create or replace view vtst (ID, N1, V1) as
SELECT ID, cast(NULL as int), NULL FROM tst2;

create or replace view vtst (ID, N1, V1) as
SELECT ID, NULL, NULL FROM tst2;

sql> show vtst

ID INTEGER
N1 INTEGER
V1 INTEGER
…
```

11) Если указано `WITH CHECK OPTION`, то таблица представления должна быть обновляемой.

Общие правила

1) Представление – это таблица-результат выполнения <запроса выборки>. Указанный <запрос выборки> выполняется СУБД каждый раз при обработке SQL-запроса, содержащего ссылку на представление.
2) При создании материализованного представления создаётся специальная таблица (спецтаблица). Типы, имена и количество столбцов спецтаблицы совпадают с типами, именами и количеством столбцов представления. При необходимости извлечения данных материализованного представления запрос к представлению переадресуется к спецтаблице (на уровне трансляции SQL-запроса).
3) Внутренний SQL-запрос для загрузки данных из спецтаблицы (`select * from` <имя схемы>.<имя материализованного представления>) выполняется СУБД под именем и с правами пользователя – владельца представления.
4) Для материализованных представлений разрешено создание простых и составных индексов (индексы создаются для спецтаблиц).

Примечание
В текущей версии СУБД ЛИНТЕР фразовые индексы для материализованных представлений не поддерживаются.

5) В случае модификации/удаления какого-либо объекта или модификации данных (добавление, удаление, модификация записей) какого-либо объекта, на основе которого создано одно или несколько материализованных представлений, все такие представления помечаются как «представления с изменёнными данными». Обновление этих представлений выполняется СУБД ЛИНТЕР автоматически при первом после изменения данных обращении к ним.
6) В материализованных представлениях недопустимо использование конструкций `SYSDATE`, `NOW`, `CURRENT_TIMESTAMP`, `USER`, `LINTER_USER_ID`, `SEQUENCE` (`CURRVAL`, `NEXTVAL`), `LAST_AUTOINC`, `LAST_ROWID`. В этом случае будет возвращён код завершения 2281 «`Недопустимое выражение для материализованного представления`».
7) Если представление обновляемое, то для каждой строки этого представления есть соответствующая строка базовой таблицы, порождающая строку представления. Для каждого столбца в представлении есть соответствующий порождающий столбец в базовой таблице. Вставка/удаление/изменение строки в представлении есть вставка/

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

удаление/изменение соответствующей строки в базовой таблице. В данной ситуации возможны следующие случаи:

- если указана опция WITH CHECK OPTION и <запрос выборки> содержит <WHERE-спецификацию>, то информация, вносимая в представление при добавлении/изменении, будет проверяться на соответствие <WHERE-спецификации>;
- если опция WITH CHECK OPTION не указана, то обновление такого представления не вносит ограничений на значения добавляемых данных;
- для представления с опцией CHECK OPTION разрешена операция UPDATE CURRENT OF.

8) Если в представлении содержатся таблицы без указания имени схемы, то при выполнении запроса выборки представления будет использоваться схема владельца представления (а не пользователя, выполняющего запрос выборки из представления).

Модификация представления (ALTER TABLE)

Функция

Определение оператора модификации представления.

Спецификация

[1] <модификации представления>::=
ALTER TABLE [<имя схемы>.] <имя представления>
{RENAME TO <имя представления> | ALTER [COLUMN] <имя столбца> RENAME TO <имя столбца>}

Общие правила

1) Команда запрещена для материализованных представлений.
2) Для выполнения команды необходимы уровень доступа RESOURCE к БД и привилегия доступа ALTER к изменяемому представлению.

Пример

```
create table test_table (name char(10));
create view test_view as select name from test_table;
alter table test_view rename to view_test;
alter table view_test alter column name rename to firstname;
select firstname from view_test;
```

Удаление представления (DROP VIEW)

Функция

Определение оператора удаления представления.

Спецификация

[1] <удаление представления>::=
DROP [VIEW | TABLE] [<имя схемы>.]<имя представления> [CASCADE]

Синтаксические правила

1) При отсутствии <имени схемы> будет произведена попытка поиска и удаления представления в текущей схеме.

Общие правила

1) Удалить представление может только его владелец. Администратор БД имеет возможность удалить одновременно все объекты некоторого пользователя с помощью команды DROP USER <имя пользователя> CASCADE.
2) Удаление представления возможно также с помощью предложения DROP TABLE <имя представления>; это удобно, когда пользователь не знает характера удаляемого объекта (базовая таблица или представление).
3) При указании опции CASCADE одновременно с удаляемым представлением удаляются и все представления, порожденные на его основе.

```
create or replace table test(i int, c char(10));
create or replace view testv1 as select * from test;
create or replace view testv2 as select * from testv1  where
 i=100;
create or replace view testv3 as select * from testv2  where
 c='abc';
drop view testv1 cascade;
```

4) При изменении объектов, на основе которых создано представление, все ссылки в БД у представления (и на представление) от этих объектов удаляется и представление помечается как «представление, нуждающееся в обновлении зависимостей». Зависимости обновляются при первом обращении к представлению за данными (возможно, в составе сложного запроса со многими таблицами (представлениями) или при обращении к представлению, которое основано на этом представлении), поэтому до обновления зависимостей каскадное удаление не выполняется.

   Пример (с использованием утилиты inl): каскадное удаление не выполняется, так как зависимости между объектами не восстановлены:

```
create or replace table test(i int);
create or replace view testv as select * from test;
! Выполняется разрыв зависимости между таблицей
! и порожденным на её основе представлением.
drop table test;

create or replace table test(i int);
! Хотя таблица, на основе которой создано представление,
! вновь существует в БД, зависимость между ней и порожденным
! на её основе представлением не восстановлена (не было
! обращения к данным представления).
! Каскадное удаление не выполняется.

drop table test cascade;

! Так как зависимости не восстановлены, то каскадное удаление
! не было выполнено. Представление все еще существует в БД.
show test
Характеристика столбца
 I                                    INTEGER
Запрос для представления
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
SELECT * FROM "TEST";
```

5) Удаление базовой таблицы, породившей представление, не приводит к удалению самого представления. Таким образом, можно пересоздавать базовую таблицу без обновления представления.

6) При запуске СУБД ЛИНТЕР с ключом `/COMPATIBILITY=STANDARD` запрещается обычное удаление представления, на которое ссылаются другие представления. При этом на консоль ядра СУБД и в файл `linter.out` выдаётся список объектов, ссылающихся на удаляемый объект и не позволяющих его удалить. Пример списка:

```
INFO Can't delete relation '"SYSTEM"."TEST1" (#187)' because
 exists reference(s):
  "SYSTEM"."TESTV" (#188)
  "SYSTEM"."TESTV2" (#190).
```

7) Каскадное удаление таблицы, на которую ссылаются представления (VIEW), разрешено всегда.

Примечание
Механизм удаления таблиц, на которые ссылаются представления, не распространяется на таблицы, созданные до включения в ядро СУБД ЛИНТЕР данной функциональной возможности.

Привилегии

Определение привилегий доступа к ресурсам БД и операций с ними (GRANT)

Функция

Определение оператора предоставления пользователю привилегий доступа к ресурсам БД и операций с ними.

Спецификация

[1] <определение привилегии>::=
<привилегия на обработку данных>|<привилегия на доступ к ресурсам БД>

[2] <привилегия на обработку данных>::=
GRANT {ALL [PRIVILEGES] |
{SELECT | INSERT | DELETE | UPDATE | ALTER | INDEX | REFERENCES}[, …]}
ON [TABLE] [<имя схемы>.]{<имя таблицы>|<имя представления>|<имя синонима>}
TO {PUBLIC|{<имя пользователя>|<имя роли>}[, …]}

[3] <привилегия на доступ к ресурсам БД>::=
GRANT {CONNECT | RESOURCE | DBA | BACKUP} [, …]
TO <имя пользователя>[, …]
[IDENTIFIED BY <тип идентификации>[, …]]

[4] <тип идентификации>::=
<пароль> | SYSTEM | PROTOCOL | LDAP | KRB

Синтаксические правила

1) <Имя таблицы> должно задавать базовую таблицу.

2) Опция `ALL` равнозначна списку, включающему все привилегии для <имени таблицы> (<имени представления>): `ALL=SELECT+INSERT+DELETE+UPDATE+ALTER+INDEX+REFERENCES`.

```
Например,
GRANT SELECT, index ON Auto TO PUBLIC;
GRANT all ON Auto TO "Иванов";
```

3) Если вместо списка пользователей указано PUBLIC, то всем пользователям (существующим и будущим) будут предоставлены определенные данным оператором права на указанную таблицу.

```
grant all on auto to public;
```

4) Вторая форма оператора GRANT предназначена для определения новых пользователей с указанием их категорий либо изменения категорий существующих пользователей.

```
grant connect to USER1, "Иванов", "Nik" identified by 'hgt66#',
 '6ыф_()', '123';
```

5) Конструкция IDENTIFIED BY SYSTEM устанавливает для пользователя режим встроенной аутентификации операционной системой (см. пункт «Создание пользователя»).

6) Конструкция IDENTIFIED BY PROTOCOL устанавливает для пользователя режим встроенной аутентификации по имени пользователя, зарегистрированного в ОС (см. пункт «Создание пользователя»).

```
grant connect to USER1, USER2 identified by PROTOCOL;
```

7) Конструкция IDENTIFIED BY LDAP предоставляет указанную привилегию пользователю, аутентификация которого выполняется через LDAP-сервер (см. пункт «Создание пользователя»).

```
grant connect to USER1, USER2 identified by LDAP;
```

8) Конструкция IDENTIFIED BY KRB предоставляет указанную привилегию пользователю, идентификация и аутентификация которого выполняется через KRB-сервер (см. пункт «Создание пользователя»).

```
grant connect to USER1, USER2 identified by KRB;
```

9) Количество пользователей и <типов идентификации> может быть разным. При этом количество пользователей должно быть не меньше количества <типов идентификации>. Соответствие <пользователь>–<тип идентификации> производится слева направо. Пользователи, оставшиеся без соответствующих им <типов идентификации>, получат «пустой» пароль (18 пробелов).

```
GRANT RESOURCE TO User_1,User_2 IDENTIFIED BY 'Hi_Life';
(у пользователя User_2 будет пустой пароль)
```

10) Длина <символьного литерала>, задающего <пароль>, должна быть не более 18 символов.

11) Привилегии могут предоставляться только владельцем указанной таблицы/представления/синонима.

Примечание

Привилегия, назначенная синониму таблицы или представления, будет унаследована соответствующей таблицей или представлением и не будет удалена при удалении синонима.

12) Нельзя назначить привилегии самому себе.

13) Вводить нового пользователя (с определением <категории пользователя>) может только администратор БД (категория DBA).

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

14) Привилегия REFERENCES, данная пользователем U1 пользователю U2 на таблицу U1.T1, разрешает пользователю U2 создавать ссылки (REFERENCES) на эту таблицу, при этом пользователь U2 должен иметь категорию как минимум RESOURCE.
15) Для удаления ссылок привилегия REFERENCES не требуется.
16) Привилегия BACKUP разрешает пользователю БД оперативное архивирование БД или её отдельных объектов.

Отмена привилегий доступа к ресурсам БД и операций с ними (REVOKE)

Функция

Определение оператора отмены привилегий доступа к ресурсам БД и операций с ними.

Спецификация

[1] <отмена привилегии>::=
<отмена привилегии на обработку данных>|<отмена привилегии на доступ к ресурсам БД>
[2] <отмена привилегии на обработку данных>::=
REVOKE {ALL [PRIVILEGES] |
{SELECT|INSERT|DELETE|UPDATE|ALTER|INDEX|REFERENCES}[, …]}
ON [TABLE] [<имя схемы>.]{<имя таблицы>|<имя представления>|<имя синонима>}
FROM {PUBLIC | { <имя пользователя> | <имя роли>}[, …]}
[3] <отмена привилегии на доступ к ресурсам БД>::=
REVOKE {CONNECT | RESOURCE | DBA | BACKUP}
FROM <имя пользователя>[, …]

Синтаксические правила

1) Спецификация конструкций <привилегия>, <пользователь>, <категория пользователя>, <список пользователей> приведена в пункте Определение привилегий доступа к ресурсам БД и операций с ними.
2) <Имя таблицы> должно задавать базовую таблицу.
3) Опция ALL равнозначна списку, отменяющему все привилегии для <имени таблицы> (SELECT+INSERT+DELETE+UPDATE+ALTER+INDEX+REFERENCES).

```
revoke all on auto from "Иванов";
```

4) Если вместо списка пользователей указано PUBLIC, то для всех пользователей (существующих и будущих) указанные привилегии будут отменены, за исключением тех пользователей, которым эти привилегии назначены персонально.

```
revoke select, index on Auto from PUBLIC;
```

5) Привилегии может отменять только владелец таблицы/представления/синонима.
6) Нельзя отменить привилегии самому себе.
7) Отнять (понизить) категорию пользователя может только администратор БД (категория DBA). При этом считается, что категория DBA включает в себя привилегии категории RESOURCE и категория RESOURCE включает в себя привилегии категории CONNECT, т.е. категории упорядочены. Когда отнимается какая-либо категория, это означает снижение категории на один уровень. Например:

Существующая категория	Отнимаемая категория	Получаемая категория
DBA	DBA	RESOURCE

Существующая категория	Отнимаемая категория	Получаемая категория
DBA	RESOURCE	CONNECT
RESOURCE	RESOURCE	CONNECT

```
REVOKE RESOURCE FROM User_1,User_2;
```

8) Лишение всех привилегий (CONNECT) равноценно запрету доступа пользователя к БД.
9) Конструкция REVOKE REFERENCES ... отменяет ранее предоставленную привилегию REFERENCES, т.е. запрещает создавать ссылки на столбец (столбцы) указанной в <имени таблицы> таблицы от других таблиц БД.
10) Конструкция REVOKE BACKUP запрещает пользователю оперативное архивирование БД или её отдельных объектов.

Индексы

Создание индекса (CREATE INDEX)

Функция

Определение оператора создания индекса на столбец таблицы.

Спецификация

[1] <создание индекса>::=
CREATE [IF NOT EXISTS | OR REPLACE] [UNIQUE] INDEX
{<неименованный индекс> |<именованный индекс> |<функциональный индекс>}
[INDEXFILE <номер файла>] [BY APPEND]
[2] <неименованный индекс>::=
<имя столбца> ON [<имя схемы>.] <имя таблицы>
[3] <именованный индекс>::=
<имя индекса> ON [<имя схемы>.] <имя таблицы> (<имя столбца> [, …])
[4] <функциональный индекс>::=
<имя индекса> ON [<имя схемы>.] <имя таблицы> (<функция>(<имя столбца>))
[5] <номер файла>::= <беззнаковое целое>
[6] <функция>::= UPPER | LOWER

Синтаксические правила

1) В качестве <имени таблицы> может выступать только базовая таблица.

```
create index model on auto by append;
```

2) Опция OR REPLACE заставляет удалять существующий в БД индекс и создавать его под тем же именем, но с другими параметрами.
3) Опция IF NOT EXISTS отменяет выполнение оператора, если указанный индекс уже существует в БД.
4) Одновременное использование опций IF NOT EXISTS и OR REPLACE запрещено.
5) Максимальное количество столбцов в составном индексе равно 31.
6) Максимальное количество составных индексов в таблице – 100, максимальное количество составных индексов для одного столбца – 10.
7) Для таблиц с одним индексным файлом по умолчанию индекс строится в данном индексном файле.

© Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

8) Для таблиц, имеющих более одного индексного файла, по умолчанию индекс строится во втором индексном файле.

9) Опция INDEXFILE задает номер основного индексного файла, где должен храниться создаваемый индекс. Если индекс не может полностью поместиться в этот файл, то он будет размещён (полностью или частично) в другом доступном индексном файле.

```
create index "Основные характеристики" on auto (model, cylnders,
 weight, color) indexfile 3;
```

10) <Номер файла> не должен превышать количества зарезервированных при создании таблицы индексных файлов.

11) Длина индексируемого столбца (столбцов) не должна превышать 1024 байт.

Примечание
Для каждого индексируемого столбца переменной длины к длине столбца прибавляется 2 байта (длина).

12) Опция BY APPEND задает режим создания индекса не через сортировку данных (как делается по умолчанию), а через последовательное добавление элементов.

Создание индекса на большую таблицу через сортировку (по умолчанию) требует большого дискового пространства и достаточно большого объема оперативной памяти. При создании индекса с помощью опции BY APPEND файл сортировки не используется, записи в процессе прохода по конвертеру начинают сразу же добавляться в файл индекса таблицы.

Скорость построения индекса будет значительно ниже, поскольку каждый раз необходимо искать место для вставки записи в индексе (если же записи отсортированы, в этом нет необходимости).

Примечание
Опцию рекомендуется использовать в случае отсутствия большого свободного дискового пространства, появления какой-либо ошибки сортировки или слишком большого времени сортировки.

Общие правила

1) Команда создает простые (не фразовые) индексы. Создание фразовых индексов описано в документе «Полнотекстовый поиск в базе данных», пункт «Создание фразового индекса».

2) Создавать индекс может владелец таблицы либо пользователь, получивший привилегию INDEX на указанную таблицу.

3) Если <именованный индекс> создается для одного столбца, то созданный индекс является именованным одностолбцовым индексом.

4) Если <именованный индекс> создается для нескольких столбцов, то созданный индекс является именованным многостолбцовым (составным) индексом.

5) Опция UNIQUE задает ограничение целостности UNIQUE для создаваемого индекса. Указание опции UNIQUE эквивалентно выполнению последовательности команд:

   - в случае создания неименованного индекса:

```
CREATE INDEX "Имя столбца" ON "Имя таблицы";
ALTER TABLE "Имя таблицы" ADD UNIQUE ("Имя столбца");
```

- в случае создания именованного индекса:

```
CREATE INDEX "Имя индекса" ON "Имя таблицы"("Имя столбца"[, ...]);
ALTER TABLE "Имя таблицы" ADD UNIQUE ("Имя столбца"[, ...]);
```

- в случае создания функционального индекса опция UNIQUE неприменима.

6) Конструкция для отмены ограничения целостности UNIQUE отсутствует. Его можно снять вручную:
   - в случае неименованного индекса:

```
ALTER TABLE "Имя таблицы" DROP UNIQUE ("Имя столбца");
DROP INDEX "Имя столбца" ON "Имя таблицы";
```

   - в случае именованного индекса:

```
ALTER TABLE "Имя таблицы" DROP UNIQUE ("Имя столбца"[, ...]);
DROP INDEX "Имя индекса" ON "Имя таблицы";
```



Примечание
Отдельно поданная команда удаления индекса

```
DROP INDEX "Имя столбца" ON "Имя таблицы";
```

или

```
DROP INDEX "Имя индекса" ON "Имя таблицы";
```

(без предварительной команды ALTER TABLE ...) не отменяет ограничение целостности UNIQUE.

7) Если указана опция UNIQUE, а столбец (столбцы) содержит неуникальные данные, то индекс не создается.
8) При определении таблиц с одностолбцовыми (многостолбцовыми) ключами создание и именование индексов выполняется автоматически (исключая столбцы с модификатором AUTOROWID).
9) Имя автоматически созданного индекса имеет вид:

```
Index #nnnnnnnnnn#
```

где nnnnnnnnnn – ROWID записи для индекса/ключа в таблице $$$ATTRI.

10) Индексы удаляются автоматически при удалении ключей.
11) Создание индекса разрешено для пустых таблиц.
12) Запрещено создание составных, простых именованных и функциональных индексов для системных таблиц.
13) Запрещено создание простых и функциональных индексов на столбцы $$$S14, $$$S24, $$$S35 (столбцы системных таблиц $$$SYSRL, $$$ATTRI, $$$USR соответственно).
14) Для столбцов типа PRIMARY KEY, FOREIGN KEY и UNIQUE индекс создается автоматически всегда.
15) Для столбцов типа AUTOROWID создавать индекс не имеет смысла, т.к. записи выбираются напрямую по значению столбца.
16) Составные индексы применяются при обработке <предиката сравнения> (=, <>, >=, >, <=, <), <интервального предиката> (BETWEEN), <предиката подобия> (LIKE), <предиката неопределенного значения> (NULL), <предиката вхождения> (IN).

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

Примечание
Время перестройки составного индекса меньше, чем суммарное время перестройки простых индексов на те же столбцы.

17) Функциональные индексы применяются при обработке <предиката сравнения> типа <операция> <константа>, где в качестве <операции> допускаются сравнения (<,>,=,<>, LIKE, IS NULL, ...), за исключением 'NOT LIKE', 'NOT SIMILAR', 'NOT NULL', 'NOT BETWEEN', 'NOT IN', 'NOT MATCH', 'NAN' и 'NOT NAN'.

```
Тестовые данные:
create or replace table test(id int, j int, ch char(10));
insert into test values (1,1,'aAa');
insert into test values (2,2,'abA');
insert into test values (3,3,'aaA');
insert into test values (4,4,'aAaA');
insert into test values (5,5,'a');
insert into test values (6,6,'AA');
insert into test values (7,7,NULL);
create index "AAA" on test(upper(ch));

select rowid, test.* from test where upper(ch)='AAA';

ROWID       ID          J           CH
 -----       --          -           --
|          1|          1|          1|aAa       |
|          3|          3|          3|aaA       |

select rowid, test.* from test where upper(ch) > 'AAA';

ROWID       ID          J           CH
 -----       --          -           --
|          2|          2|          2|abA       |
|          4|          4|          4|aAaA      |

select rowid, test.* from test where upper(ch) < 'AAA';

ROWID       ID          J           CH
 -----       --          -           --
|          5|          5|          5|a         |
|          6|          6|          6|AA        |

select rowid, test.* from test where upper(ch) is NULL;

ROWID       ID          J           CH
 -----       --          -           --
|          7|          7|          7|          |
```

```
select rowid, test.* from test where upper(ch) is not NULL;

ROWID       ID          J           CH
 -----       --          -           --
|          1|          1|          1|aAa       |
|          2|          2|          2|abA       |
|          3|          3|          3|aaA       |
|          4|          4|          4|aAaA      |
|          5|          5|          5|a         |
|          6|          6|          6|AA        |

select rowid, test.* from test where upper(ch) like '_B_';

ROWID       ID          J           CH
 -----       --          -           --
|          2|          2|          2|abA       |
```

18) Функциональные индексы не используются ни для сортировки, ни для вычисления соединений.

Удаление индекса (DROP INDEX)

Функция

Определение оператора удаления индекса.

Спецификация

[1] <удаление индекса>::=
DROP INDEX {<имя столбца> | <имя индекса>}
ON [<имя схемы>.]<имя таблицы>

Синтаксические правила

1) Опция <имя столбца> задает удаление одностолбцового неименованного индекса (как правило, созданного автоматически).

```
drop index personid on auto;
```

2) Опция <имя индекса> задает удаление любого именованного индекса (одностолбцового или составного).

```
drop index "Основные характеристики" on auto;
```

Общие правила

1) Удалять индекс может владелец таблицы либо пользователь, получивший привилегию INDEX на указанную таблицу.



Примечание
Администратор БД имеет возможность удалить одновременно все объекты некоторого пользователя (включая индексы) с помощью команды `DROP USER <имя пользователя> CASCADE`.

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

2) Если имя составного индекса совпадает с именем столбца таблицы, то первым удаляется составной именованный индекс, если его нет – то одностолбцовый именованный индекс, если его нет – то неименованный индекс столбца с указанным именем.

```
create or replace table test_drop (test int, a int, b int);
create index test on test_drop;
create index test on test_drop  (a, b);
drop index test on test_drop;
/* удаляется именованный составной индекс */
```

Корректировка индекса (CORRECT INDEX)

Функция

Определение оператора корректировки индекса.

Спецификация

[1] <корректировка индекса>::=
CORRECT INDEX {<имя столбца> | <имя индекса>}
ON [<имя схемы>.]<имя таблицы>
FOR {ROWID <номер записи>
| [FILE <номер файла>] PAGE <номер страницы> [WITH TEST]}
[WAIT | NOWAIT]

Синтаксические правила

1) <Имя столбца> должно соответствовать столбцу таблицы, по которому ранее был создан индекс (по команде CREATE INDEX или автоматически при объявлении столбца PRIMARY KEY).
2) <Имя индекса> должно задавать один из ранее созданных именованных индексов данной таблицы.
3) Если индекс создавался как неименованный, то вместо <имени индекса> необходимо использовать <имя столбца>.
4) Если <номер файла> не указан, по умолчанию используется 1.
5) Отсчет номеров файлов и страниц начинается с 1.
6) Опция WITH TEST зарезервирована для дальнейшего использования.
7) <Номер файла> задает номер индексного файла таблицы. Если он не задан, по умолчанию используется 1.
8) <Номер страницы> задает номер страницы индексного файла, которая должна быть откорректирована.



Примечание
Значения <номер файла> и <номер страницы> можно получить с помощью утилиты testdb (см. документ «Тестирование базы данных», пункт «Вывод в файл ROWID по экстенту файла таблицы»). Сама утилита testdb такие ошибки БД не исправляет.

9) <Номер записи> задает системный номер записи (ROWID), для которой должен быть скорректирован индекс.

```
correct index make on auto for rowid 100;
```

10) Имена таблиц, столбцов/индексов и номера ROWID, для которых нужно сделать корректировку индекса, можно получить с помощью SQL-команды TEST TABLE или утилиты testdb.

Пример протокола утилиты testdb (с ключом -le):

```
...
* ERROR
* Table #47, ROWID #9059427, column #1 index
index file #1 page #151271
* Index contains invalid value
* ERROR
* Table #47, converter ROWID #9091145, column #1 index
* ROWID not found in index
* ERROR
* Table #47, converter ROWID #9092958, column #1 index
* ROWID not found in index
…
```

В данном случае имеет смысл выполнить команду CORRECT INDEX для значений ROWID 9059427, 9091145, 9092958.

Получить имя таблицы и столбца можно по запросу:

```
select trim($$$s13), trim($$$s23)
  from $$$sysrl,$$$attri
 where $$$sysrl.$$$s11=$$$attri.$$$s21
   and $$$s11=47
   and $$$attri.$$$s22=1;
```

11) Получить список индексов можно по команде show утилиты inl (см. документ «Командный интерфейс», раздел «SHOW»).
12) Если не задан модификатор WAIT (NOWAIT), по умолчанию используется WAIT.

Общие правила

1) Если указана конструкция FOR ROWID:
   - команда исправляет в указанном индексе элементы, относящиеся к заданному системному номеру записи (ROWID), удаляя при необходимости неправильные элементы и добавляя правильные;
   - если указанного ROWID нет в таблице, и нет никаких элементов индекса с таким ROWID, выдается код завершения 714 («Несуществующий ROWID – нельзя исправить индекс»);
   - при выполнении команды индекс не блокируется (блокируется только ROWID записи).
2) Конструкция FOR FILE… PAGE используется при появлении массовых ошибок, связанных с индексами (фиксируются в файле протоколирования linter.out), либо по результатам тестирования БД утилитой testdb (эта утилита всего лишь выявляет дефекты в индексах, но не устраняет их).
3) Необходимые для команды значения можно получить с помощью утилиты testdb.

```
Пример протокола утилиты testdb (с ключом -lr):
Таблица #48, составной индекс #1, узел #33554917
· *  Неверный заголовок страницы
```

```
· В этом случае параметры для команды будут следующие:
· <номер файла>: 2
· <номер страницы>: 485
Для извлечения <номера файла> и <номера страницы> из значения
 узла:
· преобразовать значение узла в шестнадцатеричный вид (например,
· 33554917 = 0x200001E5);
· взять первую цифру полученного значения (это номер файла),
 например, 0x2 = 2,
 и преобразовать её в десятичный вид;
· взять оставшиеся цифры полученного значения (это номер
 страницы), например,
 0x1E5, и преобразовать их в десятичный вид (в данном примере это
 485).
```

4) Если указанная страница не найдена в указанном индексе, выдается код завершения 1131 («Страница не найдена в индексе»). Этот же код завершения выдается и в том случае, если индекс состоит только из одной страницы, в этом случае рекомендуется полностью перестроить индекс.

5) Если перестройка отдельной указанной индексной страницы не может быть выполнена – выдается код завершения 1132 («Нельзя исправить одну указанную страницу»). В этом случае необходима полная перестройка индекса.

Корректировка конвертера (CORRECT INDEX ROWID)

Функция

Определение оператора корректировки конвертера данных.

Спецификация

[1] <корректировка конвертера>::=
CORRECT INDEX ROWID ON [<имя схемы>.]<имя таблицы>
FOR ROWID <список> [WAIT | NOWAIT]
[2] <список>::= <номер записи> [, …]

Синтаксические правила

1) <Номер записи> задает системный номер записи (ROWID), для которой должен быть скорректирован конвертер данных.

```
correct index rowid on auto for rowid 100;
```

2) Имена таблиц и <список> ROWID, для которых нужно сделать корректировку конвертера, можно получить с помощью SQL-команды TEST TABLE или утилиты testdb (см. документ «Тестирование базы данных», пункт «Вывод в файл ROWID по экстенту файла таблицы»).

Общие правила

1) Необходимость корректировки конвертера возникает после получения кода завершения 10 или при наличии проблем в конвертере, выявленных в процессе тестирования БД.

2) Корректировка конвертера выполняется в следующих ситуациях:
   - в файле данных таблицы есть запись с указанным ROWID (обязательно одна), но в конвертере нет ссылки на нее (или есть ссылка на другую страницу либо пустой элемент);
   - в файле данных таблицы нет записи с указанным ROWID, но в конвертере есть ссылка на нее.
3) Ситуации, когда в файле данных несколько записей с указанным ROWID, не правятся, но на консоль ядра СУБД ЛИНТЕР выдается соответствующее предупреждение.

Корректировка битовой карты таблицы (CORRECT BITMAP)

Функция

Определение оператора корректировки битовой карты таблицы.

Спецификация

Варианты:

[1] <корректировка битовой карты> : : =
CORRECT BITMAP ON [<имя схемы>.]<имя таблицы>
FOR {DATA|INDEX|BLOB} PAGE <номер страницы>
[FILE <номер файла>] [WAIT | NOWAIT]

[1] <корректировка битовой карты> : : =
CORRECT BITMAP PAGE <номер страницы битовой карты>
ON [<имя схемы>.]<имя таблицы> FOR {DATA|INDEX|BLOB}
FILE <номер файла>
[WAIT | NOWAIT]

Синтаксические правила

1) <Номер страницы> задает номер страницы соответствующего файла таблицы:
   - DATA – файла данных;
   - INDEX – индексного файла;
   - BLOB – BLOB-файла.
2) <Номер файла> задает номер соответствующего файла таблицы. Если не задан, по умолчанию используется значение 1.
3) <Номер страницы битовой карты> задает номер страницы битовой карты соответствующего файла таблицы.

Общие правила

1) Команду целесообразно применять в случае возникновения ошибок в битовой карте таблицы при невозможности по какой-либо причине (например, большой объём БД) использовать для устранения ошибки утилиты testdb.
2) Вариант с опцией CORRECT BITMAP используется при незначительных (единичных) повреждениях битов в битовой карте (в этом случае выполняется корректировка только ошибочных битов).
3) Вариант с опцией CORRECT BITMAP PAGE используется при значительных (множественных) повреждениях битов в битовой карте (в этом случае выполняется корректировка страниц целиком).

Примеры

Сообщения утилиты testdb и соответствующие им команды корректировки битовой карты:

1)

```
!Таблица #118, битовая карта файла данных #1, страница #1, бит #20
!Очищен бит для заполненной страницы
CORRECT BITMAP ON AUTO FOR DATA PAGE 20;
```

2)

```
!Таблица #118, битовая карта файла индексов #1, страница #1, бит
 #2
!Установлен бит для незаполненной страницы
CORRECT BITMAP ON AUTO FOR INDEX PAGE 2;
```

3)

```
!Таблица #5, битовая карта файла BLOB #1, страница #1, бит #20
!Очищен бит для заполненной страницы
CORRECT BITMAP ON $$$PROC FOR BLOB PAGE 20;
```

Ошибки типа:

```
*  Таблица #48, битовая карта файла индексов #2, страница #16, бит
 #28
*  Очищен бит для страницы, которой нет в файле
```

вызваны тем, что, по какой-то причине содержимое 16-ой страницы битовой карты не было очищено от данных индекса (или данные индекса туда были возвращены при откате), после чего с этой страницей велась работа как со страницей битовой карты.

Команда для корректировки:

```
CORRECT BITMAP PAGE 16 ON <имя таблицы> FOR INDEX FILE 2;
```

Имя таблицы можно узнать по её системному номеру (48), указанному в сообщении утилиты testdb.

Последовательности

Последовательность (sequence) представляет собой специальный объект БД для генерации последовательных значений в пределах всей БД (в отличие от атрибута AUTOINC столбца таблицы, который генерирует уникальную последовательность значений в пределах только одной таблицы). Каждое следующее обращение к последовательности увеличивает ее текущее значение на шаг, определяемый при создании последовательности.

Создание последовательности (CREATE SEQUENCE)

Функция

Определение оператора создания общей или личной последовательности.

Спецификация

[1] <создание последовательности>::=
CREATE [IF NOT EXISTS | OR REPLACE] [PUBLIC] SEQUENCE
[<имя схемы>.] <имя последовательности>
[AS {BIGINT | INTEGER | SMALLINT}]
[START WITH <начальное значение>]
[INCREMENT BY <шаг>]
[MAXVALUE <верхняя граница> | NO MAXVALUE]
[MINVALUE <нижняя граница> | NO MINVALUE]
[CYCLE | NO CYCLE]

[2] <шаг>::= <целочисленный литерал>

Синтаксические правила

1) <Имя схемы> задает схему для создания личной последовательности, владельцем которой является пользователь, выполняющий команду.
2) <Имя последовательности> должно быть уникальным для PUBLIC-последовательностей и в пределах <имени схемы> для личных последовательностей.
3) Опция OR REPLACE заставляет удалять существующую в БД последовательность и создавать её под тем же именем, но с указанными параметрами.
4) Опция IF NOT EXISTS отменяет выполнение оператора, если указанная последовательность уже существует в БД.
5) Одновременное использование опций IF NOT EXISTS и OR REPLACE запрещено.
6) Если задана опция PUBLIC, последовательность доступна всем пользователям БД.
7) Указание <имени схемы> для PUBLIC-последовательности недопустимо.
8) Личная последовательность доступна:
   - с ключом /COMPATIBILITY=STANDARD – только создателю последовательности;
   - без ключа /COMPATIBILITY=STANDARD – всем пользователям.
9) Если некоторый пользователь является владельцем личной последовательности с <именем последовательности>, то для обращения к PUBLIC-последовательности с тем же <именем последовательности> ему необходимо установить текущую схему, отличную от схемы, в которой создана личная последовательность.
10) <Начальное значение>, <верхняя граница> и <нижняя граница> – целочисленные значения в диапазоне от -9 223 372 036 854 775 808 до +9 223 372 036 854 775 807 (тип данных BIGINT).
11) <Шаг> последовательности должен лежать в интервале от -72057594037927936 до 72057594037927935.
12) Величины <начального значения>, <шага>, <верхней границы> и <нижней границы> не должны логически противоречить друг другу.
13) <Верхняя граница> имеет смысл при положительном значении <шага>, <нижняя граница> – при отрицательном.

```
CREATE PUBLIC SEQUENCE "Номера платежных поручений"
  START WITH 1
  INCREMENT BY 1;

select "Номера платежных поручений".NEXTVAL;
```


© Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
|2 |

CREATE SEQUENCE My_Seq INCREMENT BY -5 MINVALUE -273;

select My_Seq. NEXTVAL;

|-6 |
```

14) Конструкция AS {BIGINT | INTEGER | SMALLINT} задаёт тип возвращаемых последовательностью значений. По умолчанию – BIGINT.
15) Если используется конструкция AS, то она должна следовать сразу после <имени последовательности>, тогда как все остальные параметры могут располагаться в произвольном порядке.
16) Конструкция CYCLE | NO CYCLE разрешает/запрещает создавать циклическую последовательность. По умолчанию – NO CYCLE.
17) Опции NO MAXVALUE и NO MINVALUE автоматически устанавливают нижнюю и верхнюю границы последовательности в зависимости от типа возвращаемого последовательностью значения.
18) Если опции MAXVALUE и NO MAXVALUE не заданы, по умолчанию используется NO MAXVALUE.
19) Если опции MINVALUE и NO MINVALUE не заданы, по умолчанию используется NO MINVALUE.

Общие правила

1) Для создания PUBLIC SEQUENCE необходим уровень прав DBA.
2) Для создания личной последовательности необходим уровень прав RESOURCE.
3) Если задана опция CYCLE, и при вызове функции NEXTVAL выявляется выход за пределы интервала между минимальным и максимальным значениями последовательности, то функция NEXTVAL вернёт:
   - минимальное значение последовательности, если значение <шага> последовательности положительно;
   - максимальное значение, если значение <шага> отрицательно.
4) Если заданы опции NO MAXVALUE и NO MINVALUE (или они используются по умолчанию), то <нижней границе> и <верхней границе> последовательности присваиваются минимальные и максимальные значения для соответствующих возвращаемых типов:
   - BIGINT: -9223372036854775807 и 9223372036854775807;
   - INTEGER: -2147483647 и 2147483647;
   - SMALLINT: -32767 и 32767.
5) Если задана опция START WITH, то при создании возрастающей последовательности <нижняя граница> устанавливается в <начальное значение> независимо от указаний MINVALUE <нижняя граница> или NO MINVALUE.

```
create or replace sequence my_seq start with 100
minvalue -500 no maxvalue increment by 5;
select my_seq.nextval;
select my_seq.currval;

|                100|
```

6) Если задана опция START WITH, то при создании убывающей последовательности <верхняя граница> устанавливается в <начальное значение> независимо от указаний MAXVALUE <верхняя граница> или NO MAXVALUE.
7) Для возрастающей последовательности при отсутствии опции START WITH минимальное значение берется равным 1. Для убывающей последовательности минимальное значение задается параметром MINVALUE; если его нет, то берется максимальное по модулю отрицательное значение.
8) Для убывающей последовательности при отсутствии опции START WITH максимальное значение берется равным -1. Для возрастающей последовательности минимальное значение задается параметром MAXVALUE; если его нет, то берется максимальное положительное значение.
9) Если опция START WITH не задана, то для возрастающей последовательности стартовым значением берется минимальное значение, для убывающей последовательности – максимальное значение.
10) Если задана опция START WITH, то для режима nocycle <начальное значение> последовательности должно быть внутри границ последовательности и не совпадать с граничными значениями. В режиме cycle <начальное значение> последовательности может совпадать с граничным значением.
11) Если заданы опции CYCLE и MINVALUE, то при достижении <верхней границы> возрастающей последовательности очередным значением последовательности будет MINVALUE.

```
create or replace sequence my_seq start with 2
minvalue 1  maxvalue 3 increment by 1 cycle;
select my_seq.nextval;
select my_seq.nextval;
select my_seq.nextval;
select my_seq.nextval;
select my_seq.nextval;

|                  1|
|                  2|
|                  3|
|                  1|
|                  2|
|                  3|
```

12) Если заданы опции CYCLE и MINVALUE, то при достижении <нижней границы> убывающей последовательности очередным значением последовательности будет MAXVALUE.
13) Значения последовательностей привязываются не к текущему, а корневому родительскому каналу.

```
Например, в хранимой процедуре test_prc() (выполняется по
 дочернему каналу) перед запросом seq.currval нет необходимости
 запрашивать seq.nextval, т.к. это было уже сделано в родительском
 канале (т.е. дочерний канал хранимой процедуры использует
 значение последовательности, взятое из родительского канала).
SQL_скрипт tst.sql:
create or replace table test1(i bigint, val1 int);
create or replace sequence test1_seq start with 1 increment by 1;
```

```
create or replace table test2(i bigint, val2 int);
create or replace sequence test2_seq start with 1 increment by 1;
create or replace table test3(i1 bigint, i2 bigint, i bigint, val3
 int);
create or replace sequence test3_seq start with 1 increment by 1;

create or replace procedure test_prc() result int
code
  execute direct "insert into test3 values(test1_seq.currval,
 test2_seq.currval, test3_seq.nextval, 3);"; //
  return errcode(); //
end;

insert into test1 values(test1_seq.nextval,?);
1
insert into test2 values(test2_seq.nextval, ?);
2
execute test_prc();

select * from test1;
select * from test2;
select * from test3;

Выполнение sql-скрипта:
>inl -u SYSTEM/MANAGER8  -f tst

 I      VAL1
|     1|     1|

 I      VAL2
|     1|     2|

 I1       I2        I         VAL3
|      1|       1|       1|      3|
```

Примеры

```
create or replace sequence INT_PLUS_C as integer start with 7
 increment by 2 minvalue 7 maxvalue 15 cycle;

create or replace sequence INT_MINUS_C as integer start with 20
 increment by -3 minvalue 7 maxvalue 20 cycle;
```

Удаление последовательности (DROP SEQUENCE)

Функция

Определение оператора удаления общей или личной последовательности.

Спецификация

[1] <удаление последовательности>::=
DROP [PUBLIC] SEQUENCE [<имя схемы>.]<имя последовательности>

Общие правила

1) Для удаления PUBLIC-последовательности необходимы привилегии DBA.
2) Личную последовательность может удалить только ее создатель.

Модификация последовательности (ALTER SEQUENCE)

Функция

Определение оператора модификации общей или личной последовательности.

Спецификация

[1] <модификация последовательности>::=
ALTER [PUBLIC] SEQUENCE [<имя схемы>.]<имя последовательности>
[{START | RESTART} WITH <начальное значение>]
[CURRENT VALUE <текущее значение>]
[INCREMENT BY <шаг>]
[MAXVALUE <верхняя граница> | NOMAXVALUE | NO MAXVALUE]
[MINVALUE <нижняя граница> | NOMINVALUE| NO MINVALUE]
[CYCLE | NO CYCLE]
[2] <текущее значение>::= <целочисленный литерал>

Синтаксические правила

1) <Имя последовательности> должно задавать существующую последовательность.
2) Описание опций START WITH, INCREMENT BY, MAXVALUE, NO MAXVALUE, MINVALUE, NO MINVALUE, CYCLE, NO CYCLE приведено в пункте «Создание последовательности».
3) Опции START и RESTART являются синонимами.
4) Опции NO MAXVALUE и NOMAXVALUE являются синонимами.
5) Опции NO MINVALUE и NOMINVALUE являются синонимами.
6) Одновременное указание опций START|RESTART WITH и CURRENT VALUE не допускается.
7) <Начальное значение>, <шаг>, <верхняя граница> и <нижняя граница> – целочисленные значения в диапазоне от -9 223 372 036 854 775 808 до +9 223 372 036 854 775 807 (тип данных BIGINT).
8) Величины <начального значения>, <верхней границы> и <нижней границы> не должны логически противоречить друг другу.
9) <Шаг> последовательности должен лежать в интервале от -72057594037927936 до 72057594037927935.

Общие правила

1) Для изменения PUBLIC-последовательности необходим уровень прав DBA.
2) Изменять личную последовательность может только ее создатель при наличии уровня прав RESOURCE.

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

3) Если задана опция START (RESTART), то следующим выбираемым значением последовательности будет <начальное значение>. Оно не сравнивается с ранее выбранными значениями этой же последовательности на предмет непротиворечивости (например, дублирование номера последовательности).

```
create sequence my_seq start with 10;
select my_seq.nextval;

|         10|

alter sequence my_seq start with 6;
select my_seq.nextval;

|          6|

alter sequence my_seq increment by 2;
select my_seq.nextval;

|          8|

select my_seq.nextval;

|         10|
```

4) Опция CURRENT VALUE устанавливает новое текущее значение последовательности, не затрагивая прежнее текущее и стартовое значение, установленное при создании или модификации последовательности с помощью опции START (RESTART). Новое текущее значение должно находится в границах созданной последовательности.

Примечание
Опция CURRENT VALUE применяется, как правило, для восстановления текущего значения последовательности после миграции БД.

Примеры

```
create or replace sequence INT_MINUS_C as integer start with 20
 increment by -3 minvalue 7 maxvalue 20 cycle;
select INT_MINUS_C.NEXTVAL from $$$SYSRL where rowid < 5;

|         20|
|         17|
|         14|
|         11|

alter sequence INT_MINUS_C current value 10;
select INT_MINUS_C.NEXTVAL from $$$SYSRL where rowid < 5;

|         10|
|          7|
```

```
|         20|
|         17|

create or replace sequence A_SEQ start with 1;
select A_SEQ.NEXTVAL;
alter sequence A_SEQ increment by 100;
select A_SEQ.NEXTVAL;
alter sequence A_SEQ increment by -111 maxvalue 1000 minvalue
 -100000;
select A_SEQ.NEXTVAL;
alter sequence A_SEQ increment by -1000 nomaxvalue nominvalue;
select A_SEQ.NEXTVAL;
```

Выбор значения последовательности (CURRVAL/ NEXTVAL)

Функция

Определение оператора получения текущего или следующего значения последовательности.

Спецификация

[1] <выбор значения последовательности>::= <имя последовательности>.{NEXTVAL | CURRVAL}

Синтаксические правила

1) Опция NEXTVAL предоставляет следующее значение последовательности.
2) Опция CURRVAL предоставляет текущее значение последовательности.
3) Конструкция <выбор значения последовательности> может использоваться:
   - в SELECT-запросах в качестве псевдостолбца и в условии WHERE;
   - в UPDATE-запросах в конструкции SET и в условии WHERE;
   - в INSERT-запросах в конструкции VALUES;
   - в DELETE-запросах в конструкции WHERE.

```
create table "Платежные поручения" ("Номер ПП" char (8),
  "Дата" date, "Получатель" varchar(50), "Сумма" decimal);
CREATE PUBLIC SEQUENCE "Номера платежных поручений"
START WITH 1 INCREMENT BY 1;
select "Номера платежных поручений".nextval;

insert into "Платежные поручения"
  ("Номер ПП", "Дата", "Получатель", "Сумма") values
  (to_char(sysdate,'yyyy') ||
   cast("Номера платежных поручений".currval as char(4)),
   sysdate, 'Администрация Советского района', 4597.0);
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

4) В случае если в SQL-запросе использованы ключевые слова NEXTVAL и CURRVAL, а последовательность не найдена, предполагается, что NEXTVAL и CURRVAL – имена столбцов.

```
create or replace table test ("NEXTVAL" int,"CURRVAL" int);
create or replace sequence test;
insert into test("CURRVAL","NEXTVAL") values (100,200);

select test."CURRVAL",test."NEXTVAL" from test;
! result (1,1):

drop sequence test;
select test."CURRVAL",test."NEXTVAL" from test;
! result (100,200):

select "CURRVAL","NEXTVAL" from test;
! result (100,200):
```

5) Запрещено использование NEXTVAL в запросе с UNION и/или DISTINCT, но разрешено использование с UNION ALL.

```
create or replace sequence test;
SELECT test.NEXTVAL UNION ALL SELECT test.NEXTVAL;
!result - 2 records (1) & (2):

SELECT test.NEXTVAL UNION SELECT test.NEXTVAL;
!error 1102:

SELECT DISTINCT test.NEXTVAL FROM AUTO;
!error 1102:

drop sequence test;
```

Общие правила

1) Перед первым после открытия соединения с БД получением CURRVAL-значения необходимо установить следующее значение последовательности, т.е. получить NEXTVAL-значение. Данная операция должна выполняться для каждого соединения.
2) Все NEXTVAL-значения, запрошенные в одном SQL-запросе, возвращают одно и то же значение для одной строки ответа. Для каждой следующей строки ответа значение NEXTVAL будет другим. Для получения действительно следующего значения надо использовать новый SQL-запрос.

```
select SEQ.NEXTVAL, SEQ.NEXTVAL, SEQ.NEXTVAL;

|  10| 10| 10|

select SEQ.NEXTVAL;

| 11|
```

3) Запрещено использование NEXTVAL в <WHERE-спецификации>.

4) В <WHERE-спецификации> используется то значение CURRVAL, которое было установлено перед выполнением запроса. Если значение CURRVAL не было установлено, будет выдан код завершения 1102 («Текущее значение последовательности не установлено»). Т.е. если используется запрос вида

```
select "TEST".NEXTVAL, "TEST".CURRVAL from AUTO where rowid <
 "TEST".CURRVAL;
```

то надо понимать, что значения "TEST".CURRVAL в секциях SELECT и WHERE разные: первое значение в секции SELECT будет отличаться от значения в секции WHERE на величину шага последовательности.

Пример

```
DROP SEQUENCE "TEST";
CREATE SEQUENCE "TEST";
!Код завершения  1102:
select rowid,"TEST".NEXTVAL, "TEST".CURRVAL
  from AUTO where rowid < "TEST".CURRVAL;

!3 записи с  rowid 1,2,3:
select rowid,"TEST".NEXTVAL, "TEST".CURRVAL
  from AUTO where rowid < 4;

 ROWID
 -----
| 1| 1| 1|
| 2| 2| 2|
| 3| 3| 3|

!2 записи с  rowid 1,2:
select rowid,"TEST".NEXTVAL, "TEST".CURRVAL
  from AUTO where rowid < "TEST".CURRVAL;

 ROWID
 -----
| 1| 4| 4|
| 2| 5| 5|

T.e. после второго SELECT-запроса значение CURRVAL стало равно 3 и
 именно это
 значение было использовано в условии WHERE последнего (третьего)
 запроса.
```

Генерация временной последовательности (SELECT LEVEL)

Функция

Определение оператора генерации временной числовой последовательности в заданном диапазоне.

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

Спецификация

[1] <генерация временной последовательности>::=
SELECT LEVEL FROM (SELECT 1) CONNECT BY LEVEL < значение

Синтаксические правила

1) Значение – целочисленное значение в диапазоне от 1 до 500 млн.

Общие правила

1) Конструкция возвращает один столбец с именем LEVEL, содержащий все значения заданной последовательности.
2) Генерируемая временная последовательность доступна только во время выполнения SQL-запроса, использующего эту последовательность.

Примеры

1) Страховая компания выпустила очередную серию из 100000 страховых полисов, начинающуюся с номера 200000. Информация о всех номерах страховых полисах заносится в таблицу med_polis (с помощью двух запросов) и используется затем для:
   - выдачи очередного полиса (номер берется из числа свободных);
   - аннулирования полиса;
   - проверки подлинности предъявленного полиса;
   - определения количество не выданных полисов.

```
create or replace table med_polis(id_polis int, "ФИО" varchar(30),
 "Дата выдачи" date) as select level, cast '' as varchar, cast
 null as date from (select 1) connect by level < 1000001;

update med_polis set id_polis = id_polis +199999;
```

2)

```
select model, personid
  from auto,
       (SELECT LEVEL as LV FROM (SELECT 1) CONNECT BY LEVEL < 31)
 as counter
 where auto.personid=counter.LV(+)
   and counter.LV is NULL limit 10;
```

Триггеры

Создание триггера (CREATE TRIGGER)

Функция

Определение оператора создания триггера.

Спецификация

[1] <создание триггера>::=
CREATE [IF NOT EXISTS | OR REPLACE] TRIGGER
[<имя схемы>.]<имя триггера>

{<триггер на обработку данных> | <триггер на системные события>}

[2] <триггер на обработку данных>::=
{BEFORE | AFTER | INSTEAD OF} <операция> [OR …]
ON [<имя схемы>.]<имя таблицы>
[FOR EACH {ROW|STATEMENT}]
[OLD [AS] <псевдоним для старой записи>]
[NEW [AS] <псевдоним для новой записи>]
EXECUTE [FOR DEBUG] <тело триггера>;

[3] <операция>::=
INSERT | DELETE | UPDATE [OF <имя столбца>[, …]];

[4] <триггер на системные события>::=
{AFTER LOGON | BEFORE LOGOFF}
ON {<имя пользователя>|DATABASE}

[5] <псевдоним для старой записи>::= <идентификатор>

[6] <псевдоним для новой записи>::= <идентификатор>

[7] <тело триггера>::= текст триггера

Синтаксические правила

1) <Имя таблицы> должно задавать объект (базовую таблицу или представление), для которого создается триггер.
2) <Имя схемы> задает имя схемы, владельцем которой является текущий пользователь. Создавать триггеры в «чужих» схемах запрещено.
3) Опция OR REPLACE заставляет удалять существующий в БД триггер и создавать его под тем же именем, но с указанными параметрами.
4) Опция IF NOT EXISTS отменяет выполнение оператора, если указанный триггер уже существует в БД.
5) Одновременное использование опций IF NOT EXISTS и OR REPLACE запрещено.
6) <Операция> задает операцию по обработке данных таблицы, при выполнении которой должен срабатывать триггер:
   - INSERT – при добавлении данных;
   - DELETE – при удалении данных;
   - UPDATE – при корректировке данных.
7) Если указана операция UPDATE без конструкции OF, триггер будет срабатывать при корректировке любого поля записи таблицы.
8) <Имя столбца> должно ссылаться на столбец в <имени таблицы>.
9) Если указана операция UPDATE OF <столбец>[, …], триггер будет срабатывать при корректировке только перечисленных столбцов записи таблицы.
10) Опции BEFORE, AFTER, INSTEAD OF определяют момент срабатывания триггера при выполнении операций обработки данных:
    - BEFORE – перед выполнением <операции>;
    - AFTER – после выполнения <операции>;
    - INSTEAD OF – взамен выполнения <операции>.
11) Конструкция INSTEAD OF применима только для базовых таблиц (не представлений).
12) Конструкция INSTEAD OF не применима для таблиц, работающих в циклическом режиме, поскольку такие триггеры препятствуют удалению записей (см. оператор ALTER TABLE … SET RECORDS LIMIT пункта «Модификация таблицы»).

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

13) Конструкция INSTEAD OF допустима как для режима FOR EACH ROW, так и для FOR STATEMENT. Она позволяет выполнить проверку достоверности значений, которые должны быть добавлены, обновлены или удалены, и, в случае их достоверности, выполняет соответствующую операцию. Например, при обработке заказа на некоторый товар триггер INSTEAD OF UPDATE может проверить, есть ли в таблице заказов строка с требуемым товаром. Если да, сумма заказа будет увеличена (операция UPDATE), если нет – запись будет добавлена (операция INSERT).
14) Если на операцию создано несколько триггеров **одного времени действия**, они будут выполняться в порядке их системных номеров, который обычно, но не всегда, совпадает с порядком создания триггеров.
15) Конструкция FOR EACH... определяет сферу действия триггера:
    - ROW – для каждой записи. Триггер будет срабатывать во время выполнения <операции> при обработке каждой записи таблицы. Возврат триггером значения TRUE разрешает, а FALSE – запрещает выполнение операции только для текущей записи (имеет смысл применительно к BEFORE);
    - STATEMENT – для каждого оператора. Триггер будет срабатывать **один раз** при выполнении оператора, соответствующего заданной <операции>. Возврат триггером BEFORE ... FOR EACH STATEMENT значения FALSE отменяет действие всей операции.
16) Если конструкция FOR EACH... не задана, по умолчанию используется FOR EACH STATEMENT.
17) Если один из BEFORE-триггеров запретил операцию, то последующие AFTER-триггеры выполняться не будут.

Примечание

Например, в операторе UPDATE AUTO SET MAKE ='BMW' where PERSONID IN (SELECT PERSONID FROM PERSON WHERE SALARY BETWEEN 2000 AND 50000); триггер на UPDATE при указании FOR EACH ROW выполнится столько раз, сколько записей выбирается по условию WHERE; при указании FOR EACH STATEMENT – только один раз (в случае выполнения условия WHERE).

18) <Псевдоним для старой записи> задает префикс, используемый для обращения к старым значениям столбца. Если не задан, по умолчанию используется OLD.
19) <Псевдоним для новой записи> задает префикс, используемый для обращения к новым значениям столбца. Если не задан, по умолчанию используется NEW.

```
update auto set old.color=new.color where personid=256;
update auto set "Старое".color="Новое".color where personid=256;
```

20) Понятия «старое» и «новое» значение столбца применимы только для операции UPDATE, независимо от времени действия триггера.
21) Для операции INSERT имеет смысл только «новое» значение столбца.
22) Для операции DELETE имеет смысл только «старое» значение столбца.
23) Запрещено создавать триггер на таблицу другого пользователя.
24) Имя триггера должно быть уникально в пределах <имя схемы>.
25) Опции <триггер на системные события>:
    - AFTER LOGON – активизирует триггер после успешного открытия канала с СУБД (после установления соединения с СУБД). Если триггер возвращает ошибку, то

для всех пользователей, за исключением LINTER_SYSTEM_USER, выполняется очистка канала;

- BEFORE LOGOFF – активизирует триггер перед закрытием канала (перед закрытием соединения по инициативе пользователя или при выполнении команд KILL, CKIL интерфейса нижнего уровня). Для курсорных запросов эти триггеры выполнять не следует;
- DATABASE – активизирует триггер при открытии/закрытии любого канала;
- <имя пользователя> – активизирует триггер при открытии/закрытии канала для указанного пользователя.

26) Триггеры AFTER LOGON/BEFORE LOGOFF может создавать/модифицировать/удалять только создатель БД.

27) Тело триггера (конструкция EXECUTE <тело триггера>) должно быть написано на процедурном языке СУБД ЛИНТЕР (см. документ «Процедурный язык», раздел «Элементы языка»).

28) Опция FOR DEBUG позволяет выполнять отладку триггера с помощью отладчика триггеров и хранимых процедур СУБД ЛИНТЕР.

Общие правила

1) На каждую операцию можно создавать не более 255 триггеров.

2) Для хранения тела оттранслированной процедуры, возвращающей тип данных «курсор», в БД должна существовать системная таблица $$$PRCD.

3) Порядок вызова триггеров с разными значениями <время действия> следующий:
   - BEFORE EACH STATEMENT;
   - BEFORE EACH ROW;
   - AFTER EACH ROW;
   - AFTER EACH STATEMENT.

4) Триггер BEFORE EACH ROW может изменить значение поля записи, помещаемое в БД, для операторов INSERT и UPDATE.

5) Возврат триггером BEFORE ... FOR EACH ROW значения FALSE означает отмену действия операции только для текущей записи.

6) В режиме FOR STATEMENT перед обработкой триггеров на условие BEFORE STATEMENT происходит выполнение всех триггеров на условие INSTEAD OF. Если был выполнен хотя бы один триггер INSTEAD OF (даже если он вернул ошибку), дальнейшая обработка запроса и всех его триггеров прекращается.

7) В режиме FOR EACH ROW перед обработкой триггеров на условие BEFORE происходит выполнение всех триггеров на условие INSTEAD OF. Если был выполнен хотя бы один триггер INSTEAD OF (даже если он вернул ошибку), обработка записи и триггеров FOR EACH ROW прекращается, происходит переход к обработке следующей записи. При этом триггеры на условие FOR STATEMENT (BEFORE и AFTER) выполняются в обычном режиме.

8) В момент активизации <триггера на системные события> канал получает статус «занят», что дает возможность триггеру выполняться без помех.

9) В триггере на системные события все операторы выполняются от имени пользователя, который устанавливает/разрывает соединение, в триггере на обработку

данных все операторы выполняются от имени пользователя, который выполняет операцию.

10) При наличии нескольких <триггеров на системные события> сначала выполняются триггеры для всей БД, затем – для указанного пользователя.

```
create or replace table test (i int, ch char( 40 ));
create or replace trigger test1 after logon on SYSTEM
execute code
  execute direct "insert into test values (1,'After logon for user
 1');";
  return TRUE;
end;

create or replace trigger test2 after logon on DATABASE
execute code
  execute direct
    "insert into test values (1,'After logon for database 1');";
  return TRUE;
end;

create or replace trigger test3 after logon on SYSTEM
execute code
  execute direct
    "insert into test values (1,'After logon for user 2');";
  return TRUE;
end;

create or replace trigger test4 after logon on DATABASE
execute code
  execute direct
    "insert into test values (1,'After logon for database 2');";
  return TRUE;
end;

username SYSTEM/MANAGER8
select * from test;

I           CH
 -           --
|          1|After logon for database 1              |
|          1|After logon for database 2              |
|          1|After logon for user 1                  |
|          1|After logon for user 2                  |
```

11) Если один из триггеров вернул ошибку, остальные триггеры не выполняются.

```
create or replace table test (i int, ch char( 40 ));
create or replace trigger test1 after logon on SYSTEM
execute code
```

```
  execute direct
    "insert into test values (1,'After logon for user 1');";
  return FALSE;
end;

create or replace trigger test2 after logon on DATABASE
execute code
  execute direct
    "insert into test values (1,'After logon for database 1');";
  return FALSE;
end;

create or replace trigger test3 after logon on SYSTEM
execute code
  execute direct
    "insert into test values (1,'After logon for user 2');";
  return FALSE;
end;

create or replace trigger test4 after logon on DATABASE
execute code
  execute direct
    "insert into test values (1,'After logon for database 2');";
  return FALSE;
end;

username SYSTEM/MANAGER8
// триггер test1 вернул FALSE, поэтому триггеры test2, test3,
 test4
// выполняться не будут, канал для пользователя SYSTEM открыт
 (т.к. это
// создатель БД – создателю БД нельзя случайно запретить
 подсоединение к БД
// созданием LOGON-триггера, возвращающего значение FALSE)

create or replace user TTT identified by '12345678';
username TTT/12345678
// ошибка – нет привилегий 1022 (возвращается в случае возврата
// LOGON-триггером значения FALSE для любого пользователя, кроме
 создателя БД)
// триггер test1 вернул FALSE, поэтому триггеры test1, test2,
 test3, test4 выполняться не будут

username SYSTEM/MANAGER8
select * from test;
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
I            CH
 -            --
|           1|After logon for database 1                |
|           1|After logon for database 1                |
```

12) По окончании работы триггеров статус канала восстанавливается.

13) В случае занятости транслятора процедурного языка ядро СУБД ЛИНТЕР будет ожидать его освобождения.

Пример

```
CREATE TRIGGER UPDPERSON AFTER UPDATE ON PERSON FOR EACH ROW
 EXECUTE
CODE
  if old.personid = new.personid then
    execute direct "insert into journal values('PERSON','UPDATE','"
 + itoa(old.personid) + ",sysdate, 'personid not changed');"; //
    return true; //
  endif
  execute direct "insert into journal values('PERSON','UPDATE','"
 + itoa(old.personid) + ",sysdate, 'set personid
 ='+itoa(new.personid)+ ' - cause update for   auto');"; //
  execute direct "update auto set personid=" + itoa(new.personid)
 + " where personid = " + itoa(old.personid) + ";"; //
END;
```

Компиляция триггера (REBUILD TRIGGER)

Функция

Определение оператора компиляции существующего в БД триггера. Необходимость в перекомпиляции триггера возникает в случае, если в теле триггера используются претранслированные запросы, а указанные в них таблицы были пересозданы или была изменена их структура.

Спецификация

[1] <компиляция триггера>::=
REBUILD TRIGGER [<имя схемы>.]<имя триггера>

Синтаксические правила

1) <Имя триггера> должно ссылаться на существующий в БД триггер.

Общие правила

1) Команда доступна только владельцу триггера.
2) Для выполнения команды надо иметь привилегию RESOURCE.

Пример

```
create or replace table test(ch char(10));
create or replace table test_audit(ch_new char(10), ch_old
 char(10), action char(10));
```

```
create or replace trigger test_trig before insert on test for each
 row execute for debug
code
  if ( inserting ) then
    execute "insert into test_audit values (?, '', 'inserting');"
 using new.ch; //
  endif; //
  return true; //
exceptions
  when all then
    return false;
end;
//проверка работы триггера
insert into test values('value 1');
select * from test;
select * from test_audit;

//удалим таблицу, используемую в претранслированном запросе
drop table test_audit;
insert into test values('value 2');
select * from test; // запись со значением 'value 2' отсутствует,
 так как в
 триггере произошло исключение (отсутствует таблица test_audit) и
 в блоке
 обработки исключений было возвращено значение false, отменяющее
 основную
 операцию
select * from test_audit; // несуществующая таблица (test_audit)

//пересоздадим таблицу test_audit
create or replace table test2(ind int);
create or replace table test_audit(ch_new char(10), ch_old
 char(10), action char(10));
insert into test values('value 2');
select * from test; // запись со значением 'value 2' отсутствует,
 так как был пересоздан объект, используемый в претранслируемом
 запросе
select * from test_audit;

rebuild trigger test_trig; // пересоздадим триггер
insert into test values('value 2');
select * from test;// запись со значением 'value 2' добавлена
select * from test_audit;// запись со значением 'value 2'
 добавлена
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

Удаление триггера (DROP TRIGGER)

Функция

Определение оператора удаления триггера.

Спецификация

[1] <удаление триггера>::=
DROP TRIGGER [<имя схемы>.]<имя триггера>

Общие правила

1) Удалить триггер может только его владелец.
2) Администратор БД имеет возможность удалить одновременно все объекты (в том числе и триггеры) некоторого пользователя с помощью команды `DROP USER <имя пользователя> CASCADE`.

Изменение состояния триггера (ALTER TRIGGER)

Функция

Определение оператора изменения состояния триггера.

Спецификация

[1] <изменение состояния триггера>::=
ALTER TRIGGER [<имя схемы>.]<имя триггера> {ENABLE | DISABLE}

Общие правила

1) Опция `ENABLE` переводит триггер в активное состояние.
2) Опция `DISABLE` переводит триггер в неактивное состояние.
3) При изменении состояния триггера его код не меняется.
4) Право изменить состояние триггера имеет только его владелец.

Удаление текста триггера (ALTER TRIGGER DROP TEXT)

Функция

Определение оператора удаления исходного текста триггера.

Спецификация

[1] <удаление текста триггера>::=
ALTER TRIGGER [<имя схемы>.]<имя триггера> DROP SOURCE TEXT

Общие правила

1) Команда очищает BLOB-значение, хранящее исходный текст соответствующего триггера, в результате чего исходный текст триггера нельзя просматривать, редактировать, отлаживать (с помощью отладчика хранимых процедур и триггеров) и повторно транслировать. В остальном поведение триггера остается неизменным – его можно удалять, изменять состояние, и он будет срабатывать при наступлении соответствующего события.

Глобальные переменные процедурного языка

В СУБД ЛИНТЕР поддерживаются только глобальные переменные процедурного языка простого типа.

Создание глобальной переменной простого типа (CREATE VARIABLE)

Функция

Определение и инициализация глобальной переменной процедурного языка простого типа.

Спецификация

[1] <создание/модификация глобальной переменной простого типа>::=
CREATE [IF NOT EXISTS | OR REPLACE]
VARIABLE [<имя схемы>.]<имя глобальной переменной>
<тип данных переменной>[=<значение по умолчанию>]
[2] <имя глобальной переменной>::= <идентификатор>
[3] <тип данных переменной>::=
CHAR | VARCHAR | INTEGER | SMALLINT | BIGINT | REAL | DOUBLE | DATE | DECIMAL | NUMERIC | NCHAR | NCHAR VARYING | BOOLEAN | BYTE | VARBYTE
[4] <значение по умолчанию>::= <литерал>

Синтаксические правила

1) <Имя глобальной переменной> должно содержать буквы только латинского алфавита, цифры, символ подчеркивания и символы: @, #, %, ?, ~.
2) При создании новой глобальной переменной <имя глобальной переменной> должно быть уникальным среди имен всех глобальных переменных в указанной схеме.
3) Опция OR REPLACE заставляет удалять существующую в БД глобальную переменную и создавать её под тем же именем, возможно, с другим типом данных или значением по умолчанию.
4) Опция IF NOT EXISTS отменяет выполнение оператора, если указанная глобальная переменная уже существует в БД.

Общие правила

1) Глобальные переменные предназначены для использования в процедурном языке СУБД ЛИНТЕР (см. документ «Процедурный язык», раздел «Глобальные переменные хранимых процедур»).
2) Созданная глобальная переменная:
   - включается в состав объектов указанной схемы;
   - либо включается в состав текущей схемы (если выполнен SQL-оператор SET SCHEMA) и <имя схемы> не задано;
   - либо её владельцем назначается пользователь, от имени которого выполнена команда создания глобальной переменной.
3) Для создания/модификации глобальной переменной необходима привилегия RESOURCE.
4) Максимальная длина значения глобальной переменной типа BYTE/VARBYTE, а также переменной символьного типа 4000 байт.

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

5) Значения символьных глобальных переменных (CHAR/VARCHAR) записывается в текущей кодировке соединения, а NCHAR/NCHAR VARYING – UTF-16.
6) Созданная/модифицированная глобальная переменная без указания <имени схемы> доступна из хранимых процедур СУБД ЛИНТЕР, созданных в той же схеме.
7) Для доступа к «чужой» глобальной переменной необходимо указывать полное квалификационное имя: <имя схемы>.<имя глобальной переменной>.

Примеры

1) Создание глобальной переменной:

```
create variable company char='RELEX';
create variable department int=125;
```

2) Создание или модификация существующей глобальной переменной:

```
create or replace variable department smallint=10;
```

3) Получить текущий список глобальных переменных с их схемами:

```
select $$$usr.$$$s34,
       $$$glbvars.$$$name
  from $$$glbvars, $$$usr
 where $$$glbvars.$$$owner=$$$usr.$$$s31
   and $$$usr.$$$s32=0;
```

4) Получить список символьных (char) и целочисленных (smallint) глобальных переменных:

```
select $$$glbvars.$$$name, getstr($$$glbvars.$$$info,10,16)
  from $$$glbvars
 where getbyteb($$$glbvars.$$$info,1)=cast 1 as byte
 union
select $$$glbvars.$$$name, cast getword($$$glbvars.$$$info,10) as
 char
  from $$$glbvars
 where getbyteb($$$glbvars.$$$info,1)=cast 2 as byte
   and getword($$$glbvars.$$$info,4) = 2;
```

или

```
select trim($$$glbvars.$$$name),
  case
    when getbyteb($$$glbvars.$$$info,1)= cast 1 as byte then
      getstr($$$glbvars.$$$info,10,16)
    when getbyteb($$$glbvars.$$$info,1)= cast 2 as byte and
      getword($$$glbvars.$$$info,4) = 2 then
    cast getword($$$glbvars.$$$info,10) as char
  end
  from $$$glbvars;
```

Удаление глобальной переменной (DROP VARIABLE)

Функция

Удаление глобальной переменной процедурного языка.

Спецификация

[1] <удаление глобальной переменной>::=
DROP VARIABLE [<имя схемы>.]<имя глобальной переменной>

Синтаксические правила

1) Глобальная переменная с указанным <именем глобальной переменной> должна существовать в БД.

Общие правила

1) Удалить глобальную переменную может только её владелец (либо пользователь с правами DBA при каскадном удалении владельца переменной) при наличии прав RESOURCE.
2) После удаления глобальной переменной все ссылки на неё во всех используемых хранимых процедурах должны быть удалены, а сами хранимые процедуры заново оттранслированы (автоматически это не выполняется).

События

Создание события (CREATE EVENT)

Функция

Определение оператора объявления нового события.

Спецификация

[1] <создание события>::=
CREATE [IF NOT EXISTS | OR REPLACE] [GLOBAL] [PRIVATE]
EVENT [<имя схемы>.]<имя события> [AUTORESET] [WITHOUT SOURCE]
[AS { <запрос выборки> | [TRANSACTION]
{DELETE | UPDATE | INSERT} [, ...] ON [<имя схемы>.]<имя таблицы>}
| TIME {CURRENT | (<символьное выражение>[, <символьный формат>])}
[CYCLE количество единиц {SECOND | MINUTE | HOUR | DAY}]]
[EXECUTE <имя хранимой процедуры> (список параметров) [AS CURRENT_USER]]
[DISABLE]

Синтаксические правила

1) <Имя события> должно иметь уникальное в БД имя среди имен событий в пределах <имя схемы>.
2) <Имя таблицы> должно ссылаться на базовую таблицу.
3) Если <имя схемы> не указано, то подразумевается схема пользователя, который подает запрос.
4) Опция OR REPLACE заставляет удалять существующее в БД событие и создавать его под тем же именем, но с другими параметрами.
5) Опция IF NOT EXISTS отменяет выполнение оператора, если указанное событие уже существует в БД.
6) Одновременное использование опций IF NOT EXISTS и OR REPLACE запрещено.
7) Если задан модификатор GLOBAL, будет создано хранимое событие, запоминаемое в системной таблице $$$EVENTS. По умолчанию создается несохраняемое событие (то есть событие будет известно только в текущем сеансе работы СУБД ЛИНТЕР).

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

8) Созданное несохраняемое событие существует до тех пор, пока не будет уничтожено командой DROP EVENT или не произойдет завершение работы ядра СУБД ЛИНТЕР.

9) Если задан модификатор PRIVATE, то создается личное событие, владельцем которого является создавший его пользователь БД. Для доступа к такому событию другие пользователи БД должны указывать полное имя события в виде «<имя схемы>.<имя события>». Если этот модификатор не задан, к событию другого пользователя можно обращаться просто по имени события. В ситуации, когда пользователь имеет доступ к событию другого пользователя с тем же именем, что и его личное событие, при указании только имени события (без уточнения его владельца) операция будет проведена над личным событием пользователя.

10) <Запрос выборки> определяет запрос, который должен автоматически выполняться при реальном изменении любой из таблиц, участвующих в этом запросе. СУБД ЛИНТЕР будет устанавливать данное событие, если <запрос выборки> вернул непустое множество ответов.

```
create event "Select_All_From_Auto_Person" as select * from auto,
 person where auto.personid=person.personid;
create event MOD_AUTO as delete, update on auto;
```

11) Для создания события необходимы права чтения на все используемые таблицы (уровень прав CONNECT).

12) Если событие относится к объекту, хранящемуся в удаленном узле распределенной БД, то данное событие будет видно только в программе, создавшей это событие. На удаленном узле оно будет неизвестно.

13) Событие, созданное с модификатором AUTORESET, автоматически сбрасывается (удаляется) из очереди событий после рассылки оповещения (уведомления) о наступлении события всем пользователям, ожидающим этого события. Например, если пользователь создал событие с модификатором AUTORESET и затем оповестил о его наступлении с помощью команды SET EVENT, то оно будет сброшено после рассылки СУБД уведомления о наступлении события всем пользователя, ожидающим этого события по команде WAIT EVENT.

14) Модификатор AUTORESET можно задавать для всех типов событий, кроме SELECT.

15) Модификатор WITHOUT SOURCE действует только на хранимые события, сохраняемые в системной таблице $$$EVENTS, то есть имеющие модификатор GLOBAL. Для всех остальных событий он игнорируется. Он запрещает сохранение текста запроса на создание события в системной таблице $$$EVENTS, вследствие чего отключенное событие не может быть включено вновь командой «ALTER EVENT … ENABLE;».

16) Если перед опциями INSERT, UPDATE или DELETE стоит модификатор TRANSACTION, это означает, что событие будет наступать не в момент подачи запросов INSERT, UPDATE или DELETE, а в момент подачи COMMIT для транзакции, включающей соответствующие запросы.

17) Модификатор TIME заставляет генерировать событие в определённое время (событие по таймеру). Если в качестве времени указан модификатор CURRENT, то событие наступит сразу после его создания по данной команде.

18) <Символьное выражение> и <символьный формат> задают время генерации события по таймеру. Синтаксис этих параметров аналогичен параметрам функции TO_DATE (см. пункт «Преобразование символьного представления типа «дата-время» во внутреннее»). Если в качестве времени наступления события указано прошедшее время, событие наступит сразу же после создания.

19) Модификатор CYCLE задает создание повторяющегося события, то есть события, которое будет постоянно генерироваться через определённый промежуток времени

после своего первого наступления. Частота генерации события задается с помощью двух параметров:

- <количество единиц> – количество единиц времени (значение от 1 до 65535);
- {SECOND | MINUTE | HOUR | DAY} – величина единицы времени.

20) Модификатор CYCLE можно указать только в случае наличия модификатора AUTORESET.
21) Модификатор AUTORESET для событий по таймеру можно указать только при наличии модификатора CYCLE.
22) Модификатор EXECUTE заставляет выполнять при наступлении создаваемого события указанную хранимую процедуру с конкретными аргументами. Для события на SELECT-запрос модификатор EXECUTE выполняет процедуру в том случае, если при проверке события оно установлено (SELECT-запрос вернул непустую выборку), а при предыдущей проверке оно не было установлено (SELECT-запрос вернул пустую выборку) или текущая проверка – первая сразу после создания события. Если при последовательных проверках события SELECT-запрос каждый раз возвращает непустую выборку, то повторные вызовы процедуры производиться не будут.
23) Если задан модификатор AS CURRENT_USER, то процедура будет выполнена от имени того пользователя, действия которого вызвали наступление события, в противном случае процедура будет выполнена от имени пользователя, создавшего событие.
24) Модификатор AS CURRENT_USER нельзя задавать для событий по таймеру.
25) На время выполнения процедуры, активизируемой при наступлении события, генерирование порождающих её вызов событий не блокируется, поэтому вполне возможна ситуация, когда событие наступает часто, а процедура не успевает выполниться в промежуток между двумя наступившими событиями. В этом случае игнорируются все необработанные процедурой события, кроме последнего. Например, если за время выполнения процедуры, активизированной по наступлению события, было сгенерировано 100 таких же событий, то данная процедура будет вызвана еще только один раз (даже после прекращения генерации вызывающих её активизацию событий).
26) Если задан модификатор DISABLE, событие сразу же после создания будет отключено. Этот модификатор можно задать только для хранимых событий, создаваемых с модификатором GLOBAL. Отключённое событие можно либо удалить, либо включить с помощью команды «ALTER EVENT … ENABLE;».

Пример

Для выполнения примера в БД должна быть создана системная таблица $$$EVENTS. Приведённые ниже запросы выполняются утилитой inl (см. документ «Командный интерфейс», раздел «Обработка SQL-операторов»).

1) Создаём таблицу для протоколирования событий и процедуру, которая будет выполнять протоколирование событий с периодичностью 5 секунд:

```
create or replace table test_proc_table(i int, dt date);
create or replace procedure test_proc(in i int) result int
code
  print(dtoa(sysdate(), "DD.MM.YYYY HH24:MI.SS.FF") + ", "+
 username() + ", " + itoa(i+10)); //
  execute direct "insert into test_proc_table (i,dt) values
 ("+itoa(i+10)+",sysdate);"; //
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
  sleep(5000); //
end;
```

2) Создаём отключённое событие по таймеру с цикличностью 1 сек.:

```
create or replace global event "TIME_EVENT" autoreset as time
 ('14.04.08 13:15', 'DD.MM.YY HH:MI') cycle 1 second execute
 test_proc(1000) disable;
```

3) Включаем событие:

```
alter event "TIME_EVENT" enable;
```

4) Наблюдаем на консоли СУБД ЛИНТЕР распечатку информации из периодически запускаемой процедуры. Также можно выполнить запрос:

```
select * from test_proc_table;
```

Получим ответ в виде:

```
 I            DT
 -            --
|       1010|16.05.2008:13:10:23.00|
|       1010|16.05.2008:13:10:28.00|
|       1010|16.05.2008:13:10:33.00|
|       1010|16.05.2008:13:10:38.00|
...
```

5) Убеждаемся, что процедура запускается через 5 секунд, хотя при создании события мы указали «cycle 1». Событие наступает действительно через каждую секунду, однако, поскольку процедура не закончила свою работу через 1 секунду, она в этот момент не запускается. Чтобы убедиться в том, что приведённое объяснение правильно, меняем в запросе на создание события строку «cycle 1» на «cycle 7» и заново повторяем пункты 2 – 4. Убеждаемся, что процедура запускается через каждые 7 секунд, как и должно быть:

```
 I            DT
 -            --
|       1010|16.05.2008:13:10:23.00|
|       1010|16.05.2008:13:10:28.00|
|       1010|16.05.2008:13:10:33.00|
|       1010|16.05.2008:13:10:38.00|
|       1010|16.05.2008:13:10:40.00|
|       1010|16.05.2008:13:10:47.00|
...
```

(жирным шрифтом выделены новые строки).

6) Удаляем событие:

```
drop event "TIME_EVENT";
```

Удаление события (DROP EVENT)

Функция

Определение оператора удаления события.

Спецификация

[1] <удаление события>::= DROP EVENT [<имя схемы>.]<имя события>

Общие правила

1) Удалить событие может его создатель или создатель БД.
2) Несохраняемое событие (т.е. не GLOBAL) автоматически удаляется при завершении работы ядра СУБД.
3) Нельзя удалить событие, которое ожидается каким-либо пользователем, подавшим команду `WAIT EVENT`.
4) При удалении события все ожидающие его каналы получают код завершения 1801 («`Событие было уничтожено`»).
5) Хранимое событие не удаляется при удалении таблицы, на которую оно создано, а только делается неактивным, и может снова быть сделано активным после пересоздания таблицы с таким же именем.

Отключение события (ALTER EVENT DISABLE)

Функция

Определение оператора отключения события.

Спецификация

[1] <отмена события>::=
ALTER EVENT [<имя схемы>.]<имя события> DISABLE

Общие правила

1) Для нехранимого события данная команда аналогична команде `DROP EVENT`.
2) Отключить событие может либо его создатель, либо создатель БД.
3) Отключаемое событие удаляется из СУБД ЛИНТЕР (как при выполнении команды `DROP EVENT`), но информация о нем сохраняется в системной таблице `$$$EVENTS`, поэтому оно впоследствии может быть включено или удалено отдельной командой.

Включение события (ALTER EVENT ENABLE)

Функция

Определение оператора включения события.

Спецификация

[1] <включение события>::=
ALTER EVENT [<имя схемы>.]<имя события> ENABLE

Общие правила

1) `<Имя события>` должно ссылаться на хранимое событие.
2) Включить событие может либо его создатель, либо создатель БД.
3) Текст запроса на создание события извлекается из системной таблицы `$$$EVENTS`, после чего событие пересоздаётся с тем же самым идентификатором.
4) Если текст запроса на создание события отсутствует (событие было создано с модификатором `WITHOUT SOURCE`), включить такое событие невозможно (его можно только удалить).

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

Оповещение о событии (SET EVENT)

Функция

Определение оператора включения события в очередь событий.

Спецификация

[1] <оповещение о событии>::=
SET EVENT [<имя схемы>.]<имя события>

Общие правила

1) Существует одна, общая для всех событий, очередь.
2) Включить событие в очередь (оповестить о его наступлении) можно «вручную» (команда `SET EVENT`) или автоматически (при наступлении в БД описанного события).
3) СУБД ЛИНТЕР автоматически включает событие в очередь событий (независимо от того, была ли транзакция завершена оператором `COMMIT` или `ROLLBACK`) в следующих случаях:
   - если событие создано с использованием `<запроса выборки>`, оно ставится в очередь в том случае, когда указанный запрос выдает непустое множество ответов. Выполнение запроса происходит каждый раз после операции реального обновления указанных в нем таблиц;
   - если событие создано с указанием типов операций, оно ставится в очередь после выполнения любой из указанных операций с непустым количеством обработанных строк указанной таблицы;
   - в обоих случаях, если событие уже находится в очереди, то повторно оно туда не ставится. Если оно было в очереди, и `<запрос выборки>` дал пустой ответ, событие удаляется из очереди.
4) Поставить событие в очередь может:
   - его создатель;
   - создатель БД;
   - пользователь с привилегией `RESOURCE` (если событие не `PRIVATE`).

Ожидание события (WAIT EVENT)

Функция

Определение оператора ожидания события.

Спецификация

[1] <ожидание события>::= WAIT EVENT <выражение-событие>

Синтаксические правила

1) `<Выражение-событие>` должно состоять из имен событий, логических условий `AND`, `OR`, `NOT` и группировочных скобок.

```
create event EVN1 as delete, insert on tab1;
create event EVN2 as update on tab2
create event EVN3 as select * from tab3;
```

```
wait event (EVN1 and EVN3) or (EVN2 and EVN3);
```

2) Если модификатор NOT не указан, то происходит ожидание до тех пор, пока заданная в <выражении-событии> комбинация событий даст значение TRUE.
3) Если модификатор NOT указан, то происходит ожидание до тех пор, пока заданная в <выражении-событии> комбинация событий даст значение FALSE.
4) Если в <выражении-событии> задано событие, которое к моменту выполнения WAIT EVENT уничтожено другим каналом, фиксируется исключительная ситуация.

Общие правила

1) Если результат вычисления <выражения-события> TRUE, управление передается в программу, ожидающую комбинацию указанных событий. Сами события при этом из очереди не удаляются.
2) Информация о том, какое именно событие наступило в <выражении-событии>, возвращается в битовой маске (поле ROWID блока управления запросом CBL) (см. документ «Интерфейс нижнего уровня», пункт «Выполнение SQL-запроса») или в переменной ROWIDPCI_ (см. документ «Встроенный SQL», раздел «Псевдопеременные встроенного языка»). Каждый бит маски (начиная с нулевого) соответствует порядковому номеру события в <выражении-событии>. Значение бита 1 – событие наступило.
3) Битовая маска событий может содержать информацию не более чем о 32 событиях из <выражения-события>, для остальных (с порядковыми номерами больше 32) информация о состоянии события через битовую маску недоступна.
4) Максимальное количество допустимых событий для команды WAIT EVENT определяется количеством доступных каналов ядра СУБД ЛИНТЕР, т.к. на каждое событие в ядре СУБД открывается отдельный канал.
5) Если количество событий больше 32 (максимальный размер битовой маски), то в случае наступления запрошенной комбинации событий ядро СУБД все равно вернет управление пользовательскому приложению, но чтобы узнать, какое именно событие произошло, необходимо опрашивать интересуемые события отдельно. Также отдельный опрос событий надо выполнять, если среди них есть события типа AUTORESET.

Опрос события (GET EVENT)

Функция

Определение оператора опроса события.

Спецификация

[1] <опрос события>::= GET EVENT <выражение-событие>

Синтаксические правила

1) <Выражение-событие> должно состоять из имен событий, логических условий AND, OR, NOT и группировочных скобок.
2) Результатом выполнения запроса является код завершения:
   - 0, если результат <выражение-событие> TRUE;
   - 2, если – FALSE (см. документ «Справочник кодов завершения», пункт «Код завершения 2»).

   Событие при выполнении данного запроса из очереди не удаляется.

© Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

Общие правила

1) Информация о том, какое именно событие в <выражении-событии> наступило, возвращается таким же образом, как и при выполнении запроса WAIT EVENT.

Сброс события (CLEAR EVENT)

Функция

Определение оператора сброса события.

Спецификация

[1] <сброс события>::= CLEAR EVENT [<имя схемы>.]<имя события>

Общие правила

1) Сбросить событие может:
   - его создатель;
   - создатель БД;
   - пользователь с привилегией RESOURCE (если событие не PRIVATE).
2) Сбросить событие (перевести его в состояние «неактивное») можно только для ранее активированного по команде SET EVENT события.

Пример

Событие настроено на добавление в таблицу новой записи (команда INSERT).

1) insert (генерируется событие «Добавление записи»);
2) приложение обрабатывает это событие и сбрасывает его (чтобы отследить очередное добавление записи);
3) insert (опять генерируется событие «Добавление записи»);
4) приложение обрабатывает это событие и сбрасывает его

и т.д.

Синонимы

Таблицы и представления в СУБД ЛИНТЕР могут иметь дополнительные имена (синонимы), причем разные для различных пользователей. Синоним – это идентификатор в терминах языка SQL, т.е. он может быть длиной до 66 символов и, если содержит не алфавитно-цифровые символы, должен заключаться в двойные кавычки.

Имя синонима может быть использовано вместо имени соответствующего таблицы/представления в любом контексте. Действия при этом выполняются не над синонимом, а над таблицей/представлением, на которую указывает синоним. Действия собственно над синонимом выполняют только команды CREATE SYNONYM и DROP SYNONYM.

Создание синонима (CREATE SYNONYM)

Функция

Определение оператора создания синонима.

Спецификация

[1] <создание синонима>::=
<создание публичного синонима>|<создание личного синонима>
[2] <создание публичного синонима>::=
CREATE [IF NOT EXISTS | OR REPLACE] PUBLIC SYNONYM
<имя синонима> FOR <спецификация объекта>
[3] <создание личного синонима>::=
CREATE [IF NOT EXISTS | OR REPLACE] SYNONYM
[<имя схемы>.]<имя синонима> FOR <спецификация объекта>
[4] <спецификация объекта>::=
[<имя схемы>.]{<имя таблицы>|<имя представления>|<имя синонима>}

Синтаксические правила

1) Объект, для которого создается синоним, должен существовать в момент создания синонима.
2) Если в <создание личного синонима> не задано <имя схемы>, то используется текущая схема БД.
3) Опция OR REPLACE заставляет удалять существующий в БД синоним и создавать его под тем же именем, но с указанными параметрами.
4) Опция IF NOT EXISTS отменяет выполнение оператора, если указанный синоним уже существует в БД.
5) Одновременное использование опций IF NOT EXISTS и OR REPLACE запрещено.
6) <Имя синонима> не должно совпадать с именем таблиц, представлений и синонимов, в пределах <имя схемы>, если не задано ключевое слово PUBLIC, и с уже существующими общими синонимами, если это слово задано.
7) Если указано PUBLIC, то синоним действует для всех пользователей БД, которые не создавали таблиц, представлений или синонимов с тем же именем. В противном случае этот синоним действует только для своего создателя, но другие пользователи также могут использовать его, задавая конструкцию <имя схемы>.<имя синонима>.
8) Конструкция вида:

```
select ... from <имя схемы>.<имя PUBLIC-синонима> ...
```

является недопустимой.

Общие правила

1) Для создания PUBLIC-синонима требуется уровень прав DBA.

```
create table "Массив случайных чисел" (d double);
create synonym RANDEM for "Массив случайных чисел";
select * from RANDEM,
  tab1 where d between 0.5 and 1.0 and randem.rowid=tab1.rowid;
create table tab1(I int autoinc);
insert into randem (d) values (0.2);
insert into randem (d) values (0.4);
insert into randem (d) values (0.7);
insert into randem (d) values (1.0);
insert into randem (d) values (1.6);
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
select * from RANDEM, tab1
 where d between 0.5 and 1.0 and randem.rowid=tab1.rowid;

| 0.7 | 2 |
| 1.0 | 3 |
```

сравните (в этом варианте локальный синоним RANDEM известен только внутри SQL-запроса и не конфликтует с созданным в БД глобальным синонимом RANDEM):

```
select * from "Массив случайных чисел" as RANDEM, tab1
 where d between 0.5 and 1.0
   and randem.rowid=tab1.rowid;

| 0.7 | 2 |
| 1.0 | 3 |
```

2) Для создания синонима с <имя схемы> требуется уровень прав RESOURCE.

3) Права на выполнение DML-операций с синонимами наследуются от объектов, для которых создан синоним и проверяются на этапе выполнения операции.

4) Длина цепочки синонимов, ссылающихся друг на друга, ограничена числом 10. При большей длине цепочки считается, что таблица не найдена.

5) При выполнении запроса с <именем синонима> без указания <имя схемы> вначале производится поиск синонима среди личных синонимов текущей схемы, при отсутствии – поиск среди общих синонимов.

Удаление синонима (DROP SYNONYM)

Функция

Определение оператора удаления синонима таблицы (представления).

Спецификация

[1] <удаление синонима>::=
DROP [PUBLIC] SYNONYM [<имя схемы>.]<имя синонима> [CASCADE]

Синтаксические правила

1) Если задана опция PUBLIC, то удаляется общий синоним, иначе – синоним в указанной <имя схемы>, иначе синоним в текущей схеме.

```
create public synonym public_auto for system.auto;
select * from public_auto;
drop public synonym public_auto;
```

2) Опция CASCADE заставляет удалять все таблицы и представления, созданные на основе данного синонима.

Общие правила

1) Удаление синонима не влияет на объект (таблицу, представление), которому он был назначен.

2) Для удаления PUBLIC-синонима требуется уровень прав DBA.

3) Для удаления синонима с <имя схемы> требуется уровень прав RESOUCE.

4) При запуске ядра СУБД ЛИНТЕР с ключом /COMPATIBILITY=STANDARD запрещается удаление таблицы и синонима, на которую ссылаются представления (VIEW). При этом на консоль ядра и в файл linter.out выдаётся список объектов, ссылающихся на удаляемый объект и не позволяющих его удалить. Пример списка:

```
INFO Can't delete relation '"SYSTEM"."TEST1" (#187)' because
 exists reference(s):
"SYSTEM"."TESTV" (#188)
"SYSTEM"."TESTV2" (#190)
```

5) Каскадное удаление таблицы на которую ссылаются представления (VIEW) разрешено всегда.

Комментарии

Создание комментария (COMMENT ON)

Функция

Создание комментария к объекту БД.

Спецификация

[1] <создание комментария> ::=
COMMENT ON
{TABLE <полное имя таблицы>
|COLUMN <полное имя таблицы>.<имя столбца>
|PROCEDURE <полное имя хранимой процедуры>
|PARAMETER <полное имя хранимой процедуры>.<имя параметра процедуры>
|USER <имя пользователя>
|ROLE <имя роли>}
IS <комментарий>

[2] <полное имя таблицы> ::= [<имя схемы>.]<имя таблицы>

[3] <полное имя хранимой процедуры> ::=
[<имя схемы>.]<имя хранимой процедуры>

[4] <имя параметра процедуры> ::= <идентификатор>

[5] <комментарий> ::= <символьный литерал>

Синтаксические правила

1) Максимальная длина <комментария> 240 символов. При превышении предельной длины он усекается до 240 символов без какой-либо диагностики.

Общие правила

1) Для создания комментариев в БД должна существовать системная таблица $$$OBJ_COMMENTS (создается с помощью SQL-скрипта systab.sql).

2) Комментарии записываются в кодировке, установленной для системных таблиц БД (SET DATABASE NAMES <имя кодировки>).

3) При удалении объекта комментарий к нему удаляется автоматически.

4) При переименовании объекта комментарий к нему не изменяется.

5) Для добавления комментария на таблицу или ее столбец нужно иметь привилегию REFERENCES на эту таблицу и соответствующий уровень мандатного доступа к ней.

6) Для добавления комментария на процедуру или ее параметр нужно быть владельцем процедуры. Для добавления комментария на пользователя или роль нужно иметь уровень прав DBA.

Примеры

1)

```
Создание и просмотр комментариев к таблицам
comment on table auto is 'Демонстрационная таблица';
-- Комментарии к таблицам
select cast ($$$usr.$$$s34 as char (18)) as "Имя схемы",
       cast ($$$sysrl.$$$s13 as char (18)) as "Имя таблицы",
       cast ($$$obj_comments.$$$text as char (30)) as
 "Комментарий"
  from LINTER_SYSTEM_USER.$$$sysrl,
       LINTER_SYSTEM_USER.$$$usr,
       LINTER_SYSTEM_USER.$$$obj_comments
 where $$$usr.$$$s31   = $$$sysrl.$$$s12
   and $$$usr.$$$s32   = 0
   and $$$obj_comments.$$$obj_type = 8
   and /* код 8 - таблица */
     $$$obj_comments.$$$obj_id = $$$sysrl.$$$s11;

Результат:

 Имя схемы       Имя таблицы     Комментарий
 ---------       -----------     -----------
|SYSTEM         |AUTO           |Демонстрационная таблица |
```

2)

```
Создание и просмотр комментариев к столбцам таблицы

comment on column auto.model is 'Марка автомобиля';
comment on column auto.color is 'Цвет автомобиля';
comment on column auto.bodytype is 'Тип кузова';

-- Комментарии к столбцам таблицы AUTO.SYSTEM
select
  cast ($$$attri.$$$s23 as char (18)) as "Имя столбца",
  cast ($$$obj_comments.$$$text as char (30)) as "Комментарий"
from
  LINTER_SYSTEM_USER.$$$sysrl,
  LINTER_SYSTEM_USER.$$$usr,
  LINTER_SYSTEM_USER.$$$attri,
  LINTER_SYSTEM_USER.$$$obj_comments
where
  $$$attri.$$$s21 = $$$sysrl.$$$s11 and
  $$$usr.$$$s31   = $$$sysrl.$$$s12 and
```

```
  $$$sysrl.$$$s13 = 'AUTO'          and /* или другое имя таблицы
  */
  $$$usr.$$$s34   = 'SYSTEM'        and /* или другое имя схемы
  */
  $$$usr.$$$s32   = 0               and
  $$$obj_comments.$$$obj_type = 22  and /* код 22 - столбец */
  $$$obj_comments.$$$obj_id = $$$sysrl.$$$s11 and
  $$$obj_comments.$$$nmr    = $$$attri.$$$s22;
```

Результат:

```
 Имя столбца        Комментарий
 -----------        -----------
|MODEL              |Марка автомобиля              |
|COLOR              |Цвет автомобиля               |
|BODYTYPE           |Тип кузова                    |
```

3)

```
! Создание и просмотр комментариев к хранимой процедуре и ее
 параметрам

create user u1 identified by '12345678';
grant dba to u1;
username U1/12345678
create or replace procedure proc_c (in i int; in j int) result int
code
  return i+j;
end;
comment on procedure proc_c is 'Сумма двух значений';
comment on parameter proc_c.i is 'Первое слагаемое';
comment on parameter proc_c.j is 'Второе слагаемое';
comment on parameter u1.proc_c.j is 'Второе слагаемое';

! Комментарии к процедурам
select
  cast ($$$usr.$$$s34 as char (18)) as "Имя схемы",
  cast ($$$proc.$$$name as char (18)) as "Имя процедуры",
  cast ($$$obj_comments.$$$text as char (30)) as "Комментарий"
from
  LINTER_SYSTEM_USER.$$$proc,
  LINTER_SYSTEM_USER.$$$usr,
  LINTER_SYSTEM_USER.$$$obj_comments
where
  $$$usr.$$$s31   = $$$proc.$$$owner and
  $$$usr.$$$s32   = 0                and
  $$$obj_comments.$$$obj_type = 12   and /* код 12 - процедура */
```

```
  $$$obj_comments.$$$obj_id = $$$proc.$$$id;

Результат:

 Имя схемы          Имя процедуры      Комментарий
 ---------          -------------      -----------
|U1                |PROC_C             |Сумма двух значений
```

4)

```
! Комментарии к параметрам процедуры U1.PROC_C
select
  cast ($$$prcd.name as char (18)) as "Имя параметра",
  cast ($$$obj_comments.$$$text as char (30)) as "Комментарий"
from
  LINTER_SYSTEM_USER.$$$proc,
  LINTER_SYSTEM_USER.$$$usr,
  LINTER_SYSTEM_USER.$$$prcd,
  LINTER_SYSTEM_USER.$$$obj_comments
where
  $$$usr.$$$s31   = $$$proc.$$$owner and
  $$$usr.$$$s32   = 0                and
  $$$prcd.procid  = $$$proc.$$$id    and
  $$$proc.$$$name = 'PROC_C'         and /* или другое имя
 процедуры */
  $$$usr.$$$s34   = 'U1'             and /* или другое имя схемы
   */
  $$$obj_comments.$$$obj_type = 23   and /* код 23 - параметр */
  $$$obj_comments.$$$obj_id = $$$proc.$$$id and
  $$$obj_comments.$$$nmr = $$$prcd.argid;

Результат:

 Имя параметра      Комментарий
 -------------      -----------
|I                 |Первое слагаемое              |
|J                 |Второе слагаемое              |
```

Изменение комментария

Если комментарий к объекту уже существует, то при повторном создании комментария к этому объекту старый комментарий заменяется новым.

Удаление комментария

Удаление комментария производится путем задания пустого комментария, например, `comment on table test is ' ';`

Кодовые страницы

Для представления символьной информации клиентские приложения могут использовать разные кодовые таблицы. Чтобы избежать накладных расходов при перекодировке данных, извлекаемых из БД, в необходимую клиентскому приложению кодировку (и, наоборот, при записи клиентских данных в БД), СУБД ЛИНТЕР обеспечивает возможность хранить данные в БД в заданной пользователем кодировке (это касается только однобайтовых кодировок, многобайтовые кодировки MBCS и UTF8 будут преобразованы в UNICODE) и получать эти данные без перетрансляции, что существенно ускоряет работу СУБД.

Трансляции служат для прямой перекодировки между двумя однобайтовыми кодировками (задают таблицу перекодировки). В случае отсутствия необходимой трансляции она генерируется автоматически через UNICODE, при этом недостающие символы заменяются символами «?».

Создание кодировки (CREATE CHARACTER SET)

Функция

Определение оператора создания в БД новой кодировки.

Спецификация

[1] <создание кодировки>::=
CREATE [IF NOT EXISTS | OR REPLACE] CHARACTER SET <имя кодировки>
[WINDOWS_CODE <код Windows-кодировки>]
[{NONEUC [WIDTH <значение>]} | EUC | UTF8]
[[PLAN <номер плана>] PAGE <номер страницы>]
EXTERNAL (<тело кодовой таблицы>)
[2] <код Windows-кодировки>::= <беззнаковое целое>
[3] <номер плана>::= <беззнаковое целое>
[4] <номер страницы>::= <беззнаковое целое>
[5] <тело кодовой таблицы>::= <байтовый литерал>

Синтаксические правила

1) Опция OR REPLACE заставляет удалять существующую в БД кодовую таблицу и создавать её под тем же именем, но с другими параметрами.

2) Опция IF NOT EXISTS отменяет выполнение оператора, если указанная кодовая таблица уже существует в БД.

3) Одновременное использование опций IF NOT EXISTS и OR REPLACE запрещено.

4) Опции <номер плана> и <номер страницы> используются только при задании MBCS-кодировок.

5) <Тело кодовой таблицы> – массив из 1536 байт. Для однобайтовой кодировки включает в себя следующие массивы: CTYPE (512 байт), CASE (256 байт), WEIGHTS (256 байт), TO_UNICODE (512 байт):

   - CTYPE – имеет смысл только для однобайтовых кодировок и представляет собой массив свойств символов кодировки (пример свойств символа – число, буква или знак; в верхнем/нижнем регистре и т.п.);
   - CASE – имеет смысл только для однобайтовых кодировок и представляет собой массив преобразований символов в верхний/нижний регистр;

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

- WEIGHTS – имеет смысл только для однобайтовых кодировок и представляет собой массив весов символов кодировки;
- TO_UNICODE – имеет смысл только для однобайтовых кодировок и представляет собой таблицу преобразования символов кодировки в UNICODE-символы (массивы для обратного преобразования из кодировки в UNICODE автоматически создаются на этапе загрузки кодировки).

6) Для MBCS-кодировки можно указывать следующие типы кодировок:
   - EUC (по умолчанию) (примеры подобных кодировок: SHIFTJIS, JIS0208, JIS0201, HANGUL, JOHAB, KSC5601, BIG5. В ОС MS DOS: кодировки: 932, 936, 949, 950);
   - NONEUC – кодировка не совместима с EUC-овскими (примеры подобных кодировок: GB 2312-80, GB 12345-90, GB 7589-87, GB 7590-87, GB 8565-89, JIS0212, KSX1001, CNS11643);
   - UTF8 – UTF-8 представление символов.
7) Опции PLAN и PAGE задают, соответственно, номер плана и номер страницы для CNS11643-кодировки и обобщенной GB-кодировки, состоящей из GB 2312-80, GB 12345-90, GB 7589-87, GB 7590-87, GB 8565-89. Если PLAN не указан, считается, что он равен нулю.
8) Опция WINDOWS_CODE задает номер кодовой страницы для Windows-платформ.

Примечание
После создания БД в обычном режиме по умолчанию задана кодировка, использующая таблицу из 127 символов. Существует возможность изменить кодировку по умолчанию для всей БД (см. пункт «Кодировка соединения по умолчанию»).

Пример

Создание однобайтовой кодировки:

```
CREATE CHARACTER SET CP866 WINDOWS_CODE 866 EXTERNAL(
hex('2000200020002000200020002000200020002800280028002800280020002000200
0020002000200020002000200020002000200020002000200028002800280028000800100010
0001000100010001000100010001000100010001000100010001000040004000400040
0040004000400040004000400010001000100010001000100010001000010101010101010101010
1010101010101010101010101010101010101010101010101010101010101010101010101010101
0101010101011000100010001000100010001000020102010201020102010201020102010
2010201020102010201020102010201020102010201020102010201020102010201020102010201
1000100010001000200001010101010101010101010101010101010101010101010101010101010
1010101010101010101010101010101010101010101010101010101010101010101010101010101
0101010201020102010201020102010201020102010201020102010201020102010201020102011
0001000100010001000100010001000100010001000100010001000100010001000100010001000
1000100010001000100010001000100010001000100010001000100010001000100010001000100
0100010001000100010001000100010001000100010001000100010002010201020102010201020102
0102010201020102010201020102010201020102010201010201010201010201020101010
2011000100010001000100010001000800'),
hex('000102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F20
212223242526272829 2A2B2C2D2E2F303132333435363738393A3B3C3D3E3F406162636
465666768696A6B6C6D6E6F707172737475767778797A5B5C5D5E5F6041424344454647
48494A4B4C4D4E4F505152535455565758595A7B7C7D7E7FA0A1A2A3A4A5A6A7A8A9AAA
```

```
BACADAEAFE0E1E2E3E4E5E6E7E8E9EAEBECEDEEEF80818283848586878889 8A8B8C8D8E
8FB0B1B2B3B4B5B6B7B8B9BABBBCBDBEBFC0C1C2C3C4C5C6C7C8C9CACBCCCDCECFD0D1D
2D3D4D5D6D7D8D9DADBDCDDDEDF909192939495969798999A9B9C9D9E9FF1F0F3F2F5F4
F7F6F8F9FAFBFCFDFEFF'),
hex('000102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F20
212223242526272829 2A2B2C2D2E2F303132333435363738393A3B3C3D3E3F404142434
445464748494A4B4C4D4E4F505152535455565758595A5B5C5D5E5F6061626364656667
68696A6B6C6D6E6F707172737475767778797A7B7C7D7E7F80818283848586878889 8A8
B8C8D8E8F909192939495969798999A9B9C9D9E9FA0A1A2A3A4A5A6A7A8A9AAABACADAE
AFB0B1B2B3B4B5B6B7B8B9BABBBCBDBEBFC0C1C2C3C4C5C6C7C8C9CACBCCCDCECFD0D1D
2D3D4D5D6D7D8D9DADBDCDDDEDFE0E1E2E3E4E5E6E7E8E9EAEBECEDEEEFF0F1F2F3F4F5
F6F7F8F9FAFBFCFDFEFF'),
hex('000001000200030004000500060007000800090 00A000B000C000D000E000F0010
001100120013001400150016001700180019001A001B001C001D001E001F00200021002
200230024002500260027002800290 02A002B002C002D002E002F003000310032003300
340035003600370038003900 3A003B003C003D003E003F004000410042004300440045 0
046004700480049004A004B004C004D004E004F005000510052005300540055005600 57
00580059005A005B005C005D005E005F006000610062006300640065006600670068006
9006A006B006C006D006E006F007000710072007300740075007600770078007900 7A00
7B007C007D007E007F001004110412041304140415041604170418041904 1A041B041C0
41D041E041F042004210422042304240425042604270428042904 2A042B042C042D042E
042F043004310432043304340435043604370438043904 3A043B043C043D043E043F049
125922593250225242561256225562555256325512557255D255C255B25102514253425
2C251C2500253C255E255F255A255425692566256025502 56C256725682564256525592
55825522553256B256A2518250C25882584258C259025802540044104420443044 0445
044604470448044904 4A044B044C044D044E044F040104510404045404070457040E045
E04B0001922B7001A221621A400A025A000')
);
```

Примечание
При копировании примера для выполнения необходимо удалить символы перевода строки.

Кодировка системного словаря БД (SET DATABASE NAMES)

Функция

Определение оператора задания кодировки, используемой СУБД ЛИНТЕР для представления данных в системных таблицах `$$$SYSRL`, `$$$ATTRI`, `$$$USR`.

Спецификация

[1] <кодировка системного словаря>::=
SET DATABASE NAMES <имя кодировки>

Синтаксические правила

1) <Имя кодировки> должно представлять имя любой загруженной в БД однобайтовой кодировки.

© Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

Общие правила

1) Для использования команды в БД должна быть создана и загружена необходимыми данными системная таблица кодировок $$$CHARSET.
2) <Имя кодировки> задает кодировку, которая будет использоваться для кодировки системного словаря БД (для представления данных в системных таблицах ($$$SYSRL, $$$ATTRI, $$$USR и других).
3) Если кодировка системного словаря не задана, используется кодовая страница с именем DEFAULT или 0 (если кодировки DEFAULT нет, т.е. встроенная ASCII7).
4) Запрещено выполнение команды SET DATABASE NAMES, если имеются индексы на символьные столбцы системных таблиц. В таком случае индексы на системные таблицы следует удалить и после выполнения SET DATABASE NAMES пересоздать.
5) При смене данной командой кодировки системного словаря в БД перекодируются имена пользователей и ролей групп. Но, так как пароли пользователей не перекодируются, после смены кодировки системного словаря все пароли, заданные в национальной кодировке (с кодами символов больше 127), становятся неизвестными СУБД (под ними невозможно зарегистрироваться в БД). В связи с этим создателю БД нельзя назначить пароль в кодировке, отличной от системной кодировки по умолчанию (системная кодировка по умолчанию – это первые 127 символов). При попытке назначить создателю БД пароль в другой кодировке выдается код завершения 1065.

Пример

```
set database names CP866;
```

Кодировка соединения по умолчанию (SET NAMES)

Функция

Определение оператора задания кодировки, используемой клиентским приложением по умолчанию для текущего соединения с БД.

Спецификация

[1] <кодировка соединения по умолчанию> **::=**
SET NAMES <имя кодировки>

Общие правила

1) Для использования команды в БД должна быть создана и загружена необходимыми данными системная таблица кодировок $$$CHARSET.
2) <Имя кодировки> задает кодировку, которая будет применяться клиентским приложением по умолчанию в том соединении с БД, по которому подана команда SET NAMES.
3) Если <кодировка соединения по умолчанию> не задана (после установления соединения с БД), выбор кодировки соединения выполняется по следующим правилам:
   - если определена переменная окружения LINTER_CP, то используется определяемая этой переменной кодировка;
   - если переменная LINTER_CP не определена, используется текущая кодировка операционной системы, в которой работает клиентское приложение.

4) <Кодировка соединения по умолчанию> может быть установлена в любой момент выполнения соединения с БД и остается неизменной до отсоединения от БД или до изменения ее другой <кодировкой соединения по умолчанию>.

5) Команда SET NAMES позволяет создавать по умолчанию все таблицы и символьные столбцы в них в указанной кодировке, потому что при создании таблиц (столбцов) используется следующий приоритет выбора кодировки:

   - используется кодировка, заданная для таблицы или столбца непосредственно в SQL-запросе (CREATE TABLE, ALTER TABLE ADD COLUMN и т.п.) – если задана;
   - иначе кодировка, заданная для канала по SET NAMES – если задана;
   - иначе кодировка, заданная для БД по SET DATABASE DEFAULT CHARACTER SET – если задана;
   - иначе кодировка, заданная для канала другим способом (например, через переменную среды окружения LINTER_CP).

Пример

```
set names CP866;
```

Кодировка данных пользовательских таблиц (SET DATABASE DEFAULT CHARACTER SET)

Функция

Определение оператора задания кодировки для представления символьных данных в пользовательских таблицах.

Спецификация

[1] <пользовательская кодировка> **::=**
SET DATABASE DEFAULT CHARACTER SET <имя кодировки>

Синтаксические правила

1) <Имя кодировки> должно представлять имя любой загруженной в БД кодировки (в том числе это может быть и MBCS-кодировка).

Общие правила

1) Для использования команды в БД должна быть создана и загружена необходимыми данными системная таблица кодировок $$$CHARSET.

2) Заданная кодировка будет использована при создании всех пользовательских таблиц.

3) Если <пользовательская кодировка> не указана в запросе создания таблицы, то при создании столбцов символьных типов (CHAR, VARCHAR) пользовательских таблиц применяется следующий алгоритм выбора кодировки (в порядке уменьшения приоритета):

   - используется кодировка, указанная в запросе создания таблицы

```
CREATE TABLE T3 CHARACTER SET CP437(C CHAR(10));
```

   - если кодировка явно задана для столбца, то она и используется для этого столбца

```
CREATE TABLE T3 (C CHAR(10) CHARACTER SET CP437);
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

- если кодировка явно не задана для столбца, но задана для всей таблицы, то она используется для всех столбцов таблицы

```
CREATE TABLE T3 CHARACTER SET CP437(C CHAR(10));
```

- если кодировка для таблицы не задана, а по каналу был выполнен SQL-запрос SET NAMES (например, SET NAMES CP950;), т.е. установлена кодировка канала, то используется она;
- если кодировка канала не задана, а установлена кодировка по умолчанию для пользовательских таблиц БД SQL-запросом SET DATABASE DEFAULT CHARACTER SET, то используется она;
- если кодировка по умолчанию для пользовательских таблиц БД не задана, то используется (если задана) кодировка системного словаря (SQL-запрос SET DATABASE NAMES);
- если кодировка системного словаря не задана, используется нулевая кодовая страница, то есть используется введенное байтовое представление символов. В результате всегда правильно будут отображаться только символы с числовым значением до 127, корректность отображения других введенных символов будет зависеть от программного средства, используемого для их визуализации.

Создание правила трансляции (CREATE TRANSLATION)

Функция

Определение оператора создания правила трансляции из одной кодировки в другую.

Спецификация

[1] <создание правила трансляции> : : =
CREATE [IF NOT EXISTS | OR REPLACE] TRANSLATION <имя трансляции>
FOR <имя исходной кодировки>
TO <имя целевой кодировки>
EXTERNAL (<массив трансляции>)
[2] <имя трансляции> : : = <идентификатор>
[3] <имя исходной кодировки> : : = <идентификатор>
[4] <имя целевой кодировки> : : = <идентификатор>
[5] <массив трансляции> : : = <байтовый литерал>

Синтаксические правила

1) <Имя трансляции> – имя правила трансляции, должно быть уникальным в БД.
2) Опция OR REPLACE заставляет удалять существующую в БД трансляцию и создавать её под тем же именем, но с другими параметрами.
3) Опция IF NOT EXISTS отменяет выполнение оператора, если указанная трансляция уже существует в БД.
4) Одновременное использование опций IF NOT EXISTS и OR REPLACE запрещено.
5) <Имя исходной кодировки> – имя кодировки, из которой будет осуществляться трансляция. Данная кодовая страница должна быть установлена в БД.
6) <Имя целевой кодировки> – имя кодировки, в которую будет осуществляться трансляция. Данная кодовая страница должна быть установлена в БД.
7) <Массив трансляции> – массив соответствия символов между исходной и целевой кодировками (256 байт).

Общие правила

1) Созданное правило трансляции доступно всем пользователям БД.

Пример

```
CREATE TRANSLATION fromCP866toCP1251 FOR CP866 TO CP1251 EXTERNAL(
hex('000102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F2
021222324252627282 92A2B2C2D2E2F303132333435363738393A3B3C3D3E3F4041424
34445464748494A4B4C4D4E4F505152535455565758595A5B5C5D5E5F6061626364656
66768696A6B6C6D6E6F707172737475767778797A7B7C7D7E7FC0C1C2C3C4C5C6C7C8C
9CACBCCCDCECFD0D1D2D3D4D5D6D7D8D9DADBDCDDDEDFE0E1E2E3E4E5E6E7E8E9EAEBE
CEDEEEFA9AE88A681828384858687AD898A8B8C8D8E8F909792939480 9691BD999A9B9
C9D9E9FA0B6B5A3BCA5B3A7B4BBB1ABACF0F1F2F3F4F5F6F7F8F9FAFBFCFDFEFFA8B8A
ABAAFBFA2A1B095B7B2B9A498BE'));
```

Примечание
При копировании примера для выполнения необходимо удалить символы перевода строки.

Удаление кодировки (DROP CHARACTER SET)

Функция

Определение оператора отмены возможности использовать в БД указанную кодировку.

Спецификация

[1] <удаление кодировки>::= DROP CHARACTER SET <имя кодировки>

Синтаксические правила

1) <Имя кодировки> должно ссылаться на ранее установленную в БД кодировку.

Общие правила

1) Для удаления кодировки необходимо иметь уровень прав доступа DBA.
2) Ограничения на удаление кодировки:
   - кодировка не должна быть пользовательской кодировкой по умолчанию (в текущем соединении) и кодировкой системного словаря БД;
   - нет открытых (активных) соединений с БД, использующих данную кодировку;
   - в БД нет объектов (таблиц, отдельных столбцов) использующих данную кодировку.

Пример

```
drop character set cp866;
```

Удаление правила трансляции (DROP TRANSLATION)

Функция

Определение оператора отмены возможности использовать ранее созданное правило трансляции.

© Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

Спецификация

[1] <удаление правила трансляции>::=
DROP TRANSLATION <имя трансляции>

Общие правила

1) Для удаления трансляции необходимо иметь уровень прав доступа DBA.
2) Удаляемое правило трансляции должно существовать в БД.
3) Если после удаления правила трансляции возникает необходимость в его использовании, то СУБД ЛИНТЕР выполняет трансляцию символов через универсальную UNICODE-кодировку (создавая в оперативной памяти временную трансляцию). Временная трансляция остается в ядре СУБД до окончания работы СУБД либо до замены ее эквивалентной трансляцией с помощью конструкции `CREATE TRANSLATION`.

Пример

```
drop translation fromcp1251tocp866;
```

Создание алиаса кодировки (CREATE ALIAS)

Функция

Определение оператора создания алиаса кодировки.

Спецификация

[1] <создание алиаса кодировки>::=
CREATE [OR REPLACE] ALIAS <имя алиаса>
FOR [CHARACTER SET] <имя кодировки>

Синтаксические правила

1) `<Имя алиаса>` не должно совпадать с именами других алиасов или кодировок БД.
2) В качестве `<имени кодировки>` может быть задан алиас, тогда создается дополнительный алиас для той же кодировки, на которую указывает первый.

Общие правила

1) Для создания алиаса необходимо иметь уровень прав DBA.

Пример

```
create alias "Кириллица" for "KOI8-R";
create or replace table tab1 character set "Кириллица" (c char
 (10));
create alias DOS for character set  cp866;
create alias WIN32 for cp866;
create alias WIN for WIN32;
create or replace table tab1
 (c1 char (10) character set "Кириллица",
  c2 varchar(100) character set WIN);
```

Удаление алиаса кодировки (DROP ALIAS)

Функция

Определение оператора удаления алиаса кодировки.

Спецификация

[1] <удаление алиаса кодировки>::= DROP ALIAS <имя алиаса>

Синтаксические правила

1) <Имя алиаса> должно ссылаться на существующий в БД алиас.

Общие правила

1) Для удаления алиаса необходимо иметь уровень прав DBA.
2) Алиасы можно удалять, даже если они были использованы при создании (модификации) объектов БД.

Пример

```
drop alias "Кириллица";
drop alias WIN;
```

Создание/изменение описания кодировки (ALTER CHARACTER SET)

Функция

Определение оператора создания/изменения описания кодировки.

Спецификация

[1] <создание/изменение описания кодировки>::=
ALTER CHARACTER SET <имя кодировки>
SET DESCRIPTION <имя описания>
[2] <имя описания>::= <идентификатор>

Синтаксические правила

1) <Имя кодировки> должно ссылаться на существующую в БД кодировку.

Общие правила

1) Для создания/изменения описания кодировки необходимо иметь уровень прав DBA.
2) Если для указанной кодировки описания не существует, то оно создается; в противном случае старое описание заменяется новым.

Пример

```
alter character set "cp932"
set description "japanese(cp932,shift_jis,x-sjis,x-shift-jis,x-ms-
cp932)";
```


© Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
alter character set cp936 set description chinese_simplified;
```

Удаление описания кодировки (ALTER CHARACTER SET DROP)

Функция

Определение оператора удаления описания кодировки.

Спецификация

[1] <удаление описания кодировки>::=
ALTER CHARACTER SET <имя кодировки> DROP DESCRIPTION

Синтаксические правила

1) <Имя кодировки> должно ссылаться на существующую в БД кодировку.

Общие правила

1) Для удаления описания кодировки необходимо иметь уровень прав DBA.

Пример

```
alter character set cp936  drop description;
```

Манипулирование данными

При запуске ядра СУБД с ключом /COMPATIBILITY=STANDARD, если при выполнении DML-операций (INSERT, DELETE, UPDATE, MERGE) с таблицей в процессе выполнения не обрабатывают ни одной записи, то будет возвращен код завершения 2 ("нет данных"), как и при пустом результате SELECT-запроса.

Добавление записи (INSERT)

Функция

Определение запроса добавления записи в таблицу.

Спецификация

[1] <запрос добавления> : : =
INSERT INTO <объект>[<список столбцов>]
<добавляемые данные>[<модификаторы запроса>]
[2] <объект> : : = [<имя схемы>.]<имя объекта> [<метка доступа>]
[3] <список столбцов> : : = (<имя столбца>[<метка доступа>][, …])
[4] <метка доступа> : : = #[<группа>]#[<RAL>]#[<WAL>]
[5] <добавляемые данные> : : =
{DEFAULT VALUES
| VALUES (<конструктор данных>[, …])
| <запрос выборки>}
[6] <модификаторы запроса> : : =
[WITH LOCK] [WAIT | NOWAIT]
[{QUANT | QUANTUM} TIMEOUT <время>] [WITH PRIORITY <приоритет>]
[7] <конструктор данных> : : = (<значение>[, …])
[8] <запрос выборки> : : =
[()<подзапрос>()] [<ORDER BY-спецификация>]
[9] <значение> : : =
<значимое выражение>
| NULL
| DEFAULT
| <литерал>
| <имя последовательности>.{currval|nextval}
| GUID
| USER
| SYSDATE
| LASTROWID
| LAST_AUTOINC
| SQL-параметр
| <хранимая процедура>
| EXTFILE (<имя файла>[, <имя фильтра>])
[10] <хранимая процедура> : : = <идентификатор>
[11] <имя файла> : : = NULL | ? | <спецификация файла>
[12] <спецификация файла> : : = <символьный литерал>
[13] <группа> : : = <идентификатор> | целое положительное число

Синтаксические правила

1) Опция <метка доступа> поддерживается только в СУБД ЛИНТЕР БАСТИОН.

2) <Имя объекта> должно задавать пользовательскую базовую или таблицу «в памяти» или обновляемое пользовательское представление.

3) Допустимые привилегии для <объекта>, заданного <именем объекта>, должны включать INSERT.

4) <Имена столбцов> должны указывать на столбцы <объекта>, заданного <именем объекта>, один и тот же столбец не должен быть указан более одного раза.

5) <Имя столбца> должно задавать реальный столбец <объекта>. Указание псевдостолбцов (типа ROWID или ROWTIME) не допускается.

6) Отсутствие <имен столбцов> подразумевает неявную спецификацию списка, идентифицирующего все столбцы в <имени объекта> согласно их позициям, указанным при создании таблицы.

```
create table tab1 (i int, vc varchar(10) default 'unknown', b
 byte(5));
insert into tab1 values (1,'System', hex('00fc6733da'));
insert into tab1 values (12,default, hex('01cc34abee'));
insert into tab1 values (34,default, null);
select * from tab1;

| 1  | System  | 00fc6733da |
| 12 | unknown | 01CC34ABEE |
| 34 | unknown | NULL       |
```

7) Количество значений в <конструкторе данных> должно быть равно числу столбцов в явном или подразумеваемом <списке столбцов>. При этом считается, что i-е значение относится к i-му столбцу.

```
create table tab1 (i int autoinc, vc varchar(50) not null, n
 numeric default 0, d date, bol boolean);
insert into tab1(vc, d, bol) values ('АКБ "Промстройбанк"',
 to_date('23.04.2000','dd.mm.yyyy'), false);
```

8) Если в заданном <списке столбцов> не указаны все столбцы <объекта>, то для не упомянутых столбцов заносятся их значения по умолчанию.

9) Опция VALUES позволяет использовать один или несколько списков вставляемых значений данных.

10) Максимальное количество <конструкторов данных> в опции VALUES (т.е. добавляемых одновременно записей) в одном <запросе добавления> ограничено допустимой длиной текста SQL-запроса (не более 32 Кбайта).

```
create or replace table tst (inv_num int, name char(10));

insert into tst (inv_num, name) values (67,'Компьютер'),
 (88,'Принтер'), (678,'Клавиатура');

select * from tst;

 INV_NUM     NAME
 -------     ----
|         67|Компьютер |
|         88|Принтер   |
|        678|Клавиатура|
```

11) В опции VALUES для значения элемента <конструктора данных> разрешены любые выражения, не содержащие обращений к столбцам <объекта>, например: оператор SELECT (в том числе в виде <table-запроса> и <value-запроса>), вызов пользовательской функции, не содержащей SQL-запросов и т.д.

```
CREATE OR REPLACE TABLE Items (
    item_no INT PRIMARY KEY,
    maker CHAR(10),
    type CHAR(10) DEFAULT 'PC',
    value INT
  );
CREATE OR REPLACE TABLE Printer ( code INT PRIMARY KEY,  model
 INT);
INSERT INTO PRINTER VALUES (1, 100), (77, 7700);
INSERT INTO Items VALUES
  (1, 'A', 'Laptop', 12),
  (2, 'B', DEFAULT, NULL),
  (3, 'C', 'Printer', (SELECT CAST(model AS INT) FROM Printer
 WHERE code=1)),
  (4, 'C', 'Printer', (SELECT CAST(model AS INT) FROM Printer
 WHERE code=77));
```

12) Если указан <подзапрос>, то количество столбцов таблицы, порождаемой этим <подзапросом>, должно быть равно числу столбцов в явном или подразумеваемом <списке столбцов>. При этом i-й столбец <подзапроса> относится к i-ому столбцу <объекта>.

```
create table tab1 (i int, "Модель" varchar(20));
insert into tab1 select personid, model from auto;
create table tab1 (i int, "Модель" varchar(20), d date);
insert into tab1(I, "Модель") select personid, model from auto;
```

или

```
insert into tab1(I, "Модель") (select personid, model from auto);
```

13) Если значение i-го элемента опции VALUES не является NULL-значением, то оно должно позволять автоматически преобразовывать его к типу i-го элемента списка столбцов. Автоматическое преобразование выполняется для следующих пар типов:
    - SMALLINT, INT, BIGINT, DECIMAL – все между собой;
    - REAL, DOUBLE – между собой;
    - SMALLINT, INT, BIGINT, DECIMAL преобразуются в REAL, DOUBLE;
    - все строковые типы между собой, но только значение меньшей или равной длины должно вставляться в столбец большей или равной длины;
    - строка, содержащая дату и/или время, преобразуется в DATE.
14) Для присвоения NULL-значений столбцу типа EXTFILE допускается использование конструкции EXTFILE (NULL).

```
create table tab1 (i int, ext1 extfile root 'c:\linter\ext', ext2
 extfile);
insert into tab1(i, ext1, ext2) values (1, null, extfile(null));
```

15) Добавляемое <значение> может быть логическим значением.

```
create or replace table tst (i int, b boolean);
insert into tst values (1,TRUE);
insert into tst values (2, CAST (NULL as BOOLEAN));
insert into tst values (3, 2<1);
select * from tst;

 I          B
 -          -
|        1 | T |
|        2 |   |
|        3 | F |
```

16) Если задана опция DEFAULT VALUES, столбцам присваивается значение по умолчанию. Если для столбца не задано значение по умолчанию, и он может содержать NULL-значение, то вставляется NULL-значение; в противном случае запись не добавляется и выдается код завершения 901 («Не задано значение первичного ключа или NOT NULL столбца»). Транслятор SQL СУБД ЛИНТЕР не проверяет, действительно ли у столбца есть значение по умолчанию, а возможные ошибки выдаются ядром СУБД ЛИНТЕР.

```
create table tab1 (i1 int, i2 int default 0, i3 int default 1000,
  i4 int default -1, vc varchar(20));
insert into tab1(i1, i2, i3, i4, vc)
  values (1, default, default, default, 'SYSTEM');
select * from tab1;

|1 |0 |1000 |-1 |SYSTEM |
```

17) <SQL-параметр> может быть именованный «:имя параметра» или не именованный «?» (см. пункт «SQL-параметр»).

18) Числовое значение <значимого выражения> допускается представлять в виде строкового значения без использования оператора преобразования типа данных.

```
create or replace table tst (i1 int, i2 int, i3 int,
  db1 double, db2 double);
insert into tst values (15, '15', cast '15' as int,
  103.545, '1.03545e+2');

select * from tst;

|       15|       15|       15|     103.545|      103.545|
```

Общие правила

1) Добавление записей выполняется в следующей последовательности:
   - создается строка-кандидат, структура данных которой совпадает со структурой строки базовой таблицы. Если <имя таблицы> идентифицирует представление, то в качестве базовой таблицы выступает та, из которой создано данное представление;
   - все поля строки-кандидата заполняются значениями по умолчанию;

- если задана спецификация DEFAULT VALUES, строка-кандидат вставляется в базовую таблицу;
- для каждого столбца, указанного в <списке столбцов>, значение в строке-кандидате заменяется вставляемым значением;
- строка-кандидат добавляется в базовую таблицу.

2) Если <конструктор данных> содержит добавление нескольких записей и одна из этих записей не может быть добавлена (например, переполнение значения, ограничение целостности CHECK и т.п.), то процесс добавления прерывается на ошибочной записи с выдачей соответствующего кода завершения, а ранее добавленные записи удаляются.

3) При добавлении в BLOB-столбец текстовых данных (типа CHAR, VARCHAR, NCHAR, NCHAR VARYING) выполняется автоматическая перекодировка этих данных в кодировку BLOB-столбца, а бинарные данные добавляются «как есть»:

```
create or replace table
  test(i int, bl blob character set "UCS2");
insert into test(i,bl) values (1, 'АБВГДЕЁЖ');
insert into test(i,bl) values (2, n'АБВГДЕЁЖ');
insert into test(i,bl)
  values (3, HEX('4100420043004400450046 00'));

select i, lenblob(bl), getblobstr(bl, 1, 20) from test;

|          1|         16|АБВГДЕЁЖ..|
|          2|         16|АБВГДЕЁЖ..|
|          3|         12|ABCDEF....|
```

4) Если значение для AUTOROWID-столбца не задано при занесении записей в таблицу, содержащую AUTOROWID-столбец, то для каждой записи в него заносится значение, равное ROWID этой записи (здесь возможен как INSERT одной записи, так и INSERT FROM SELECT).

5) Если значение для AUTOROWID-столбца задано, то запись должна быть занесена в таблицу с ROWID, равным указанному значению. Если означенный ROWID занят другой записью, либо указано недопустимое значение ROWID, то операция завершается с ошибкой. Здесь также возможен как INSERT одной записи, так и INSERT FROM SELECT.

6) Если <имя объекта> ссылается на необновляемый объект, будет зафиксирована исключительная ситуация 2162 («Данное представление не может быть обновлено»).

7) При указании спецификации DEFAULT VALUES автоматически выполняются заданные свойства столбца (например, при AUTOINC значение столбца увеличивается на заданную величину).

```
create table tab1 ( i int autoinc, c char (10) default '???',
  d date default sysdate, n dec default 0);
insert into tab1 default values;
insert into tab1 default values;

select * from tab1;
```

```
|1 |??? |28.04.2003:13:10:15.00 |0.0 |
|2 |??? |28.04.2003:13:10:15.00 |0.0 |
```

8) При занесении значения в столбец типа DECIMAL проверяется, удовлетворяет ли оно значениям PRECISION и SCALE, заданным для столбца. Если задано слишком много цифр дробной части, то число округляется; если задано слишком много цифр целой части, выдается код завершения OVRDECIMAL.

```
create table tab1 ( d1 dec(4,2), d2 dec(15,7));
insert into tab1 values (15.347, 45.12345678);
select * from tab1;

|15.35 |45.1234568 |
```

9) При присвоении по INSERT значения CAST AS <строковый тип> некоторому столбцу без явного указания длины строкового типа длина делается равной длине столбца.

```
create table tab1 (c varchar(2700));
insert into tab1 values (cast '12345' as char);
```

10) Если столбец имеет модификатор AUTOINC, добавляемое значение должно быть больше всех ранее добавлявшихся в таблицу значений данного столбца.

```
create table tab1 (i int autoinc);
insert into tab1 default values;
insert into tab1 values (100);
select * from tab1;

|1   |
|100|
```

11) Если столбец имеет модификатор AUTOINC RANGE, то можно вставлять произвольные значения вне диапазонов.

```
create table tab1 (i int autoinc range (1:100, 500:1000));
insert into tab1 default values;
insert into tab1 values (400);
select * from tab1;

|1   |
|400|
```

12) Возвращаемое <хранимой процедурой> значение должно быть скалярным (не курсорным) и иметь тип данных соответствующего столбца или приводимый к нему.

```
Хранимая процедура:
create or replace procedure insert_tst(in prm int) result char(5)
code
  return to_char(prm*100);//
end;

Вставка значения, возвращаемого хранимой процедурой:
```

```
create or replace table tst (id int autoinc, i int, ch char(10));
insert into tst (i, ch) values(100, '$'+insert_tst(5));
insert into tst (i, ch) values(200, '$'+insert_tst(? (int)));
7

select * from tst;

|          1|        100|$500      |
|          2|        200|$700      |
```

13) Конструкция <ORDER BY-спецификация> задаёт порядок, в котором записи должны добавляться в таблицу.

Примечание

В действительности, реальные значения ROWID добавленных в таблицу записей во многих случаях будут не соответствовать порядку добавления записей. Соответствие будет в следующих случаях:

- если записи заносятся в пустую таблицу;
- после очистки таблицы;
- после сжатия таблицы (команда PRESS);
- в таблице не было удаления записей.

```
CREATE OR REPLACE TABLE T1( i int);
CREATE OR REPLACE TABLE T2( i int);
insert into t2 (i) values(1);
insert into t2 (i) values(2);
insert into t2 (i) values(3);
insert into t2 (i) values(4);
insert into t2 (i) values(5);

insert into t1 (i) select * from t2 order by 1 desc;
select rowid, i from t1;

 ROWID       I
 -----       -
|          1|          5|
|          2|          4|
|          3|          3|
|          4|          2|
|          5|          1|
```

14) Параметр <время> задает максимально допустимую продолжительность выполнения запроса (от 1 до 65535 сек.). Если запрос в отведенное для него время не был выполнен, его обработка прекращается с выдачей соответствующего кода завершения.

15) Конструкция WITH PRIORITY <приоритет> устанавливает заданный приоритет (значение в диапазоне от 0 до 255) выполняемому запросу. Если задать приоритет

больше текущего приоритета пользователя, от имени которого подается запрос, то выдается код завершения 1022 («Нарушение привилегий»).

16) Конструкция WITH PRIORITY <приоритет> устанавливает приоритет только тому запросу, в котором она указана. На приоритет любых последующих запросов она не влияет.

17) При вставке строки в таблицу, содержащую генерируемый столбец, вычисляется ассоциированное с ним <логическое выражение>, и полученное значение становится значением этого столбца в добавляемой записи.

```
create or replace table emp (
  emp_no integer,
  emp_sal double,
  emp_bonus double,
  emp_total generated always as (emp_sal + emp_bonus));
```

При выполнении оператора добавления записи

```
insert into emp (emp_no, emp_sal, emp_bonus) values (1, 40000,
 4000);
```

путем вычисления выражения emp_sal + emp_bonus будет автоматически сгенерировано значение столбца emp_total, и в таблицу emp будет добавлена запись со значениями (1, 40000, 4000, 44000).

18) В добавляемой записи вместо явного значения генерируемого столбца можно использовать опцию DEFAULT.

Приведенные ниже insert-команды эквивалентны.

```
create or replace table tst ("Сумма" numeric, "Скидка" numeric,
  "Итого" generated always as (("Сумма"*(100-"Скидка")/100)));
insert into tst ("Сумма", "Скидка") values (1000,5);
insert into tst ("Сумма", "Скидка") values (2000,7);
select * from tst;

|    1000.0|   5.0|   950.0|
|    2000.0|   7.0|  1860.0|
```

```
insert into tst ("Сумма", "Скидка", "Итого")
  values (1000, 5, default);
insert into tst ("Сумма", "Скидка", "Итого")
  values (2000, 7, default);
select * from tst;

|    1000.0|   5.0|   950.0|
|    2000.0|   7.0|  1860.0|
```

19) Добавление новых записей в циклическую таблицу из этой же таблицы через подзапрос (INSERT INTO tab FROM SELECT tab) имеет некоторые особенности. Это связано с тем, что при попытке добавить первую запись в переполненную таблицу сначала происходит её автоматическое удаление. А когда ядро СУБД обращается за данными для добавления, выясняется, что исходной записи нет и данные брать уже неоткуда. Поэтому место первой удаленной записи в таблице занимает вторая запись из подзапроса, т.е. добавление записей

из подзапроса происходит со сдвигом на одну запись, при этом первая запись подзапроса будет потеряна.

20) Для таблиц «в памяти» <запрос добавления> в режиме OPTIMISTIC не поддерживается (так же как и другие DML-запросы).

21) Модификатор WITH LOCK заставляет блокировать добавляемую запись. Разблокирование добавленной записи выполняется после завершения транзакции (COMMIT/ROLLBACK).

```
create or replace table tst (i int default 0, c char(20) default
 'xxx');
insert into tst(i,c) values(100,'abc') with lock;
insert into tst default values with lock;
insert into tst (i,c) values (100,'abc'), (200, 'def') with lock;
insert into tst (i,c) select personid, make from auto where
 rowid=100 with lock;
```

22) Накладываемая модификатором WITH LOCK блокировка может подвергаться эскалации, т.е. если суммарное количество блокированных (по командам UPDATE и INSERT) в таблице записей превысит 1000, то блокировка записей сбрасывается и блокируется вся таблица (это означает, что с этого момента все вновь добавляемые записи считаются блокированными).

Примечание
Добавляемая запись без модификатора WITH LOCK не блокируется и становится сразу же видимой параллельно работающим транзакциям.

23) Если добавляемая запись с модификатором WITH LOCK ссылается на заблокированную запись, то выдается код завершения 135 («Строка таблицы заблокирована другим пользователем»).

```
create table tpk(i int primary key);
create table tfk(i int references tpk);
exclusive
-- установили транзакционный режим
insert into tpk values (1) with lock;
-- здесь транзакции, которые пытаются читать таблицу tpk или
 только ее запись с i=1, будут ждать снятия блокировки
-- транзакции, которые пытаются внести запись в таблицу tfk с i=1,
 ссылающиеся на добавленную запись, получат код завершения 135
commit;
-- блокировка снимается
```

Примеры

1) Добавление данных в циклическую таблицу

```
create or replace table ct(i int);
alter table ct set records limit 5;
insert into ct values(1),(2),(3), (4),(5);
select * from ct;
```

Этот select-запрос возвращает следующие 5 записей:

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
 I
 -
|          1|
|          2|
|          3|
|          4|
|          5|
```

2)

```
insert into ct(i) select i+10 from ct;
select * from ct;
```

Этот select-запрос возвращает 5 записей, из которых видно, что реально изменились только 4 записи:

```
 I
 -
|         12|
|         13|
|         14|
|         15|
|          5|
```

3) Добавление данных с метками мандатного доступа

```
username SYSTEM/MANAGER8
create level "НЕСЕКРЕТНО"=1;
create level "ДСП"=2;

grant DBA to B identified by '12345678';

alter user B LEVEL("НЕСЕКРЕТНО","НЕСЕКРЕТНО");
username B/12345678

! Создание объектов БД (таблиц) и передача прав на них всем
! пользователям (разрешение всех действий по дискреционному
 доступу):

create or replace table TB(I INT LEVEL("НЕСЕКРЕТНО",
 "НЕСЕКРЕТНО"), C CHAR(20) LEVEL("НЕСЕКРЕТНО", "НЕСЕКРЕТНО"))
 LEVEL("НЕСЕКРЕТНО", "НЕСЕКРЕТНО");

grant all on TB to PUBLIC;

! Добавление данных:

insert into tb values(1,'one');
insert into tb##"НЕСЕКРЕТНО"#"ДСП" values(2,'two');
insert into tb##"НЕСЕКРЕТНО"#"НЕСЕКРЕТНО" values(3,'three');
```

Корректировка записи (UPDATE)

Функция

Определение запроса корректировки записи таблицы.

Спецификация

[1] <запрос корректировки>::=
UPDATE [<имя схемы>.]<имя объекта>
[[AS] <псевдоним объекта>]
[JOIN <соединяемый объект>]
[[AS] <псевдоним соединяемого объекта>[, …]]
SET <значение корректировки>[, …]
[<WHERE-спецификация>]
[WAIT | NOWAIT]
[{QUANT | QUANTUM} TIMEOUT <время>]
[WITH PRIORITY <приоритет>]

[2] <имя объекта>::=
<имя таблицы>[<метка доступа>]|<имя представления>

[3] <псевдоним объекта>::= <идентификатор>

[4] <соединяемый объект>::= <имя таблицы>|<имя представления>

[5] <псевдоним соединяемого объекта>::= <идентификатор>

[6] <значение корректировки>::=
<имя столбца> [<метка доступа>] =
{ <значимое выражение>
| <подзапрос>
| DEFAULT
| NULL
| EXTFILE(NULL | ? | <спецификация файла>[, <имя фильтра>])}

Синтаксические правила

1) <Имя таблицы> должно задавать пользовательскую базовую таблицу или пользовательское обновляемое представление.
2) На таблицу с <именем таблицы> после фразы UPDATE должны быть привилегии UPDATE, для остальных таблиц (из списка JOIN) – привилегии SELECT.

```
create table tab1 (type char(10), id int);
insert into tab1 (type,id) values('System', 1);
UPDATE tab1 SET type=NULL,id=id+3 WHERE id=1;
select * from tab1;

|NULL |4 |
```

3) Имя столбца в <значении корректировки> должно принадлежать только <имени таблицы> из фразы UPDATE (не JOIN).

```
UPDATE Auto JOIN Person SET auto.make='Ford'
 WHERE auto.personid = person.personid and auto.year>70;
```

4) Тип <значимого выражения> должен позволять автоматическое преобразование к типу столбца.

```
create table tab1 (num decimal);
update tab1 set num=length(user) where rowid=12;
update tab1 join tab2
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
   set tab1."Сумма"=tab2."Должн. коэфф."*tab1."Оклад"
 where tab1."Ид. Должности"=tab2."Ид. Должности";

Добавление строковых порций данных в конец BLOB-столбца BLB
 таблицы T_BLOB:
update T_BLOB set BLB=BLB+'порция 1';
update T_BLOB set BLB=BLB+'порция 2';
…
```

5) Числовое значение <значимого выражения> допускается представлять в виде строкового значения без использования оператора преобразования типа данных.

```
create or replace table tst
  (i1 int, i2 int, i3 int, db1 double, db2 double);
insert into tst values (15, '15', cast '15' as int,
  103.545, '1.03545e+2');
update tst set i1=20, i2='20', i3=cast '20' as int;

select * from tst;

|      15|      15|      15|      103.545|     103.545|
|      20|      20|      20|      103.545|     103.545|
```

6) В качестве <значимого выражения> можно использовать <логическое выражение>.

```
create or replace table tst (i int, b boolean);
insert into tst values (1,TRUE);
insert into tst values (2,FALSE);
update tst set b=100-2 > 97 where i=2;
select * from tst;

 I           B
 -           -
|          1|T|
|          2|T|
```

7) В качестве <значимого выражения> можно использовать неименованный (?) или именованный (:имя) параметр. В этом случае тип параметра предполагается совпадающим с типом столбца, которому присваивается <значимое выражение>, если только тип параметра не указан явно.

8) <Подзапрос> должен возвращать единственное значение.

```
create table tab1 (i int autoinc, c char(40));
insert into tab1 default values;
insert into tab1 default values;
insert into tab1 default values;
update tab1 join auto
   set c= (select 'Модель ' || model || ' (' ||
     to_char(year+1900, '9999') || ')'
  from auto where tab1.i=auto. personid);
```

```
select * from tab1;

| 1 | Модель MERCURY COMET GT V8 (1971) |
| 2 | Модель A-310 (1970)               |
| 3 | Модель MATADOR STATION (1971)     |

update tab1 join auto set c =
 (select to_char(sysdate, 'dd.mm.yyyy'))
 where tab1.i=auto. personid;
```

9) Если <подзапрос> вернул пустую выборку, изменяемому значению присваивается NULL-значение.

```
select rowid, i from tst;

|          1|        100|

update tst set i =
 (select rowid from auto where make = 'АвтоВАЗ')
 where rowid=1;

select rowid, i from tst;

|          1|        |
```

10) Опция DEFAULT присваивает столбцу установленное для него значение по умолчанию.

11) Если задана конструкция JOIN, то она должна содержать список реальных или порожденных таблиц, соединяемых с таблицей, в которой выполняется корректировка записей. Условие соединения всех таблиц задается в <WHERE-спецификации>.

```
update auto JOIN person d
   set auto.year = 2014
 where auto.personid = person.personid
   and auto.year = 70;

update tab1 as a join (select user_id as id, sum(ptz) as s
  from tab2
  where PR_ID <> 0 and PR_ID <> -1
  group by user_id) as b
  set a.ptz_m = s
  where a.user_id = b.id;
```



Примечание

Синтаксис оператора UPDATE с соединяемыми таблицами не соответствует стандарту SQL, т.к. в этом стандарте не прописан механизм корректировки записей при наличии соединяемых таблиц. Согласно стандарту SQL можно изменять напрямую или через обновляемое представление записи только в одной таблице или выборке из неё без соединения с другими таблицами.

12) Для корректировки записей в <имени объекта> рекомендуется, чтобы каждой записи из <имени объекта> соответствовала только одна запись первой и последующих соединяемых таблиц. В противном случае результат выполнения запроса не предсказуем.
13) Корректировка записей с использованием соединения таблиц происходит следующим образом:
    - выполняется соединение таблиц;
    - обновляются записи результирующего набора, которые удовлетворяют <WHERE-спецификации>.

Тестовые данные:

Пусть имеются таблицы сотрудников (persons), отделов (departments), этажей (floors).
Таблица persons связана с departments по номеру отдела (d_id), departments связана с floors по номеру этажа (num_f).

```
create or replace table floors (num_f int, f_name char(20));
insert into floors values (1, 'First');
insert into floors values (2, 'Second');
insert into floors values (3, 'Third');
insert into floors values (4, 'Fourth');
insert into floors values (5, 'Fifth');
insert into floors values (6, 'Sixth');

create or replace table departments(d_id int, d_name char(20),
 num_f int);
insert into departments values (1, 'Sales',           1);
insert into departments values (2, 'IT-technologies', 3);
insert into departments values (3, 'Finance',         4);
insert into departments values (4, 'Management',      4);
insert into departments values (5, 'Design',          3);

create or replace table persons(p_id int, p_name char (20), d_id
 int);
insert into persons values (1, 'John',  3);
insert into persons values (2, 'Mary',  2);
insert into persons values (3, 'Kate',  4);
insert into persons values (4, 'Jack',  2);
insert into persons values (5, 'Peter', 7);
insert into persons values (6, 'Ann',   5);
```

Обновить список сотрудников, заменив номер отдела на 5-ый у тех сотрудников, которые работают на 4 этаже (в запросе обновится только левая таблица, т.е. persons):

```
UPDATE persons p JOIN departments d
   SET p.d_id = 5
```

```
 WHERE (p.d_id = d.d_id)
   AND (d.num_f = 4);

SELECT * FROM persons;

 P_ID        P_NAME               D_ID
 ----        ------               ----
|          1|John                |          5|
|          2|Mary                |          2|
|          3|Kate                |          5|
|          4|Jack                |          2|
|          5|Peter               |          7|
|          6|Ann                 |          5|

В результирующем наборе у сотрудников 'John' и 'Kate' номера
 отделов поменялись на '5', т.к. эти сотрудники работали
 соответственно в отделах 'Finance' и 'Management', которые
 находятся на 4 этаже.
```

Общие правила

1) Областью видимости таблицы <имя таблицы> является весь <запрос корректировки>. При этом:
   - при отсутствии <спецификатора выборки> будут изменены все записи таблицы;
   - если задана <WHERE-спецификация>, то изменению подвергается каждая запись таблицы с <именем таблицы>, для которой результат <WHERE-спецификации> истинный. Каждый <подзапрос> в <WHERE-спецификации> реально выполняется для каждой записи в таблице, а результаты <подзапроса> используются в <WHERE-спецификации> применительно к данной записи.
2) Корректировка записей выполняется следующим образом:
   - создается строка-кандидат, структура данных которой совпадает со структурой строки базовой таблицы. Если <имя таблицы> идентифицирует представление, то в качестве базовой таблицы выступает та, из которой создано данное представление;
   - для каждого <значения корректировки> значение заданного столбца в строке-кандидате заменяется заданным корректируемым значением;
   - корректируемая строка заменяется строкой-кандидатом.
3) <Значение корректировки> не должно включать столбцы типа AUTOROWID, AUTOINC и псевдостолбцы ROWID, ROWTIME, DBROWTIME, BLOB-столбцы.
4) При присвоении значения CAST AS <строковый тип> некоторому столбцу без явного указания длины строкового типа длина делается равной длине столбца.
5) При обновлении записей в таблице, содержащей столбец с модификатором AUTOROWID, обновление значений в этом столбце запрещено.
6) Команда UPDATE CURRENT OF ... выполняется по тому каналу, по которому она была подана (а не по каналу, по которому был подан соответствующий SELECT-запрос).

7) Параметр <время> задает максимально допустимую продолжительность выполнения запроса (от 1 до 65535 сек.). Если запрос в отведенное для него время не был выполнен, его обработка прекращается с выдачей соответствующего кода завершения.

8) Конструкция WITH PRIORITY <приоритет> устанавливает заданный приоритет (значение в диапазоне от 0 до 255) выполняемому запросу. Если задать приоритет больше текущего приоритета пользователя, от имени которого подается запрос, то выдается код завершения 1022 («Нарушение привилегий»).

9) Конструкция WITH PRIORITY <приоритет> назначает приоритет только тому запросу, в котором она указана. На приоритет любых последующих запросов она не влияет.

10) Для таблицы, созданной с атрибутом NODE, <запрос корректировки> с локального сервера недоступен.

11) При модификации записей таблицы <логическое выражение>, ассоциированное с каждым генерируемым столбцом, вычисляется заново, и столбец получает соответствующее значение.

```
create or replace table tst
("Сумма" numeric, "Скидка" numeric,  "Итого"
generated always as (("Сумма"*(100-"Скидка")/100)));
insert into tst ("Сумма", "Скидка", "Итого" ) values
 (1000,5,default);
insert into tst ("Сумма", "Скидка", "Итого" ) values
 (2000,7,default);
select * from tst;

|  1000.0|  5.0|   950.0|
|  2000.0|  7.0|  1860.0|

update tst set "Скидка"="Скидка"+0.5;
select * from tst;

|  1000.0|  5.5|   945.0|
|  2000.0|  7.5|  1850.0|
```

12) Модификация значений идентификационных столбцов разрешена только для столбцов, созданных с атрибутом GENERATED BY DEFAULT.

13) Для таблиц «в памяти» <запрос корректировки> в режиме OPTIMISTIC не поддерживается.

14) Разрешены внешние ссылки в SELECT-списке. Кроме того, при проверке наличия столбцов под агрегатными функциями и без них эти внешние ссылки не учитываются (т.е. рассматриваются как константы).

```
Пример запроса с использованием такой ссылки (к таблицам S, SP в
 дистрибутивной базе):
update s set status = (select s.status+count(*) from sp where
 s.snum = sp.snum);
```

15) Любой JOIN (больше одной таблицы во FROM), либо обращение к VIEW на основе JOIN, либо при работе в режиме /COMPATIBILITY=STANDARD, либо VIEW было создано при работе в таком режиме автоматически является необновляемым (будет выдан код завершения 2162).

Пример

Непорционная загрузка BLOB-данных (например, сразу 1 Мбайт). В этом случае необходимо использовать параметрический запрос:

```
update T_BLOB set BLB = ? WHERE ...;
  или
insert into T_BLOB (..., BLB) values (..., ?);
```

привязав к параметру для BLOB-значения соответствующий массив.

Удаление записи (DELETE)

Функция

Определение оператора удаления записей таблицы.

Спецификация

[1] <запрос удаления>::=
DELETE FROM [<имя схемы>.]<имя объекта> [[AS] <псевдоним таблицы>]
[JOIN <соединяемая таблица>[, …]]
[<WHERE-спецификация>]
[WAIT | NOWAIT]
[{QUANT| QUANTUM} TIMEOUT <время>]
[WITH PRIORITY <приоритет>]
[2] <имя объекта>::= <имя таблицы> | <имя представления>
[3] <соединяемая таблица>::=
<первичная таблица> | <порожденная таблица>

Синтаксические правила

1) Допустимые привилегии для <имени объекта> должны включать DELETE.
2) Для соединяемых таблиц (с конструкцией JOIN) привилегия DELETE требуется только для левых таблиц соответствующих запросов, для правых таблиц требуется привилегия SELECT.
3) Таблица с <именем таблицы> не должна быть только считываемой таблицей (т.е. системной таблицей), <имя представления> не должно ссылаться на необновляемое представление (представление, основанное на выборке из нескольких таблиц). В данном случае, если <WHERE-спецификация> не задана, удаляются все записи.

   Удаление всех записей таблицы:

```
DELETE FROM Person;
```

4) Если <WHERE-спецификация> задана, то она применяется к каждой строке таблицы (представления), и строки, для которых <WHERE-спецификация> истинна, удаляются.

Удаление по условию выборки:

```
delete from auto where color ='green';
```

Удаление из таблицы TST дубликатов поля I:

```
delete from tst
 where rowid in (select rowid from tst m1 where exists (select *
 from tst m2 where m1.i=m2.i and m1.rowid < m2.rowid));
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

5) Если задана конструкция JOIN, то она должна содержать список реальных или порожденных таблиц, соединяемых с таблицей, из которой удаляются строки. Условие соединения всех таблиц задается в <WHERE-спецификации>.

```
DELETE FROM Auto JOIN Person
 WHERE auto.personid=person.personid
   AND auto.color in ('BLACK','WHITE');
```

Примечание
Синтаксис оператора DELETE с соединяемыми таблицами не соответствует стандарту SQL, т.к. в этом стандарте не прописан механизм удаления записей при наличии соединяемых таблиц. Согласно стандарту SQL можно удалять напрямую или через обновляемое представление записи только из одной таблицы или выборки из неё без соединения с другими таблицами.

6) Для удаления записей из <имени объекта> рекомендуется, чтобы каждой записи из <имени объекта> соответствовала только одна запись первой и последующих соединяемых таблиц. В противном случае результат выполнения запроса не предсказуем.

Общие правила

1) Если при выполнении операции удаления (в том числе и каскадной) встречается заблокированная запись, то выдается код завершения 135 («Строка таблицы заблокирована другим пользователем»), даже если задан модификатор WAIT.
2) Удаление записей с использованием соединения таблиц происходит следующим образом:
   - выполняется соединения таблиц;
   - из полученного результирующего набора удаляются те записи, которые удовлетворяют <WHERE-спецификации>.
3) Параметр <время> задает максимально допустимую продолжительность выполнения запроса (от 1 до 65535 сек.). Если запрос в отведенное для него время не был выполнен, его обработка прекращается с выдачей соответствующего кода завершения.
4) Конструкция WITH PRIORITY <приоритет> устанавливает заданный приоритет (значение в диапазоне от 0 до 255). Если задать приоритет больше текущего приоритета пользователя, от чьего имени подается запрос, то выдается код завершения 1022 («Нарушение привилегий»).
5) Конструкция WITH PRIORITY <приоритет> назначает приоритет только тому запросу, в котором она указана. На приоритет любых последующих запросов она не влияет.
6) Для таблицы, созданной с атрибутом NODE, <запрос удаления> с локального сервера недоступен.
7) Разрешается удалять записи со спецификацией действия NO ACTION (RESTRICT) и ссылающиеся на самих себя.
8) Для таблиц «в памяти» <запрос удаления> в режиме OPTIMISTIC не поддерживается.

Тестовые данные:

Пусть имеются таблицы сотрудников (persons), отделов (departments), этажей (floors).

Таблица persons связана с departments по номеру отдела (d_id), departments связана с floors по номеру этажа (num_f).

```
create or replace table floors (num_f int, f_name char(20));
insert into floors values (1, 'First');
insert into floors values (2, 'Second');
insert into floors values (3, 'Third');
insert into floors values (4, 'Fourth');
insert into floors values (5, 'Fifth');
insert into floors values (6, 'Sixth');

create or replace table departments(d_id int, d_name char(20),
 num_f int);
insert into departments values (1, 'Sales',           1);
insert into departments values (2, 'IT-technologies', 3);
insert into departments values (3, 'Finance',         4);
insert into departments values (4, 'Management',      4);
insert into departments values (5, 'Design',          3);

create or replace table persons(p_id int, p_name char (20), d_id
 int);
insert into persons values (1, 'John',  3);
insert into persons values (2, 'Mary',  2);
insert into persons values (3, 'Kate',  4);
insert into persons values (4, 'Jack',  2);
insert into persons values (5, 'Peter', 7);
insert into persons values (6, 'Ann',   5);
```

Удалить из таблицы departments отделы, которые находятся на этаже 'First'.

```
DELETE FROM departments d JOIN floors f
 WHERE (f.num_f = d.num_f) AND (f.f_name = 'First');

SELECT * FROM departments;
```

D_ID	D_NAME	NUM_F
2	IT-technologies	3
3	Finance	4
4	Management	4
5	Design	3

В результирующем наборе в списке отделов (departments) теперь не содержится отдел 'Sales', т.к. он находится на этаже 'First'.

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

Позиционная корректировка записи (UPDATE OF CURSOR)

Функция

Определение оператора позиционной корректировки записи таблицы.

Спецификация

[1] <позиционная корректировка> : : =
UPDATE {[<имя схемы>.]<имя таблицы> | <имя представления>}
[#[<группа>]#[<RAL>]#[<WAL>]] [[AS] <псевдоним таблицы>]
SET <значение корректировки>[, …]
WHERE CURRENT OF {CURSOR | <имя курсора>}
[WAIT | NOWAIT]
[{QUANT | QUANTUM} TIMEOUT <время>] [WITH PRIORITY <приоритет>]

[2] <значение корректировки> : : =
<имя столбца>[#[<группа>]#[<RAL>]#[<WAL>]] =
{<значимое выражение>
|<подзапрос>
|NULL
|DEFAULT
|EXTFILE(NULL | ? |<спецификация файла>[, <имя фильтра>])}

Синтаксические правила

1) <RAL>, <WAL> – уровни доступа на чтение и запись (см. документ «Администрирование комплекса средств защиты данных», раздел «Уровни доступа»).

Примечание
Поддерживается только в СУБД ЛИНТЕР БАСТИОН.

2) Допустимые привилегии для <имени таблицы> должны включать UPDATE.
3) <Имя таблицы> должно быть таблицей, указанной в первой <FROM-спецификации> текущего <запроса выборки>.
4) Результат <запроса выборки> и таблица <имя таблицы> должны быть обновляемыми.

Общие правила

1) Запрос <позиционной корректировки> должен подаваться после выполнения (по тому же каналу) <запроса выборки>, т.е. с помощью этого оператора можно изменить в таблице ту или иную запись, вошедшую в ответ данного <запроса выборки>.
2) Общая схема выполнения позиционного удаления:
   - открыть курсор;
   - сделать выборку (SELECT-запрос) из таблицы или обновляемого представления;
   - переместиться по выборке в нужную строку (например, в интерактивной программе в ту строку, которую пользователь желает удалить), т.е. установить текущую строку ответа;
   - выполнить конструкцию <позиционная корректировка>.

3) При выполнении <позиционной корректировки> корректируется запись, являющаяся текущей строкой ответа <запроса выборки>.
4) <Значимое выражение> в <позиционной корректировке> не должно включать агрегатных функций.
5) Одно и то же <имя столбца> не должно появляться в <значении корректировки> более одного раза.
6) Областью видимости <имени таблицы> является весь оператор <позиционной корректировки>.
7) Если <значимое выражение> содержит столбец из <имени таблицы>, то это ссылка на значение данного столбца в изменяемой строке перед тем, как любое значение этой строки будет изменено.
8) Параметр <время> задает максимально допустимую продолжительность выполнения запроса (от 1 до 65535 сек.). Если запрос в отведенное для него время не был выполнен, его обработка прекращается с выдачей соответствующего кода завершения.
9) Конструкция WITH PRIORITY <приоритет> устанавливает заданный приоритет (значение в диапазоне от 0 до 255) выполняемому запросу. Если задать приоритет больше текущего приоритета пользователя, от имени которого подается запрос, то выдается код завершения 1022 («Нарушение привилегий»).
10) Конструкция WITH PRIORITY <приоритет> назначает приоритет только тому запросу, в котором она указана. На приоритет любых последующих запросов она не влияет.
11) Для таблицы, созданной с атрибутом NODE, <позиционная корректировка> с локального сервера недоступна.
12) Для таблиц «в памяти» <позиционная корректировка> в режиме OPTIMISTIC не поддерживается.

Примеры

1)

```
select * from auto where make='LAMBORGHINI';
update auto set CHKMILE = CHKMILE + 10 where current of cursor;
```

2)

```
create or replace procedure proc1 () result int
code
  execute direct "select * from auto;";//
  execute "update auto set year = year + 1 where current of
 cursor;";//
  return errcode();//
exceptions
  when all then resignal;//
end;
```

3)

```
create or replace procedure proc2 () result int
declare
  var cur1 typeof(auto);//
```

```
code
  open cur1 as "cursor_cur1" for direct "select * from auto;";//
  execute "update auto set year = year + 1 where current of
 \"cursor_cur1\";";//
  return errcode();//
exceptions
  when all then resignal;//
end;
```

Позиционное удаление записи (DELETE OF CURSOR)

Функция

Определение оператора позиционного удаления записи из таблицы.

Спецификация

[1] <позиционное удаление>::=
DELETE FROM [<имя схемы>.]
{<имя таблицы> |<имя представления>} [[AS] <псевдоним таблицы>]
WHERE CURRENT OF {CURSOR | <имя курсора>} [WAIT | NOWAIT]
[{QUANT | QUANTUM} TIMEOUT <время>] [WITH PRIORITY <приоритет>]

Синтаксические правила

1) Допустимые привилегии для <имени таблицы> должны включать привилегию DELETE.
2) <Имя таблицы> должно быть таблицей, указанной в первой <FROM-спецификации> текущего <запроса выборки>.
3) Результат <запроса выборки> и таблица <имя таблицы> должны быть обновляемыми.

Общие правила

1) Запрос <позиционного удаления> должен подаваться после выполнения (по тому же каналу) <запроса выборки>, т.е. с помощью этого оператора можно удалить из таблицы ту или иную строку, вошедшую в ответ данного <запроса выборки>.
2) Общая схема выполнения позиционного удаления:
   - открыть курсор;
   - сделать выборку (SELECT-запрос) из таблицы или обновляемого представления;
   - переместиться по выборке в нужную строку (например, в интерактивной программе в ту строку, которую пользователь желает удалить), т.е. установить текущую строку ответа;
   - выполнить конструкцию <позиционное удаление>.
3) При выполнении <позиционного удаления> удаляется строка, являющаяся текущей строкой ответа на выполненный <запрос выборки>.
4) Запрос <позиционного удаления> может подаваться после выполнения иерархического <запроса выборки>.

5) Запрос <позиционного удаления> выполняется по тому каналу, по которому он был подан (а не по каналу, по которому был подан соответствующий SELECT-запрос).
6) Параметр <время> задает максимально допустимую продолжительность выполнения запроса (от 1 до 65535 сек.). Если запрос в отведенное для него время не был выполнен, его обработка прекращается с выдачей соответствующего кода завершения.
7) Конструкция WITH PRIORITY <приоритет> устанавливает заданный приоритет (значение в диапазоне от 0 до 255) выполняемому запросу. Если задать приоритет больше текущего приоритета пользователя, от имени которого подается запрос, то выдается код завершения 1022 («Нарушение привилегий»).
8) Конструкция WITH PRIORITY <приоритет> назначает приоритет только тому запросу, в котором она указана. На приоритет любых последующих запросов она не влияет.
9) Для таблицы, созданной с атрибутом NODE, <позиционное удаление> с локального сервера недоступно.
10) Для таблиц «в памяти» <позиционное удаление> в режиме OPTIMISTIC не поддерживается.

Слияние данных (MERGE)

Функция

Определение оператора слияния (синхронизации) приемника и источника данных.

В некоторых информационных системах встречается потребность в передаче строк из таблицы, созданной или обновленной при выполнении транзакции (транзакционной или исходной таблицы), в некоторую основную (целевую) таблицу БД. Исходная таблица может содержать как обновленные варианты строк, существующие в целевой таблице, так и новые строки, которые должны быть добавлены в целевую таблицу.

С помощью традиционных операторов манипулирования данными содержимое исходной таблицы можно перенести в целевую таблицу за два шага:

1) выполнить оператор UPDATE для всех строк целевой таблицы, для которых имеются модифицированные «двойники» в исходной таблице;
2) выполнить оператор INSERT для добавления в целевую таблицу тех строк исходной таблицы, для которых в целевой таблице нет «двойников».

Оператор MERGE реализует эти функции за один шаг.

Спецификация

[1] <слияние данных> : : =
MERGE INTO [<имя схемы>.]целевая таблица [[AS] <псевдоним таблицы>]
USING [<имя схемы>.]исходная таблица [[AS] <псевдоним таблицы>]
ON <условие слияния> <спецификация слияния>
[2] <условие слияния> : : = <логическое выражение>
[3] <спецификация слияния> : : =
<слияние с сопоставлением>
| <слияние без сопоставлением>
[4] <слияние с сопоставлением> : : =
WHEN MATCHED THEN <корректировка строки>
[5] <слияние без сопоставления> : : =
WHEN NOT MATCHED THEN <вставка строки>

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

[6] <корректировка строки>::=
UPDATE SET <значение корректировки>[, ...]
[7] <значение корректировки>::=
<имя столбца целевой таблицы> [#[<группа>]#[<RAL>]#[<WAL>]] =
{<значимое выражение>
|<подзапрос>
|DEFAULT
|NULL
|EXTFILE(NULL | ? |<спецификация файла>[, <имя фильтра>])}
[8] <вставка строки>::=
INSERT [(<список столбцов целевой таблицы>)]
VALUES <список значений>
[9] <список столбцов целевой таблицы>::=
<имя столбца целевой таблицы> [#[<группа>]#[<RAL>]#[<WAL>]][, ...]
[10] <имя столбца целевой таблицы>::= <идентификатор>
[11] <список значений>::=
(<элемент списка>[{, <элемент списка>}...])
[12] <элемент списка>::= <значимое выражение> |<литерал>

Синтаксические правила

1) `<RAL>`, `<WAL>` – уровни доступа на чтение и запись (см. документ «Администрирование комплекса средств защиты данных», раздел «Уровни доступа»).

Примечание
Поддерживается только в СУБД ЛИНТЕР БАСТИОН.

2) `<Целевая таблица>` – имя объекта БД (базовая таблица или обновляемое представление), который должен быть синхронизирован с `<исходной таблицей>` по заданному `<условию слияния>`.
3) `<Целевая таблица>` не может быть удаленной таблицей, таблицей «в памяти», временной таблицей, циклической таблицей или реплицируемой таблицей.
4) `<Псевдоним>` задает альтернативное имя, используемое для указания ссылок на целевую или исходную таблицу.
5) `<Исходная таблица>` – имя объекта БД (базовая/временная/глобальная/удаленная таблица (за исключением циклической), подзапрос, любое представление), который является источником данных для синхронизации (слияния) с `<целевой таблицей>`.
6) Оператор `MERGE` может иметь только одну конструкцию `<слияние с сопоставлением>`.
7) Оператор `MERGE` может иметь только одну конструкцию `<слияние без сопоставления>`.
8) Должно быть указано либо `<слияние сопоставление>`, либо `<слияние без сопоставления>`, либо обе конструкции одновременно (в любом порядке).
9) `<Значимое выражение>` может включать в себя имена столбцов как исходной, так и целевой таблицы.

Общие правила

1) Выполнение `MERGE` требует привилегии `UPDATE` на обновляемую таблицу при наличии конструкции `WHEN MATCHED` и привилегии `INSERT` на обновляемую

таблицу при наличии конструкции WHEN NOT MATCHED, а также привилегии SELECT на все используемые таблицы.

2) Алгоритм выполнения оператора.

   Пусть T1 – <исходная таблица>, а T2 – <целевая таблица>

   Тогда алгоритм выполнения оператора определяется следующим образом:

   - строки таблицы T1 просматриваются в некотором порядке. Пусть R1 – очередная строка T1. Для этой строки вычисляется <условие слияния> например, синхронизация заказов на определенную дату

```
T1.<N_заказа>=T2.<N_заказа> AND T2.<дата>='28.04.2010';
```

   - если значением <условия слияния> является TRUE, то есть в <целевой таблице> есть строка с данным номером заказа за указанную дату, то:
     - если в операторе содержится раздел <слияние с сопоставлением>:
       - в этой строке корректируются значения тех столбцов, которые указаны в <списке столбцов> раздела <корректировка строки> (например, вносятся сведения об изменении заказа). В <списке столбцов> указываются имена столбцов таблицы T1, т.е. что строка R2 будет модифицироваться на основе значений столбцов строки R1;
       - в противном случае строка R1 игнорируется;
   - если значением <условия слияния> является FALSE, то есть в <целевой таблице> нет строки с данным номером заказа на указанную дату, то:
     - если в операторе содержится раздел <слияние без сопоставления>:
       - в таблицу T2 вставляется строка, специфицируемая списком выражений раздела <вставка строки>. В <списке столбцов> указываются имена столбцов таблицы T1, т.е. строка, заново вставляемая в таблицу T2, будет формироваться на основе значений столбцов строки R1);
       - в противном случае строка R1 игнорируется.

3) Для заданного <условия слияния> каждой строке таблицы T1 должна соответствовать не более чем одна строка таблицы T2. В противном случае генерируется код завершения 76 («Получено результирующее множество из нескольких записей (ожидалась одна запись)»). Таким образом, на одну строку <исходной таблицы> допускается только одна операция вставки или обновления.

4) <Условие слияния> определяет критерии совпадения строк целевой и исходной таблиц. Задается с помощью конструкции <логическое выражение>.

```
Пусть исходная таблица содержит список автовладельцев, заменивших
 свой автомобиль и его цвет в течение 1970 года. Необходимо внести
 соответствующие изменения в таблицу AUTO.
create or replace table upgrade_auto (id int, mod_auto char (20),
 color char (10));

insert into upgrade_auto (id, mod_auto, color) values (10, 'FORD',
 'BLACK');
insert into upgrade_auto (id, mod_auto, color) values (5000,
 'MASDA', 'RED');
```

```
MERGE INTO AUTO
USING upgrade_auto as src ON (auto.personid=src.id and
 auto.year=70)
 WHEN MATCHED THEN UPDATE SET auto.model=src.mod_auto,
 auto.color=src.color;
```

5) В случае указания <слияния с сопоставлением> выполняется корректировка значениями из <исходной таблицы> всех строк <целевой таблицы>, которые удовлетворяют <условию слияния>.

6) Обновляемые столбцы в <целевой таблице> не должны быть автогенерируемыми столбцами (последовательностями, autoinc, autoinc range и т.п.).

7) Если при добавлении строк <исходная таблица> содержит значения для вставки в автогенерируемые столбцы <целевой таблицы>, то фиксируется ошибочный код завершения, точно так же, как и при выполнении соответствующего INSERT. Чтобы избежать этого, можно, как и для INSERT, либо вообще не задавать явно значения автогенерируемых столбцов, либо задавать такие значения, которые не вызовут этих проблем (большие максимального добавлявшегося значения AUTOINC, не попадающие в диапазоны AUTOINC RANGE и т.п.).

8) <Корректировка строки> задает список имен столбцов <целевой таблицы>, которые должны быть обновлены, и значений, которые необходимо использовать для их обновления.

```
Обновить список автовладельцев, которые в порядке обмена старых
 авто на новые заменили свои автомобили. Таблица upgrade_auto
 содержит идентификаторы таких автовладельцев.

MERGE INTO AUTO
USING upgrade_auto as src ON (auto.personid=src.id and)
 WHEN MATCHED THEN UPDATE SET auto.model='Калина',
 auto.make='АвтоВАЗ';
```

9) В случае указания <слияния без сопоставления> выполняется вставка всех строк из <исходной таблицы> в <целевую таблицу>, для которых не выполнено <условие слияния>.

10) <Вставка строки> задает <список столбцов> <целевой таблицы>, в которые должны быть вставлены значения из <списка значений> <исходной таблицы>.

```
Добавить в таблицу AUTO информацию о новых автовладельцах.

MERGE INTO AUTO
USING upgrade_auto as src ON (auto.personid=src.id)
 WHEN NOT MATCHED THEN INSERT (auto.personid, auto.model,
 auto.color) VALUES (src.id, src.mod_auto, src.color);
```

11) На все операции вставки или обновления, указанные применительно к целевой таблице распространяются все ограничения, определенные для этой таблицы, включая все каскадные ограничения ссылочной целостности.

12) Для каждой операции вставки и обновления запускаются все соответствующие триггеры, определенные для целевой таблицы. Очередность запуска триггеров устанавливается ядром СУБД.

13) Если в <целевой таблице> определены триггеры INSTEAD OF UPDATE или INSTEAD OF INSERT, то операции обновления или добавления строк не выполняются. Вместо этого запускаются соответствующие триггеры.

14) Чтобы полученные результаты были гарантированно верными, должны быть правильно указаны условия поиска, которые используются для сопоставления исходных и целевых строк. Рекомендуется придерживаться следующих правил:

- укажите в предложении ON <условие слияния> только те условия поиска, которые определяют критерий совпадения данных в исходных и целевых таблицах. То есть необходимо указать только те столбцы целевой таблицы, которые сравниваются с соответствующими столбцами исходной таблицы;
- не включайте сравнения с другими значениями, такими, как константа;
- для фильтрации данных используйте представления.

Примеры

1) Одновременная корректировка и вставка записей.

```
MERGE INTO AUTO
USING upgrade_auto as src ON (auto.personid=src.id and)
 WHEN MATCHED THEN UPDATE SET auto.model='Калина',
 auto.make='АвтоВАЗ'
 WHEN NOT MATCHED THEN INSERT (auto.personid, auto.model,
 auto.color) VALUES (src.id, src.mod_auto, src.color);
```

2) Использование вместо исходной таблицы подзапросов.

2.1 Обновить информацию только о пользователе с идентификатором 10 и 11 (без создания исходной таблицы).

```
MERGE INTO AUTO
USING (SELECT 10 as id, 'HONDA'as mod_auto, 'GREEN'as color
        UNION
       SELECT 11 as id, 'BEANTLY' as mod_auto, 'RED'as color ) as
 src
  ON (auto.personid=src.id)
WHEN MATCHED THEN UPDATE SET auto.model='Калина',
 auto.make='АвтоВАЗ'
WHEN NOT MATCHED THEN INSERT (auto.personid, auto.model,
 auto.color) VALUES (src.id, src.mod_auto, src.color);
```

2.2 Все владельцы автомобилей марки MUSTANG BOSS 351 белого цвета, проживающие в городе SAN FRANCISCO, перекрасили свои автомобили в черный цвет. Отразить этот факт в таблице AUTO.

а) составляем представление – список всех жителей города SAN FRANCISCO, владеющих автомобилем марки MUSTANG BOSS 351 белого цвета:

```
create or replace view ListAutoUser as
select person.personid, person.city, auto.model from auto, person
 where person.personid=auto.personid
   and person.city='SAN FRANCISCO'
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
   and auto.model='MUSTANG BOSS 351'
   and auto.color='WHITE';
```

б) вносим изменение в таблицу AUTO

```
MERGE INTO AUTO
USING (select personid as id from ListAutoUser) as src
   ON (auto.personid=src.id)
 WHEN MATCHED THEN UPDATE SET auto.color='BLACK';
```

3) Пусть есть таблица T с двумя целочисленными столбцами I и J:

```
 I           J
 -           -
|          1|          1|
|          2|          1|
|          3|          3|
|          4|          4|
```

Выполняем слияние этой таблицы с собой:

```
merge into T using T as A on T.I = A.J
 when matched then update set T.J = T.J + 1
 when not matched then insert (T.I, T.J) values (0, 0);
```

Результат:

```
 I           J
 -           -
|          1|          3|
|          2|          1|
|          3|          4|
|          4|          5|
```

Добавление новых строк не произошло, т.к. для каждой строки исходной таблицы A (неважно, что она та же, что и целевая таблица) нашлась строка из T, в которой T.I = A.J. Именно поэтому при выполнении запроса всегда выполняется конструкция `when matched then`, т.е. всегда только корректировка.

К первой строке таблицы значение прибавлено дважды, т.к. для двух строк таблицы A (первая и вторая строка) в T нашлась одна строка (первая), для которой T.I = A.J, поэтому для этой первой строки замена производилась дважды.

4) Добавление в таблицу новых строк или корректировка существующих без использования исходной таблицы.

```
create or replace table "Count of children" (FIO char(15), total
 int);

MERGE INTO "Count of children" AS child
USING (VALUES('Иванов И. И.' FIO, 1 total)) AS details ON
 (child.FIO = details.FIO)
 WHEN MATCHED THEN UPDATE SET child.total = child.total +
 details.total
```

```
 WHEN NOT MATCHED THEN INSERT (child.FIO, child.total) VALUES
 (details.FIO, details.total);
```

5) В разделе INSERT, UPDATE имя целевой таблицы перед именем столбцов можно не указывать. Выше приведенный запрос можно написать так:

```
MERGE INTO "Count of children" AS child
USING (VALUES('Иванов И. И.' FIO, 1 total)) AS details ON
 (child.FIO = details.FIO)
 WHEN MATCHED THEN UPDATE SET total = total + details.total
 WHEN NOT MATCHED THEN INSERT (FIO, total) VALUES (details.FIO,
 details.total);
```

Пакетное добавление (START APPEND)

Пакетное добавление данных выполняется командой PUTM интерфейса нижнего уровня СУБД ЛИНТЕР (см. документ «Интерфейс нижнего уровня», пункт «Пакетное добавление данных»). Эта команда доступна только в режиме пакетного добавления, который инициируется SQL-запросом START APPEND … и заканчивается при выполнении SQL-запроса END APPEND… В пакетном режиме выполнение других команд интерфейса нижнего уровня недопустимо (кроме команд COMMIT, ROLLBACK и CLOSE).

Начать пакетное добавление

Функция

Определение запроса инициирования пакетного добавления.

Спецификация

[1] <начать пакетное добавление>::=
START APPEND INTO [<имя схемы>.] {<имя таблицы> |<имя представления>}
[#[<группа>]#[<RAL>]#[<WAL>]] [<формат пакета>]
[<формат столбца>]
[VALUES (? [,…])]
[WAIT | NOWAIT]
[{QUANT | QUANTUM} TIMEOUT <время>]

[2] <формат пакета>::= BYTE | CHAR

[3] <формат столбца>::= [(<спецификация столбца>[, …])]

[4] <спецификация столбца>::=
<имя столбца> [#[<группа>]#[<RAL>]#[<WAL>]]
[BYTE | CHAR | '<дата-время литерал>']

[5] <формат даты>::= <символьный литерал>

Синтаксические правила

1) <RAL>, <WAL> – уровни доступа на чтение и запись (см. документ «Администрирование комплекса средств защиты данных», раздел «Уровни доступа»).

Примечание
Поддерживается только в СУБД ЛИНТЕР БАСТИОН.

2) <Имя таблицы> должно ссылаться на пользовательскую базовую таблицу или обновляемое представление.

© Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

3) <Дата-время литерал> допустимо применять только для столбцов с типом данных DATE.
4) <Имя столбца> не должно быть именем BLOB-столбца (столбец типа EXTFILE является допустимым).
5) Опция VALUES задает список параметров для данной конструкции. Количество параметров (знаков ?) должно строго соответствовать количеству загружаемых столбцов. Используется после претрансляции для автоматического определения типов данных и длин загружаемых значений.

Общие правила

1) <Формат пакета> задает формат представления данных, используемый для всех столбцов таблицы. В формате BYTE преобразование полей записи пакета не производится – данные в таблицу записываются в том виде, в каком они содержатся во входном буфере данных. В формате CHAR выполняется преобразование данных из символьного вида к типу данных соответствующего столбца. Если <формат пакета> не задан, по умолчанию используется CHAR.

Примечание
При формировании пакета PUTM для значений переменной длины с последующим преобразованием их к типу BYTE требуется обязательное выделение двух байт, содержащих длину значения переменного типа данных.

2) <Формат столбца> задает формат индивидуального представления данного столбца. Преобразование по форматам BYTE и CHAR выполняется подобно преобразованию по этим форматам для всей таблицы. <Формат даты> используется для преобразования столбцов, имеющих тип данных DATE. Если <формат столбца> задан без указания типа преобразования, по умолчанию используется CHAR.
3) Если одновременно заданы <формат пакета> и <форматы столбцов>, то высший приоритет имеет <формат столбца>.
4) Параметр <время> задает максимально допустимую продолжительность выполнения запроса (от 1 до 65535 сек.). Если запрос в отведенное для него время не был выполнен, его обработка прекращается с выдачей соответствующего кода завершения.
5) Если для таблицы, в которую выполняется пакетная загрузка данных, задано ограничение целостности CHECK, и хотя бы одна из записей в порции не удовлетворяет условию CHECK, то вся порция не будет добавлена.

Примечание
При пакетной вставке данных триггеры, настроенные на вставку данных, срабатывать не будут.

Пример

См. приложение 10 в документе «Интерфейс нижнего уровня».

Закончить пакетное добавление (END APPEND)

Функция

Определение запроса окончания пакетного добавления.

Спецификация

[1] <закончить пакетное добавление>::=
END APPEND [INTO [<имя схемы>.]{<имя таблицы> |<имя представления>}]

Синтаксические правила

1) <Имя таблицы> должно ссылаться на таблицу, указанную в запросе <начать пакетное добавление>, выполненном ранее по тому же соединению с БД.

SQL-операторы с параметрами

Общие сведения

Стандарт языка SQL предусматривает два способа формирования SQL-запросов: интерактивный и динамический. При использовании интерактивного способа текст SQL-запроса полностью определяется при формулировании запроса, и запрос сразу готов к выполнению. Динамический способ формирования SQL-запросов используется в приложениях. Формирование запроса в этом случае может осуществляться двумя способами.

Первый способ

Многократное выполнение SQL-оператора с заменой формальных параметров фактическими (см. документ «Встроенный SQL»).

SQL-оператор подготавливается для выполнения и хранится в пользовательском приложении. Такой оператор может быть выполнен либо сразу, либо позднее, причем допускается любое количество вызовов. Подготовленный оператор может быть полностью специфицирован (и тогда он совпадает с командным SQL-запросом) или включать в себя неопределенные значения, называемые параметрами. Параметры могут быть входными и выходными. В выходные параметры помещаются результаты выполнения запроса. Присвоение значений входным параметрам и окончательное формирование текста SQL-оператора возлагается на приложение. Способ присвоения значений входным параметрам определяется логикой и интерфейсом приложения (программное вычисление, интерактивный ввод, извлечение из БД). Процесс связывания входных параметров с частично подготовленным SQL-оператором определяется инструментальными средствами разработки приложения.

Данный способ выполнения SQL-запросов подобен функциям или процедурам: функция имеет тело функции (которое не меняется) и набор аргументов (параметров), которые при каждом вызове имеют различные значения и заменяют в теле функции формальные параметры. Аналогично: частично подготовленный SQL-оператор имеет неизменяемое тело оператора и набор формальных параметров, вместо которых перед каждым выполнением запроса подставляются их фактические значения.

После того, как необходимость в использовании такого оператора отпадает, он может быть удален, при этом будут освобождены занятые этим оператором системные ресурсы.

Второй способ

Однократное выполнение оператора с заменой формальных параметров фактическими.

В этом случае SQL-оператор подготавливается к выполнению и сразу же выполняется, после чего все использованные этим оператором ресурсы немедленно освобождаются.

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

Определение значений входных параметров и привязка их к тексту SQL-оператора осуществляются так же, как и в первом случае, однако разрыв между стадией подготовки оператора и его выполнением не предусмотрен. Данный способ предполагает полную подготовку SQL-оператора перед каждым выполнением, поэтому в процессе подготовки оператора, кроме привязки значений параметров, можно менять и собственно текст SQL-оператора.

SQL-параметр

Функция

Определение формального параметра.

Спецификация

[1] <SQL-параметр>::= {<неименованный параметр> |<именованный параметр>}[(<тип данных>)]
[2] <неименованный параметр>::= ?
[3] <именованный параметр>::= :<имя параметра>
[4] <имя параметра>::= <идентификатор>

Синтаксические правила

1) В качестве `<типа данных>` может использоваться любой тип данных, кроме BLOB-данных.
2) `<Тип данных>` должен обязательно указываться в тех случаях, когда SQL-транслятор не может самостоятельно определить тип данных параметра. Например, если `<SQL-параметр>` привязан к столбцу таблицы, то в качестве типа данных такого параметра будет использован тип данных столбца; если же `<SQL-параметр>` – аргумент некой функции, то тип данных этого аргумента не известен SQL-транслятору и поэтому его надо явно указывать.

```
select hextoraw(? (char(10)));
```

3) `<Имя параметра>` в `<именованном параметре>` может совпадать с именами объектов БД (таблиц, столбцов и т.п.) и встроенных в SQL СУБД ЛИНТЕР функций.
4) Для типа данных `EXTFILE` допускается использовать конструкцию `EXTFILE(?)`.
5) Для всех символьных параметров применяется установленная в канале кодировка по умолчанию (если клиентское приложение не укажет её явно).

Параметрическое выражение

Функция

Определение выражения, включающего формальные параметры.

Спецификация

[1] <параметрическое выражение>::= <параметрический операнд> | <параметрический операнд> <операция> <параметрическое выражение>
[2] <операция>::= <оператор сравнения> | <арифметический оператор>
[3] <параметрический операнд>::= <литерал> | <параметр> | встроенная функция
[4] <арифметический оператор>::= <сложение> | <вычитание> | <умножение> | <деление>
[5] <сложение>::= +

[6] <вычитание>::= -
[7] <умножение>::= *
[8] <деление>::= /

Синтаксические правила

1) Применение <параметрического выражения> допускается в следующих конструкциях:
   - <предикат сравнения>;
   - <интервальный предикат>;
   - <предикат вхождения>;
   - <предикат подобия>;
   - <WHERE-спецификация>;
   - <GROUP BY-спецификация>;
   - <HAVING-спецификация>;
   - <запрос выборки>;
   - <запрос удаления>;
   - <запрос добавления>;
   - <запрос корректировки>;
   - <спецификация типа>;
   - <выбор значения по условию>;
   - <выполнение процедуры>;
   - встроенные функции.
2) <Параметрическое выражение> может использоваться в любом месте, где синтаксические правила допускают значение <литерала> в соответствующем интерактивном варианте.

Примечание
Из данного ограничения следует, что параметр не может представлять явно или косвенно имена объектов БД (например, явные имена таблицы, столбца, представления, синонима, роли и т.п. или номера столбцов сортировки в конструкции ORDER BY).

3) В тех случаях, когда из контекста нельзя установить тип параметра, например, в выражении ? + ?, необходимо явное указание типов параметров.
4) Если результат подзапроса вставляется в таблицу, то параметры подзапроса по умолчанию получают тип данных соответствующих столбцов этой таблицы.

```
create or replace table test (i int, c char(20));
insert into test(i,c) select ?, make from auto where rowid=100;
Параметр 1 (INTEGER)>
```

5) Одноименные параметры должны иметь совпадающие типы данных.

   Корректная конструкция, т.к. make и model имеют один и тот же тип char(20):

```
select * from auto where make=:m and model=:m;
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

Некорректная конструкция, т.к. bodytype имеет другой тип char(15):

```
select * from auto where make=:m and bodytype=:m;
```

Корректная конструкция:

```
select * from auto where make=:m and bodytype=:m(char(20));
```

6) Конструкция CAST ? AS <тип данных> эквивалентна конструкции ? (<тип данных>).
7) Значения <неименованных параметров> подставляются в SQL-оператор в порядке их следования в этом операторе.
8) <Параметры> должны иметь тип данных, совпадающий с типом данных соответствующих им <значимых выражений>, или приводиться к нему.
9) В конструкции вида <значение>||<параметр> тип параметра по умолчанию приводится к типу значения.
10) Параметр, для которого задан строковый тип без явного указания длины, считается имеющим длину 1 (2 для UNICODE).
11) Разрешена конструкция INTO для задания выходных параметров в <запросе выборки>:

```
<запрос выборки> ::=
  SELECT [ALL | DISTINCT] <список выборки>
  INTO <параметр> [,<параметр>….] <табличное выражение>
```

12) Количество <параметров> в <запросе выборки> должно совпадать с количеством элементов в <списке выборки>. В <списке выборки> можно использовать <параметрическое выражение>, содержащее, по крайней мере, один <литерал>.
13) Разрешена конструкция INTO для указания параметра, в который помещается результат выполнения хранимой процедуры (значение, передаваемое оператором RESULT процедурного языка):

```
EXECUTE INTO <параметр> <имя хранимой процедуры> (<аргумент>
 [,<аргумент> ….])
```

Примеры

```
select make from auto where 1900+year=1900+:year;

select make from auto where year>?;

select make from auto where year between :beg_year and :end_year;

select make from auto where year in ( ?, ?, ?);

select make from auto where make like 'FO'+?(char(2));

select make from auto where make=:make or make like ?;

select make, count(*) from auto group by make having
 count(*)>:year;
```

```
select distinct make,year into ?,? from auto where ?(char(4))=
 make;

select distinct 'MODEL:'+:make(char(10)), year into ?(char(10)),?
(int)
  from auto where ?(char(4))= make;

select substr( model,1,?(int)) into ? from auto where ?(char(4))=
 make;

update auto join person set auto.make=?(char(4))
 where auto.personid=person.personid
   and person.salary between :sum1 and :sum2;

select cast ? as smallint from auto; // не допустимо

select cast 0+? as smallint from auto;

select cast ? (int) as smallint from auto;

select distinct make,model from auto
 where make = case cylnders when : num_cyl(int) then
 upper(:if_make(char(10)))
                            else upper(:else_make(char(10))) end;

execute into :result saldo(?,?);

insert into auto( personid,make,year,model) values
 (?,:make,:year,:model);

insert into auto( personid,make,year,model) values
 (?,:make,:year,:model+cast(:p(int) as char(20)));

delete from auto where year<= ?;

select dt1,dt2 from table1 where :1 BETWEEN DT1 AND DT2;

select distinct 'MODEL:'+:make(char(10)),year into ?(char(10)),?
(int) from auto where ?(char(4))= make;

select abs(?-?);

select (case when (t1.currency = ?) then (t1.amount) else (?) end)
 p1 from db1_transaction t1;
```

Управление работой СУБД

Блокирование доступа к данным

Блокирование таблицы (LOCK TABLE)

Функция

Определение оператора блокирования таблицы.

Спецификация

[1] <блокирование таблицы>::=
LOCK TABLE [<имя схемы>.]<имя таблицы>
[IN <режим доступа> MODE] [WAIT | NOWAIT]
[2] <режим доступа>::= {SHARE | EXCLUSIVE}

Синтаксические правила

1) <Имя таблицы> должно ссылаться на базовую таблицу. Если <режим доступа> не указан, по умолчанию принимается EXCLUSIVE.
2) Если опция WAIT (NOWAIT) не указана, по умолчанию принимается WAIT.

```
lock table auto in share mode wait;
```

Общие правила

1) <Режим доступа> SHARE позволяет другим пользователям БД только читать блокированную таблицу.
2) <Режим доступа> EXCLUSIVE/SHARE блокирует доступ к таблице не только других пользователей БД, но и доступ того же пользователя по его другим соединениям с БД, т.к. соединения, открытые разными процессами или нитями от имени одного и того же пользователя (например, с помощью команды OPEN интерфейса нижнего уровня) конкурируют за доступ к БД как разные пользователи. Если несколько соединений открыты одной нитью, такой конкуренции между ними не происходит.
3) Опция WAIT (NOWAIT) задает реакцию пользователя на невозможность блокирования таблицы в данный момент: WAIT – ждать момента, когда таблицу можно будет блокировать, NOWAIT – отказаться от блокирования таблицы и вернуть соответствующий код завершения.
4) Блокировка таблицы отменяется SQL-операторами:
   - в режиме AUTOCOMMIT: UNLOCK, COMMIT;
   - в режиме PESSIMISTIC: COMMIT, ROLLBACK.

Разблокирование таблицы (UNLOCK TABLE)

Функция

Определение оператора разблокирования таблицы.

Спецификация

[1] <разблокирование таблицы>::=
UNLOCK TABLE [<имя схемы>.]<имя таблицы>

Общие правила

1) Разблокировать таблицу может только пользователь, выполнивший ее блокировку.

Блокирование доступа к БД (ALTER USER LOCK)

Функция

Определение оператора блокирования доступа к БД.

Спецификация

[1] <блокирование доступа к БД>::=
ALTER USER <имя пользователя> LOCK

Синтаксические правила

1) <Имя пользователя> – имя зарегистрированного в БД пользователя, которому блокируется доступ к БД.

Общие правила

1) Команда разрешена только создателю БД.
2) Блокируется доступ к той БД, к которой по текущему соединению была подана эта команда.
3) При попытке пользователя, которому заблокирован доступ к БД, установить соединение с этой БД, выдается код завершения 1026 («Неверный пароль»).
4) Отмена блокировки доступа выполняется с помощью команды ALTER USER ... UNLOCK.
5) Команда GRANT CONNECT TO... не снимает блокировки, установленные командой ALTER USER ...LOCK.

Пример

```
username SYSTEM/MANAGER8
create or replace user "test" identified by '12345678';
alter user "test" lock;
username "test"/'12345678'
1026: неверный пароль
```

Разблокирование доступа к БД (ALTER USER UNLOCK)

Функция

Определение оператора отмены блокирования доступа к БД.

Спецификация

[1] <разблокирование доступа к БД>::=
ALTER USER <имя пользователя> UNLOCK

Синтаксические правила

1) <Имя пользователя> – имя зарегистрированного в БД пользователя, для которого отменяется ранее установленная с помощью команды ALTER USER ... LOCK блокировка доступа к БД.

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

Общие правила

1) Команда разрешена только создателю БД.
2) Отмена блокирования доступа применяется к той БД, к которой по текущему соединению была подана эта команда.
3) Команда `ALTER USER ... UNLOCK` не отменяет блокировки, установленные командой `REVOKE CONNECT FROM ...`

Локальные транзакции

Транзакция – набор операций манипулирования данными в БД, завершающихся их подтверждением или отменой.

Локальные транзакции выполняются «локально», т.е. изменение данных происходит только в одной БД (на локальном или удаленном ЛИНТЕР-сервере).

Создание точки сохранения (SET SAVEPOINT)

Функция

Создание промежуточной точки сохранения в текущей локальной транзакции.

Спецификация

[1] <создание точки сохранения>::=
SET SAVEPOINT <имя точки сохранения>

Синтаксические правила

1) `<Имя точки сохранения>` должно быть уникальным для канала в течение транзакции.

```
SET SAVEPOINT "Итоговые суммы";
SET SAVEPOINT AFTER_SALE;
```

Пример

```
1) открываем канал в режиме EXCLUSIVE;
2) выполняем SQL-запросы, которые относятся к транзакции в целом
 (например, вносим в БД поступление новых заявок на товары);
3) организуем цикл по обработке  заявок на товары. Заявка может
 быть удовлетворена (если товаров достаточно), или отклонена – в
 противном случае;
4) в начале обработки каждой заявки устанавливаем контрольную
 точку:
set savepoint sp1;
5) выполняем SQL-запросы по обработке заявки;
6) по окончании обработки каждой заявки устанавливаем контрольную
 точку:
set savepoint sp2;
7) если заявка не может быть принята, делаем откат к контрольной
 точке, выставленной до начала обработки заявки, sp1:
```

```
rollback to savepoint sp1;
иначе принимаем заявку к исполнению – делаем commit до контрольной
 точки, выставленной по окончании обработки заявки, sp2:
commit to savepoint sp2;
8) продолжаем обработку заявок (шаг 4);
9) выполняем общий commit или rollback всей транзакции.
```

Сохранение изменений (COMMIT)

Функция

Сохранение (фиксация) в БД изменений, внесенных текущей локальной транзакцией.

Спецификация

[1] <сохранение изменений>::=
COMMIT [WORK] [TO SAVEPOINT [<имя точки сохранения>]] [RELEASE]

Синтаксические правила

1) COMMIT является сокращенной записью (синонимом) фразы COMMIT WORK.
2) Если <имя точки сохранения> задается, то оно должно ссылаться на существующую точку сохранения, созданную оператором SET SAVEPOINT.

```
commit to savepoint AFTER_SALE;
```

3) Фиксация изменений выполняется от
   - начала транзакции (в случае отсутствия предыдущих команд COMMIT TO SAVEPOINT в той же транзакции);
   - нового начала транзакции, установленного после выполнения в текущей транзакции оператора COMMIT TO SAVEPOINT до конца транзакции, если не задана конструкция TO SAVEPOINT;
   - указанной точки сохранения, если <имя точки сохранения> задано;
   - последней определенной точки сохранения, если <имя точки сохранения> в конструкции TO SAVEPOINT не задано (или до начала транзакции в случае отсутствия точек сохранения).
4) Опции TO SAVEPOINT и RELEASE несовместимы.

Общие правила

1) После выполнения COMMIT сделанные изменения становятся постоянными и необратимыми.
2) Установленные во время транзакции блокировки отменяются.
3) Если команда COMMIT задана без конструкции TO SAVEPOINT, то текущая транзакция неявно завершается и начинается новая транзакция.
4) Если команда COMMIT задана с конструкцией TO SAVEPOINT, то заданная (или последняя, если <имя точки сохранения> не задано) точка сохранения сохраняется, а все предшествующие ей удаляются.
5) Если при выполнении COMMIT произошла ошибка, выполняется автоматический ROLLBACK к началу транзакции.

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

6) Опция RELEASE заставляет автоматически закрывать все подчиненные курсоры соединения (по умолчанию при подтверждении транзакции все подчиненные курсоры соединения, по которому подана команда COMMIT, остаются в открытом состоянии).

Отмена изменений (ROLLBACK)

Функция

Отмена (откат) изменений, внесенных текущей локальной транзакцией.

Спецификация

[1] <отмена изменений>::=
ROLLBACK [WORK] [TO SAVEPOINT [<имя точки сохранения>]] [RELEASE]

Синтаксические правила

1) ROLLBACK является сокращенной записью (синонимом) фразы ROLLBACK WORK.

2) Если <имя точки сохранения> задается, то оно должно ссылаться на существующую точку сохранения, созданную оператором SET SAVEPOINT.

```
rollback to savepoint AFTER_SALE;
```

3) Если в конструкции TO SAVEPOINT <имя точки сохранения> не задано, то откат изменений выполняется до последней установленной точки сохранения или до начала транзакции в случае отсутствия точек сохранения.

4) Опции TO SAVEPOINT и RELEASE несовместимы.

Общие правила

1) После выполнения ROLLBACK сделанные изменения отменяются.

2) Установленные во время транзакции блокировки отменяются.

3) Если команда ROLLBACK задана без конструкции TO SAVEPOINT, то текущая транзакция завершается и начинается новая транзакция.

4) Если команда ROLLBACK задана с конструкцией TO SAVEPOINT, то указанная в ней (или последняя, если <имя точки сохранения> не задано) точка сохранения сохраняется и все последующие за ней точки сохранения удаляются.

5) Опция RELEASE заставляет автоматически закрывать все подчиненные курсоры соединения (по умолчанию при откате транзакции все подчиненные курсоры соединения, по которому подана команда ROLLBACK, остаются в открытом состоянии).

Управление функционированием

Включение проверки пароля при запуске ядра (STARTUP AUTHORIZATION)

Функция

Определение оператора включения режима проверки пароля при запуске ядра.

Спецификация

[1] <включение режима проверки пароля при запуске ядра>::=
ALTER DATABASE SET STARTUP AUTHORIZATION {ON | OFF}

Общие правила

1) После включения режима проверки пароля при запуске ядра будет требоваться задание учетных данных в ключе /U=<имя>/<пароль> строки запуска ядра СУБД ЛИНТЕР (см. документ «Запуск и останов СУБД ЛИНТЕР в среде ОС Windows» или «Запуск и останов СУБД ЛИНТЕР в среде ОС Linux»).

Примечание
Команда поддерживается со сборки 6.0.20.6.

2) Выполнить команду имеет право только создатель БД.

Включение режима сертифицированной БД (SET CERTIFIED)

Функция

Определение оператора включения режима сертифицированной БД.

Спецификация

[1] <включение режима сертифицированной БД>::=
ALTER DATABASE SET CERTIFIED

Общие правила

1) После включения режима сертифицированной БД отключение режима невозможно.

Примечание
Команда поддерживается со сборки 6.0.20.1.

2) Выполнить команду имеет право только создатель БД.

3) При повторном выполнении команды будет выдан код завершения 2197 («База данных уже сертифицированная»).

4) После включения режима будут выполняться следующие проверки:

- контроль целостности файлов БД: выполняется каждый раз при старте ядра. При несовпадении фактически вычисленных контрольных сумм и ожидаемых будет выдан код завершения 521 («Несовпадение контрольной суммы») и ядро будет остановлено;
- контроль длины пароля пользователя: длина пароля должна быть не менее 8 символов. При указании длины пароля менее 8 символов будет выдан код завершения 2194 («Слишком короткий пароль»), при попытке создания пользователя без пароля - будет выдан код завершения 2195 («Пароль не задан»).
- контроль максимального количества неудачных попыток ввода пароля: не более 4. При указании количества неудачных попыток ввода пароля более 4 будет выдан код завершения 2196 («Ошибочное значение лимита неудачных попыток

входа») и данный пользователь будет заблокирован до явной подачи команды разблокировки пользователя.

Включение/отключение режима суперпользователя для DBA (SET SUPERUSER)

Функция

Определение оператора включения/отключения режима суперпользователя для DBA.

Спецификация

[1] <включение/отключение режима суперпользователя для DBA>::=
ALTER DATABASE SET SUPERUSER {ON | OFF}

Общие правила

Примечание
Команда поддерживается со сборки 6.0.20.4.

1) Если подать команду ALTER DATABASE SET SUPERUSER ON (и не отменить ее действие командой ALTER DATABASE SET SUPERUSER OFF), то любому пользователю, имеющему уровень прав DBA, дополнительно разрешаются следующие действия (DML):
   - любые операции над любыми таблицами (включая SELECT, INSERT, DELETE, UPDATE);
   - запуск любых процедур (как EXECUTE, так и EXECUTE AS OWNER).

   В обоих случаях пользователю, имеющему уровень прав DBA, если только он не создатель БД, не разрешаются действия, вызывающие нарушение мандатного доступа.
2) По умолчанию режим суперпользователя для DBA отключен.
3) Выполнить команду имеет право только создатель БД.

Включение/отключение режима очистки оперативной памяти (SET MEMORY CLEAR)

Функция

Определение оператора включения/отключения режима очистки оперативной памяти.

Спецификация

[1] <включение/отключение режима очистки оперативной памяти>::=
ALTER DATABASE SET MEMORY CLEAR {ON | OFF}

Общие правила

Примечание
Команда поддерживается со сборки 6.0.20.6.

1) По умолчанию очистка оперативной памяти в режиме CERTIFIED включена.
2) Выполнить команду имеет право только создатель БД.
3) Команда ALTER DATABASE SET MEMORY CLEAR OFF выключает очистку оперативной памяти.
4) Описание работы очистки памяти приведено в разделе «Очистка оперативной и внешней памяти» документа «Администрирование комплекса средств защиты данных».

Управление максимальным квантом обработки запросов пользователя (SET QUANT TIMEOUT)

Функция

Определение оператора управления максимальным квантом обработки SQL-запросов конкретного пользователя.

Спецификация

[1] <управление максимальным квантом обработки запросов пользователя>::= SET {QUANT | QUANTUM} TIMEOUT {LIMIT <размер> | UNLIMITED} FOR <имя пользователя>

Синтаксические правила

1) Параметр <размер> задает максимальный размер кванта (от 1 до 65535 сек.) в течение которого будут непрерывно выполняться запросы заданного пользователя (см. документ «Архитектура СУБД»).
2) Если длительность непрерывного выполнения запроса пользователя превысит установленный <размер>, его обработка прекращается с выдачей кода завершения 183 («Выполнение запроса препятствует выполнению других запросов»).
3) Опция UNLIMITED устанавливает неограниченный максимальный квант обработки SQL-запросов указанного пользователя.

Примечание
При установке опции UNLIMITED для пользователя запросы прочих пользователей будут ожидать завершения выполнения запросов указанного пользователя.

Общие правила

1) Устанавливать максимальный размер кванта обработки запросов пользователя имеет право только создатель БД.
2) По умолчанию значение <размер> принимается равным 120 секунд.
3) Установленное ограничение сохраняется в БД, распространяется на все каналы, по которым подаются запросы пользователя, и начинает применяться ко всем SQL-запросам, поданным после команды SET QUANT TIMEOUT.

```
create user TEST identified by '12345678';
grant DBA to TEST;
username TEST/12345678
```


© Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
set quant timeout limit 5 for TEST;
Состояние выполнения: 1022 (нарушение привилегий)
username SYSTEM/MANAGER8
set quant timeout limit 5 for TEST;
select
  case when getbyte($$$s35,225) & 0x10 <> 0 then
    case when getbyte($$$s35,228) = 0 then 'UNLIMITED'
    else cast getbyte($$$s35,228) as char
    end
  else 'DEFAULT'
   end
  from $$$usr
 where $$$s31 > 0 and $$$s32 = 0 and $$$s34='SYSTEM';

| 5|
```

4) Если для пользователя установлен максимальный размер кванта, то изменить его для сессии данной командой можно только в меньшую сторону.

Управление максимальным квантом обработки запросов в сессии (SET SESSION QUANT TIMEOUT)

Функция

Определение оператора управления максимальным квантом обработки SQL-запросов в текущей сессии.

Спецификация

[1] <управление максимальным квантом обработки запросов в сессии>::=
SET SESSION {QUANT | QUANTUM} TIMEOUT {LIMIT <размер> | UNLIMITED}

Синтаксические правила

1) Параметр <размер> задает максимальным размер кванта непрерывной обработки запросов (от 1 до 65535 сек.) в текущей сессии (см. документ «Архитектура СУБД»).
2) Если длительность непрерывного выполнения запроса в сессии превысит установленный <размер>, его обработка прекращается с выдачей кода завершения 183 («Выполнение запроса препятствует выполнению других запросов»).
3) Опция UNLIMITED устанавливает неограниченное время обработки SQL-запросов в сессии.

Примечание
При установке опции UNLIMITED в сессии запросы в прочих сессиях будут ожидать завершения выполнения запросов указанной сессии.

Общие правила

1) Устанавливать максимальный размер кванта обработки запросов в сессии может пользователь, который инициировал сессию (установил соединение с БД).

2) Установленное ограничение действует только на время сессии (от начала открытия до закрытия канала (соединения) с БД и распространяется на все запросы, которые подаются по данному каналу, в том числе и на курсоры в хранимых процедурах.

3) Если для пользователя установлен максимальный размер кванта обработки его запросов, изменить его в сессии с помощью данной команды можно только в меньшую сторону.

Управление квантованием по времени (SET DATABASE QUANT)

Функция

Определение оператора управления квантом обработки SQL-запросов в БД.

Спецификация

[1] <управление квантом обработки запросов в БД>::=
SET DATABASE {QUANT | QUANTUM}
{<разрешение/запрет квантования> | <длительность кванта>}
[2] <разрешение/запрет квантования>::= TIME {ON | OFF}
[3] <длительность кванта>::= FOR TIME <миллисекунды>
[4] <миллисекунды>::= положительное целочисленное значение

Синтаксические правила

1) Опция ON разрешает, OFF – запрещает квантование по времени в БД.

2) <Миллисекунды> задаются в интервале [1, 65535].

3) Если после разрешения режима квантования по времени новая <длительность кванта> не задана, используется предыдущее значение, а если и его нет, то по умолчанию <длительность кванта> равна 1 миллисекунде.

Общие правила

1) Команда доступна только создателю БД.

2) Параметры команды начинают действовать после перезагрузки ядра СУБД.

3) По умолчанию режим квантования по времени запрещен.

4) Если задан режим квантования по времени, действующий в ядре СУБД режим квантования по умолчанию (квант определяется фиксированным временным счетчиком) отменяется.

Управление количеством квантов (SET DATABASE QUANTUM FOR)

Функция

Управление количеством выполняемых подряд квантов базы данных.

Спецификация

[1] <управление количеством квантов базы данных>::=
SET DATABASE QUANTUM FOR
{[переменная_1 = N1,] [переменная_2 = N2,] ... [переменная_M = NM]}

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

Синтаксические правила

1) В качестве переменных `переменная_1 ... переменная_M` могут выступать значения столбца "Переменная" из таблицы 4.

   Таблица 4. Переменные квантования

Переменная	Диапазон значений (значение по умолчанию)	Столбец в таблице $$$SYSINFO	Комментарий
insert	1-65536 (10)	INSERT_QUANT	Квант обработки при добавлении записей
delete	1-65536 (10)	DELETE_QUANT	Квант обработки при удалении записей
update	1-65536 (10)	UPDATE_QUANT	Квант обработки при модификации записей
scan	1-65536 (98)	SCAN_QUANT	Квант обработки при сканировании записей
index scan	1-65536 (10)	INDEX_SCAN_QUANT	Квант обработки при сканировании индекса
index page	1-65536 (8)	INDEX_PAGE_QUANT	Квант обработки при сканировании страниц индекса
index values	1-65536 (10)	INDEX_VALUES_QUANT	Квант обработки для найденных значений при сканировании индекса
sortpool scan	1-65536 (10)	SORTING_QUANT	Квант обработки при заполнении страниц сортировки
channel	1-65536 (2)	CHANNEL_QUANT	Количество выполняемых подряд квантов одного канала

2) Текущие переменные квантования можно увидеть, если извлечь данные из столбцов таблицы `$$$SYSINFO`, приведённых в таблице 4.

   Например, запрос:

```
select INSERT_QUANT,
       DELETE_QUANT,
       UPDATE_QUANT,
       SCAN_QUANT,
       INDEX_SCAN_QUANT,
       INDEX_PAGE_QUANT,
       INDEX_VALUES_QUANT,
       SORTING_QUANT,
       CHANNEL_QUANT
  from LINTER_SYSTEM_USER.$$$SYSINFO;
```

вернет следующие значения:

```
 INSERT_QUANT DELETE_QUANT UPDATE_QUANT SCAN_QUANT
 ------------ ------------ ------------ ----------
|     10     |     10     |     10     |    10
 INDEX_SCAN_QUANT INDEX_PAGE_QUANT INDEX_VALUES_QUANT
 ---------------- ---------------- ------------------
|      98        |      10        |        10
 SORTING_QUANT CHANNEL_QUANT
 ------------- -------------
|      2      |      10     |
INL : выдано строк : 1
```

Чтобы сделать все каналы БД «более квантуемыми», можно подать команду, уменьшив все значения переменных квантования вдвое:

```
set database quantum for insert 5, delete 5, update 5, sortpool
 scan 1, index scan 10, channel 5, index page 5, index values 5,
 scan 50;
```

Общие правила

1) Выполнение команды доступно только создателю БД.

Управление количеством выполняемых подряд квантов сессии (SET SESSION QUANTUM FOR)

Функция

Управление количеством выполняемых подряд квантов сессии.

Спецификация

[1] <управление количеством квантов сессии>::=
SET SESSION QUANTUM FOR
{[переменная_1 = N1,] [переменная_2 = N2,] ... [переменная_M = NM]}

Синтаксические правила

1) В качестве переменных `переменная_1 ... переменная_M` могут выступать значения столбца "Переменная" из таблицы 4.

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

Общие правила

1) Выполнение команды доступно любому пользователю и её действие распространяется на сессию. Т.е. при подаче такой команды по текущему соединению параметры квантования изменятся у всех родительских и у всех дочерних каналов.
2) Значения переменных квантования для сессии не могут превышать соответствующих значений для базы данных.

Управление протоколированием (SET LOG)

Функция

Определение оператора управления выводом информации в файл протокола СУБД ЛИНТЕР.

Спецификация

[1] <управление протоколированием>::=
SET LOG {OFF | ON | BRIEF | DEFAULT | FULL}

Общие правила

1) Опция `OFF` отключает (`ON` – включает) протоколирование SQL-запросов (т.е. отмену внесения протокольной информации в файл `LINTER.LOG`, расположенный в подкаталоге подключенной БД) пользователя БД, подавшего команду.
2) Опция `BRIEF` включает протоколирование SQL-запросов аналогично запуску ядра СУБД ЛИНТЕР с ключом `/LOGQUERY` (см. документ «Запуск и останов СУБД ЛИНТЕР в среде ОС Windows», раздел «Ключи протоколирования работы ядра СУБД» или «Запуск и останов СУБД ЛИНТЕР в среде ОС Linux», раздел «Ключи протоколирования работы ядра СУБД»).
3) Опция `DEFAULT` включает протоколирование аналогично запуску ядра СУБД ЛИНТЕР с ключом `/LOG` и без ключей `/LOGQUERY` и `/LOGALL`.
4) Опция `FULL` включает полное протоколирование аналогично запуску ядра СУБД ЛИНТЕР с ключом `/LOGALL`.

Управление выводом процедур (SET PROCEDURE TRACE)

Функция

Определение оператора управления выводом сообщений процедур в консоль ядра и в файл протокола СУБД ЛИНТЕР.

Спецификация

[1] <управление протоколированием>::=
SET PROCEDURE TRACE {ON | OFF}

Общие правила

1) Опция `OFF` отключает (`ON` – включает) вывод сообщений процедур в консоль ядра и в файл `linter.out`. Аналог ключа ядра `/PROCPRINT`, но не требующий перезапуска ядра.



Примечание
Команда поддерживается со сборки 6.0.17.96.

2) Опция ON разрешает хранимым процедурам выводить:

- на консоль ядра СУБД ЛИНТЕР и в файл протоколирования linter.out сообщения процедурной функции PRINT. Максимальный размер выводимого сообщения 980 символов;
- на консоль ядра СУБД ЛИНТЕР и в файл протоколирования linter.out информацию об исключениях. Если хранимая процедура оттранслирована с отладкой, то выдается номер ошибочной строки в процедуре.

Управление трассировкой выполняемых запросов (SET TRACE)

Функция

Разрешение/запрет трассировки выполняемых запросов.

Спецификация

[1] <управление трассировкой>::=
SET TRACE {<разрешение трассировки> | <запрет трассировки>}
[2] <разрешение трассировки>::=
ON [(<ключ>)] [FOR CONNECTION [MESSAGE <символьный литерал>]]
[3] <запрет трассировки>::= OFF [FOR CONNECTION]
[4] <ключ>::= <символьный литерал>

Синтаксические правила

1) Максимальная длина <символьного литерала> в опции MESSAGE равна 255 байтов.

2) Параметр <ключ> задает параметр трассировки. Его значения аналогичны значениям ключа /trace, задаваемого при старте ядра СУБД ЛИНТЕР (см. документ «Запуск и останов СУБД ЛИНТЕР в среде ОС Windows», раздел «Ключи протоколирования работы ядра СУБД» или «Запуск и останов СУБД ЛИНТЕР в среде ОС Linux», раздел «Ключи протоколирования работы ядра СУБД»). Значения ключа необходимо указывать без пробелов:

```
/TRACE=DECOMP=FULL;

/TRACE=DECOMP=DELAY[=<тики>]
```

Например,

```
set trace on ('/trace=decomp=full');
set trace on ('/trace=decomp=delay=50');
```

Примечание
Использование фразы /TRACE в тексте ключа является обязательным. В противном случае (например, при выполнении команды SET TRACE ON ('DECOMP=FULL');) на консоль ядра СУБД ЛИНТЕР и в файл linter.out будет выдано сообщение вида: «INFO: Incorrect argument 'DECOMP=FULL' for TRACE command. Ignored.».

3) Если параметр <ключ> не задан, команда по действию аналогична запуску ядра СУБД с ключом /trace=decomp

```
set trace on;
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

Общие правила

1) Трассировка выполняется в файл lintrace.log, создаваемый в каталоге БД ЛИНТЕР. Если файл lintrace.log уже существует, трассировочная информация добавляется в конец файла.
2) Если опция FOR CONNECTION не задана, то устанавливаемый режим трассировки распространяется на все соединения с БД, в противном случае устанавливаемый режим относится только к тому соединению, по которому подана команда.
3) Установленный режим начинает действовать с момента выполнения команды и распространяется только на вновь открываемые соединения (для текущих открытых соединений режим трассировки не меняется).
4) Опция MESSAGE определяет пользовательский текст (например, имя трассируемого клиентского приложения), который добавляется в трассировочный файл в начало каждого трассировочного сообщения.
5) В трассировочный файл <символьный литерал> опции MESSAGE записывается в кодировке ASCII (для ОС Windows) и KOI8-R (в ОС Linux, ЗОСРВ Нейтрино).
6) При повторном выполнении команды SET TRACE ON без отключения трассировки с новыми значениями параметров старые параметры трассировки будут заменены новыми, а сам файл lintracel.log переоткрыт не будет.

```
set trace on ('/trace=decomp=(full)');
select count(*) from auto where personid <= 21;

  Трассировочная информация:
C#3 QUERY:
  SELECT
    COUNT(*)
  FROM
    <TABLE "SYSTEM"."AUTO" AS T_0>
  WHERE
    T_0."PERSONID" <= 21;
C#3 DECOMP.C (Start_Cur_Dec): Now computing derived set #0.
C#3 OBRSTRAT.C (OBRSTRAT): Start set: TABLE("SYSTEM"."AUTO" AS
 T_0). Set
 included 1461 rows.
    List of predicates:
    Predicate [strategy #2(one index)]:
  T_0."PERSONID" <= 21
C#3 PROZA.C (PROZA): Strategy: #2(by indexes).
C#3 ODINKL.C (ODINKL): Snap_Bv: 21 rows.
    Predicate [strategy #2(one index)]:
  T_0."PERSONID" <= 21
C#3 DECOMP.C (Compute_Group): Get answer from start set. Start
 set:
 TABLE("SYSTEM"."AUTO" AS T_0).
    Rows count: 21.
C#3 DECOMP.C (End_Dekart): Derived set #0 is computed, Rows count:
 1.
C#3 FORMOTW.C (FORMOTW): Read: 5 blocks, write: 3 blocks.
```

```
 Additional statistics for read blocks:
  converter 1,index 2,data 1,work 0,sorting 0,blob 0,other 0.
C#3 FORMOTW.C (FORMOTW): Read logical: 22 blocks, write logical: 0
 blocks.
C#3 FORMOTW.C (FORMOTW): Journal read: 1 blocks, written: 3
 blocks.
C#3 FORMOTW.C (FORMOTW): Time of query execution: 00:00:00:00.10
```

Управление режимом ядра (SET TRUE COMMIT)

Функция

Определение оператора управления производительностью ядра СУБД ЛИНТЕР при обработке транзакций.

Спецификация

[1] <управление режимом ядра>::= SET TRUE COMMIT {ON | OFF}

Общие правила

1) По умолчанию СУБД использует опцию `ON`. При выполнении транзакции все произведенные СУБД изменения фиксируются в системном журнале. Если транзакция завершается оператором `COMMIT`, ядро СУБД переносит данные из журнала в файлы БД. Если в этот момент произошел отказ оборудования или программного обеспечения, данная транзакция (одна) будет потеряна. Это основной режим работы ядра (по умолчанию).
2) Если задана опция `TRUE COMMIT OFF`, работа ядра СУБД по обработке транзакции выполняется в следующем режиме:
   - при выполнении транзакции все произведенные СУБД изменения фиксируются в системном журнале;
   - если транзакция завершается оператором `COMMIT`, то пользователь (или пользовательское приложение) информируется о фиксировании измененных данных в БД (т.е. об успешном выполнении `COMMIT`), однако реальное обновление файлов БД происходит после накопления в системном журнале данных нескольких транзакций. Это повышает общую производительность СУБД, но чревато тем, что в случае отказа оборудования или программного обеспечения в момент изменения БД часть или, в худшем случае, все транзакции будут потеряны.

Примечание
Для повышения надежности сохранения данных опция OFF должна использоваться только временно.

Размер рабочей области (SET WORKSPACE)

Функция

Определение оператора задания размера рабочей области для конкретного пользователя СУБД.

Спецификация

[1] <размер рабочей области>::=
SET WORKSPACE {LIMIT <длина> | UNLIMITED} FOR <имя пользователя>

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

Синтаксические правила

1) <Длина> задается в мегабайтах.

```
set workspace limit 100 for "Гость";
```

Общие правила

1) Для выполнения оператора необходимы привилегии DBA.
2) Конструкция LIMIT ограничивает размер внешней памяти, выделяемой СУБД ЛИНТЕР пользователю в качестве его рабочей области, указанной в <длине>. При исчерпании этого ресурса выполнение пользовательского запроса прекращается с выдачей пользователю соответствующего кода завершения.
3) Конструкция UNLIMITED предоставляет пользователю все свободные в момент выполнения запроса ресурсы внешней памяти (вплоть до фактического исчерпания памяти на рабочих дисках СУБД).
4) По умолчанию размер рабочей области для пользователей установлен UNLIMITED.

Ограничение длины записи (SET RECORD SIZE LIMIT)

Функция

Определение оператора задания максимально допустимой длины записей таблиц в БД.

Спецификация

[1] <ограничение длины записи>::=
ALTER DATABASE SET RECORD SIZE LIMIT <длина>

Синтаксические правила

1) <Длина> задается в байтах.
2) Максимально допустимое значение <длины> записи 65535 байт. Реальная длина записи (в байтах) вычисляется путем округления заданного размера до кратного числу 4096 в большую сторону. Если значение <длины> больше 65535, то выдается предупреждающее сообщение, а размер записи берется равный 65535 (чтобы это значение уместилось в типе данных WORD).
3) По умолчанию СУБД устанавливает максимальную длину записи таблиц 65535 байт.

Общие правила

1) Для выполнения оператора необходимы привилегии DBA.
2) <Длина> задает размер буфера, который выделяется под распакованную запись. В этот буфер также должна всегда помещаться и упакованная запись, в связи с чем реально максимальный размер записи будет несколько меньше размера буфера (т.к. чем больше столбцов в таблице, тем больше требуется памяти для размещения дополнительной информации в упакованной записи).
3) Конструкция <ограничение длины записи> может изменять длину записи таблиц только в сторону увеличения по сравнению с текущей установленной длиной.
4) Чтобы новое ограничение на длину записи стало отслеживаться ядром СУБД ЛИНТЕР, необходимо после выполнения конструкции <ограничение длины записи> перезапустить ядро.

Примечание
Значение RECORD SIZE LIMIT можно получить с помощью запроса:

```
SELECT GETWORD($$$s14,130) from $$$sysrl WHERE ROWID=1;
```

Управление максимальным размером памяти каналов (SET CHANNEL MEMORY LIMIT)

Функция

Определение оператора управления максимальным размером памяти каналов ядра СУБД.

Спецификация

[1] <максимальный размер памяти каналов>::=
ALTER DATABASE SET CHANNEL MEMORY LIMIT <размер>

Синтаксические правила

1) <Размер> задается в байтах.
2) Допустимый диапазон значений <размера>: от 65536 до 16777216 (от 64 Кбайт до 16 Мбайт).

Общие правила

1) Для выполнения оператора необходимы привилегии DBA.
2) Заданная <размер> округляется в большую сторону с точностью 4 Кбайта.
3) По умолчанию максимальный размер памяти каналов, устанавливаемый утилитой gendb (см. «Создание и конфигурирование базы данных», пункт «Установка параметров конфигурирования»), равен 1 Мбайт.
4) В случае, когда по данной команде максимальный размер памяти каналов уменьшается, а один или несколько каналов в настоящий момент используют больший размер памяти, то будет возвращен код завершения 186 («Невозможно уменьшить размер памяти канала»). Следует либо закрыть каналы, использующие большой объем памяти, либо перезапустить ядро СУБД, после чего выполнить прерванные SQL-запросы.
5) Если затребованный размер памяти ядро СУБД выделить не сможет, будет выдан один из возможных кодов завершения (например, 812).
6) В остальных случаях, когда SQL-запрос, устанавливающий новый максимальный размер памяти каналов, возвращает успешный код завершения, перезапуск ядра СУБД не требуется.

Пример

```
alter database set channel memory limit 262144;
```

Ограничение времени работы всех неактивных сессий СУБД (ALTER DATABASE SET SESSION TIME LIMIT)

Функция

Ограничение времени работы неактивных сессий БД.

Спецификация

[1] <ограничение времени работы неактивных сессий БД>::=
ALTER DATABASE SET SESSION TIME LIMIT {<значение> | OFF}
[2] <значение>::= <беззнаковое целое>

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

Синтаксические правила

1) Команда действует на все сессии СУБД, открытые с момента выполнения команды и задаёт числовое <значение> времени в минутах (в интервале от 1 до 262144), через которое все каналы неактивных сессий будут автоматически закрыты.
2) Команда запоминается ядром СУБД и действует при каждом запуске ядра СУБД, предупреждая пользователя в случае установленного <значения> выдачей на консоль ядра сообщения: "Time limit for inactive sessions is <time_limit> min".

Примечание
«Неактивным» считается канал, находящийся в режиме IDLE, у которого не запущена сортировка и трансляция SQL, который не является каналом события на SELECT-запрос или процедурного события.

Время ограничение сессии инициализируется в момент отсылки последнего сообщения пользователю и <значение> отсчитывается от этого момента.

Проверка сессии на «неактивность» осуществляется вместе с проверкой «мертвых каналов». По умолчанию интервал проверки «мертвых каналов» составляет 30 секунд, этот интервал может изменяться пользователем (см. ключ /KILL в документе «Запуск и останов СУБД ЛИНТЕР в среде ОС Linux», раздел «Ключи управления функционированием» или «Запуск и останов СУБД ЛИНТЕР в среде ОС Windows», раздел «Ключи управления функционированием»). То есть, если мы выполнили команду:

```
alter database set session time limit 1;
```

то неактивная сессия, по которой подана команда, будет закрыта через промежуток времени в интервале от 1 мин. до 1 мин. 30 сек.

Общие правила

1) Для выполнения оператора необходимы привилегии DBA.

Пример

```
alter database set session time limit 5;
```

Ограничение времени работы отдельной неактивной сессии СУБД (SET SESSION TIME LIMIT)

Функция

Ограничивает время работы неактивной сессии.

Спецификация

[1] <ограничение времени работы неактивной сессии>::=
SET SESSION TIME LIMIT {<значение> | OFF}

Синтаксические правила

1) Команда действует на текущую сессию и задаёт числовое <значение> времени в минутах (в интервале от 1 до 262144), через которое данный канал неактивной сессии будет закрыт (см. описание команды «Ограничение времени работы всех неактивных сессий СУБД»).

Пример

```
set session time limit 15;
```

Приоритет пользователя (SET PRIORITY)

Функция

Определение оператора задания или изменения приоритета пользователя.

Спецификация

[1] <приоритет пользователя>::=
SET PRIORITY FOR <имя пользователя> <уровень приоритета>[, …]
[2] <уровень приоритета>::=
{BASE=<значение> | MAX=<значение> | RANGE=<значение>[, …]}

Синтаксические правила

1) <Значение> – целое положительное число в диапазоне от 0 до 255 (см. документ «Архитектура СУБД», раздел «Приоритетная обработка»).

Общие правила

1) Параметр MAX задает максимально возможный приоритет пользователя (верхнюю границу приоритетов).
2) Параметр BASE задает значение приоритета по умолчанию.
3) Параметр RANGE задает минимально возможный приоритет пользователя (нижнюю границу приоритетов).
4) Если параметры MAX, BASE, RANGE явно не заданы, то принимается, что MAX=BASE (превышения нет), RANGE=BASE (снижение до 0), т.е. если для пользователя были ранее заданы все три значения приоритетов, а при новой установке приоритета указывается только BASE, то неявным образом изменяются и MAX, и RANGE.
5) Значения приоритетов в диапазоне от 0 до 99 задают возможность динамического изменения приоритетов пользователем. За полный цикл квантования каждый канал должен получить свою порцию квантования. Место канала в очереди определяется временем его активизации, а не его приоритетом. Кроме квантования запросов происходит динамическое изменение приоритета. Границы изменений определяются диапазоном изменения приоритета пользователя и его максимальным приоритетом.
6) Значения приоритетов в диапазоне от 100 до 199 задают квантование запросов пользователя на одном уровне (циклическое планирование).
7) Значения приоритетов в диапазоне от 200 до 249 отменяют квантование запросов пользователя (предназначены для системного процесса и системных запросов).
8) Значения приоритетов в диапазоне от 250 до 255 – резервная группа.
9) Для системного канала приоритет равен 249 и является неизменным.
10) При открытии канала (открытие соединения или курсора) значение приоритета устанавливается в 0. В дальнейшем его значение может меняться.
11) Приоритет пользователя – приоритет, задаваемый администратором БД пользователю БД. Используется для назначения приоритетов каналу при его открытии и дальнейшем запросу при его подаче (в случае, если не указан приоритет при подаче запроса). Администратор БД не может назначить приоритет выше собственного максимального приоритета, кроме создателя БД.
12) Диапазон изменения приоритета пользователя указывает максимальную величину, на которую может быть динамически снижен приоритет канала в процессе работы.

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

13) Максимальный приоритет пользователя – это максимальный приоритет, с которым могут выполняться запросы пользователя. Ни динамически, ни принудительно невозможно превысить это значение.

Управление логированием BLOB-данных (SET SESSION BLOB LOG)

Функция

Запрещает/разрешает записывать в системный журнал BLOB-данные.

Спецификация

[1] <управление логированием BLOB-данных>::=
SET SESSION BLOB LOG {ON|OFF}

Синтаксические правила

1) Опция ON разрешает (значение по умолчанию), OFF – запрещает записывать в системный журнал BLOB-данные при выполнении транзакции в режиме EXCLUSIVE (PESSIMISTIC). В режиме AUTOCOMMIT данная конструкция игнорируется (полное логирование в системный журнал BLOB-данных всегда отключено).

Общие правила

1) Команда действует на всё соединение (то есть и на все каналы открытые в данном соединении).
2) Команда не предназначена для работы с системой горячего резервирования.
3) Если задан режим LOG ON, то все изменения BLOB-данных таблицы записываются в системный журнал и при фиксации транзакции (COMMIT) переносятся в соответствующие BLOB-файлы этой таблицы.
4) Если задан режим LOG ON, то для обеспечения возможности «теплого рестарта» СУБД обработка BLOB-данных выполняется следующим образом:
   - добавляемые во время транзакции BLOB-данные записываются одновременно в BLOB-файлы таблицы и в системный журнал;
   - удаляемые BLOB-данные записываются в системный журнал только при подтверждении транзакции (то есть после подачи команды COMMIT, когда они реально удаляются из BLOB-файлов).
5) Если задан режим LOG OFF, то все изменения BLOB-данных (то есть удаление/добавление страниц BLOB-файла) выполняются непосредственно в самих BLOB-файлах таблицы, а в системном журнале делается только отметка об удаленных (или добавленных) в BLOB-файле страницах. При фиксации транзакции (COMMIT) добавленные страницы принудительно сбрасываются в БД, а страницы, помеченные как «удаленные», реально удаляются из BLOB-файлов без сохранения BLOB-данных в системном журнале.
6) При откате транзакции (ROLLBACK) «удаленные» страницы остаются в прежних BLOB-файлах, а добавленные очищаются тоже без сохранения BLOB-данных в журнале.
7) Использование режима LOG OFF повышает производительность СУБД при выполнении транзакций с BLOB-данными, но при этом «теплый рестарт» не гарантирует откат незаконченного подтверждения (COMMIT) транзакции, содержащей удаление BLOB-данных (удаление будет продолжено). Физическая непротиворечивость БД гарантируется.

Режим выполнения хранимых процедур (SET SESSION PROCEDURE AS DEFINER)

Функция

Задает режим выполнения хранимых процедур.

Спецификация

[1] <режим выполнения процедур>::=
SET SESSION PROCEDURE EXECUTE BY DEFAULT AS
<от имени владельца> | <от имени текущего пользователя>
[2] <от имени владельца>::= DEFINER
[3] <от имени текущего пользователя>::= CURRENT_USER

Общие правила

1) Команда действует только на канал, по которому она подана.
2) Опция `<от имени владельца>` устанавливает режим выполнения всех внутренних SQL-запросов процедуры, созданной без опции `AUTHID DEFINER`, от имени пользователя-владельца процедуры.
3) Опция `<от имени текущего пользователя>` устанавливает режим выполнения всех внутренних SQL-запросов процедуры, созданной без опции `AUTHID CURRENT_USER`, от имени текущего пользователя соединения.

Пример

```
username SYSTEM/MANAGER8
drop user USER1 cascade;
create user "USER1" identified by '12345678';
grant DBA to USER1;

create or replace table TEST(i int, ch char(66));
insert into test values (1,'test record');

set session procedure execute by default as definer;

create or replace procedure TEST_PROC(IN id INTEGER) result
 integer
declare
  var i int;//
code
  execute "select count(*) from test;" into i;//
  return i; //
end;
grant execute AS OWNER on SYSTEM.TEST_PROC to USER1;

username USER1/12345678
execute SYSTEM.TEST_PROC();
Return value = 1
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

Уровни мандатного доступа к записям по умолчанию

Функция

Задает RAL/WAL-уровни мандатного доступа по умолчанию для всех операций INSERT/UPDATE в текущем соединении с ядром СУБД ЛИНТЕР.

Примечание
Поддерживается только в СУБД ЛИНТЕР БАСТИОН.

Спецификация

[1] <уровни по умолчанию мандатного доступа>::=
SET SESSION DEFAULT SECURITY #[<группа>]#[<RAL>]#[<WAL>]

Общие правила

1) Указанные метки доступа применяются ко всем операциям INSERT/UPDATE на всех ранее открытых дочерних каналах и на всех дочерних каналах, которые будут открыты позднее (т.е. указывать метки доступа в операциях INSERT/UPDATE станет обязательным только в том случае, если они отличаются от заданных в данной команде).
2) Устанавливаемый уровень чтения не может превышать уровень чтения, уже установленный в канале (изначально в канале он берётся из описателя пользователя).
3) Устанавливаемый уровень записи не может быть меньше уровня записи уже установленного в канале (изначально он также берётся из описателя пользователя).
4) Команда устанавливает также и значения меток доступа для INSERT- или UPDATE-запросов (метка генерируется по задаваемой метке доступа для работы с объектами БД; алгоритм генерации тот же, что и при открытии канала).
5) При попытке сменить группу будет выдан код завершения 1070 «Нарушение мандатного доступа» (т.е. нельзя работать в чужой группе).
6) В отличие от команды set session security при попытке сменить группу код завершения 1070 «Нарушение мандатного доступа» (т.е. нельзя работать в чужой группе) выдаваться не будет (т.е. можно вносить в БД данные и для другой группы).

Пример

```
create level "NS" = 1;
create level "DSP" = 2;
create level "C" = 3;
create level "CC" = 4;

create user "TEST" identified by '12345678';
grant DBA to "TEST";
username TEST/12345678

create or replace table tst (i int, j int);
insert into tst values(1,1);
username SYSTEM/MANAGER8
alter user TEST level ("DSP","DSP");
username TEST/12345678
```

```
insert into tst values(2,2);
select security(*,'R'), security(*,'W') from tst;

|нарушение доступа|
|2  |2  |

set session default security ##C#C;
insert into tst values(3,3);
set session default security ##CC#CC;
insert into tst values(4,4);
select security(*,'R'),security(*,'W') from tst;

|нарушение доступа|
|2  |2  |
|нарушение доступа|
|нарушение доступа|
|нарушение доступа|

username SYSTEM/MANAGER8
alter user TEST level ("CC","CC");
username TEST/12345678
select security(*,'R'),security(*,'W') from tst;

|4  |4  |
|2  |2  |
|3  |3  |
|4  |4  |

update tst set i=10,j=10 where i=1;
select security(*,'R'),security(*,'W') from tst;

|4  |4  |
|2  |2  |
|3  |3  |
|4  |4  |

set session default security ##NS#NS;

|нарушение доступа|
```

Уровни мандатного доступа к объектам БД (SET SESSION SECURITY)

Функция

Задает RAL/WAL-уровни мандатного доступа для объектов БД (таблиц, представлений и т.п.) в текущем соединении с ядром СУБД ЛИНТЕР.

Примечание
Поддерживается только в СУБД ЛИНТЕР БАСТИОН.

Спецификация

[1] <уровни мандатного доступа к объектам БД>::=
SET SESSION SECURITY #[<группа>]#[<RAL>]#[<WAL>]

Общие правила

1) В отличие от команды `set session default security`, данная команда влияет не на метки доступа в INSERT/UPDATE-запросах, а устанавливает в канале метку доступа для работы с объектами БД (таблицами, представлениями и т.п.). По умолчанию эта метка берётся из описателя пользователя, выполняющего данную команду.
2) Команда распространяется на все открытые дочерние каналы (которые уже открыты ранее или будут открыты впоследствии).
3) Устанавливаемые командой уровни чтения и записи должны быть не меньше текущего уровня записи – это условие должно выполняться для каждого вызова команды (в случае, если для одной и той же сессии команда вызывается многократно), т.е. понижать уровни (даже до их первоначальных значений) запрещается.
4) Если ранее в канале по команде `set session default security` были установлены метки доступа для INSERT/UPDATE-операций, то данная команда отменяет их и устанавливает новые значения этих меток по умолчанию в соответствии с метками доступа, задаваемыми для объектов БД в команде `session default security`.
5) Если в команде не задана группа или уровень доступа, то используется либо значение по умолчанию ядра СУБД ЛИНТЕР, либо значения, указанные в ранее выполненной команде `set session default security` либо `set session security`.

```
CREATE LEVEL "НЕСЕКРЕТНО"   = 1;
CREATE LEVEL "ДСП"          = 2;
CREATE LEVEL "СЕКРЕТНО"     = 3;
CREATE LEVEL "СОВ.СЕКРЕТНО" = 4;

create GROUP GROUP1;
create GROUP GROUP2;

grant DBA to A identified by '12345678';

alter user A LEVEL("ДСП", "ДСП");
alter user A group GROUP2;
username A/12345678

set session default security ##"СЕКРЕТНО"#"СЕКРЕТНО";

set session default security #GROUP1#"СЕКРЕТНО"#"СЕКРЕТНО";

set session security ##"НЕСЕКРЕТНО"#"СОВ.СЕКРЕТНО";
```

```
set session security ###;

set session security ###"СОВ.СЕКРЕТНО";

!6 error 1070 (we can not change group):
set session security #GROUP1#"НЕСЕКРЕТНО"#"СОВ.СЕКРЕТНО";
```

6) При попытке сменить группу будет выдан код завершения 1070 «Нарушение мандатного доступа» (т.е. нельзя работать в чужой группе).

Режим проверки ограничения ссылочной целостности (SET CONSTRAINTS)

Функция

Определение режима проверки ограничения целостности (немедленная проверка или отложенная).

Спецификация

[1] <режим проверки ограничения ссылочной целостности> ::=
SET CONSTRAINTS ALL {DEFERRED | IMMEDIATE}

Синтаксические правила

1) Модификатор IMMEDIATE устанавливает режим немедленной проверки ограничений целостности. Используется по умолчанию.
2) Модификатор DEFERRED устанавливает режим отложенной проверки ограничений целостности.

Общие правила

1) Модификатор DEFERRED относится только к проверке ограничений целостности. Обработка других ошибок обработки данных выполняется немедленно.
2) DEFERRED действует только для ограничений целостности с признаком NO ACTION (для режимов CASCADE, SET DEFAULT, SET NULL не действует).
3) Режим SET CONSTRAINTS ALL DEFERRED:

   применяется к проверке ограничений целостности CHECK и FOREIGN KEY и не применяется для проверки ограничений целостности:
   - PRIMARY KEY (на NOT NULL-значение и отсутствие дубликатов);
   - UNIQUE (на отсутствие дубликатов);
   - NOT NULL-значений.
4) Модификатор DEFERRED устанавливает режим отложенной проверки ограничений целостности – до окончания текущей транзакции (только по COMMIT) или до установки режима IMMEDIATE.
5) Режим IMMEDIATE может быть установлен в любой момент обработки данных.
6) Если задан режим DEFERRED, и транзакция завершается пользователем по COMMIT, но в процессе выполнения COMMIT выявляется нарушение целостности, такая транзакция откатывается (автоматически выполняется ROLLBACK).
7) Подтверждение или откат транзакции (COMMIT/ROLLBACK) устанавливает режим IMMEDIATE.

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

Пример

```
Для составных индексов (sql-скрипт).
Общая для всех часть:
drop table sec;
drop table prim;
create table prim( i1 int, ch char(1), j1 int, primary
 key(i1,j1) );

create table sec( i2 int, ch char(3), j2 int );
alter table sec add FOREIGN KEY (i2,j2) references prim( i1, j1 );
exclusive;  // это команда inl-интерфейса

SET CONSTRAINTS ALL DEFERRED;
1) добавление значения в таблицу с ссылочным ключом, когда запись
 отсутствует в
 основной таблице:
exclusive; // это команда inl-интерфейса
insert into sec(i2,ch,j2) values(1, 'aaa',1);
insert into prim(i1,ch,j1) values(1,'b',1);

2) изменение значения в таблице с ссылочным ключом, когда новое
 значение отсутствует в основной таблице:
insert into prim(i1,ch,j1) values(1,'b',1);
insert into sec(i2,ch,j2) values(1,'aaa',1);

update sec set j2 = 2, i2 = 3;
update prim set j1 = 2, i1 = 3;

3) удаление значения в основной таблице, когда значение
 присутствует в таблице с ссылочным ключом:
insert into prim(i1,ch,j1) values(1,'b',1);
insert into sec(i2,ch,j2) values(1,'aaa',1);

delete from prim where i1=1 and j1=1;
delete from sec where i2=1 and j2=1;

4) изменение значения в основной таблице, когда значение
 присутствует в таблице с ссылочным ключом:
insert into prim(i1,ch,j1) values(1,'b',1);
insert into sec(i2,ch,j2) values(1,'aaa',1);

update prim set i1=2, j1=3;
update sec set i2=2, j2=3;

Общая для всех часть:
commit;
```

```
select * from prim;
select * from sec;
```

Архивирование БД

Начать оперативное архивирование (BACKUP DATABASE)

Функция

Определение оператора запуска процесса архивирования БД в оперативном режиме (не прекращая работу ядра СУБД).

Спецификация

[1] <начать архивирование>::=
BACKUP DATABASE [[START | STOP] INCREMENT]
[DEVICE <имя устройства>]
[FILE <спецификация файла> [REWRITE]]
[COMMENT <комментарий>]
[PASSWORD <пароль>]
[VOLUMES <размер тома> [K | M]]
[ASYNC]

[2] <размер тома>::= <целочисленный литерал>

Синтаксические правила

1) Опция DATABASE задает полное сохранение БД без возможности нарастающего архивирования.
2) При отсутствии конструкций [[START |STOP] INCREMENT] будет выполнено полное сохранение БД без возможности нарастающего архивирования.
3) <Спецификация файла> задает полную спецификацию (устройство, путь к каталогу и имя) архивного файла. Если тип файла не указан, по умолчанию используется .lhb.
4) Если опция FILE <спецификация файла> не задана, по умолчанию архивный файл db.lhb создается в каталоге архивируемой БД.
5) Конструкция DATABASE START INCREMENT задает полное сохранение БД с возможностью нарастающего архивирования.

   Полное сохранение БД с последующим инкрементным архивированием; файл архива будет разбит на тома размером по 100 Кбайт:

```
BACKUP DATABASE START INCREMENT FILE 'inc.lhb' VOLUMES 100k;
```

6) Конструкция DATABASE INCREMENT задает сохранение всех изменений в БД со времени последнего нарастающего сохранения (от последней контрольной точки, созданной, например, с помощью DATABASE START INCREMENT).

   Выполнение нарастающего архивирования (запись из системного журнала БД в архив накопленных после контрольной точки изменений):

```
BACKUP DATABASE INCREMENT FILE 'inc.lhb' VOLUMES 100k;
```

7) Конструкция DATABASE STOP INCREMENT задает очистку (удаление) контрольной точки из БД. Информация о контрольной точке берется из файла архива, созданного путем команды DATABASE START INCREMENT. После этой команды дальнейшее

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

инкрементное наращивание архива станет невозможным. В результате сохранение данных не происходит.

Прекращение нарастающего архивирования:

```
BACKUP DATABASE STOP INCREMENT FILE 'inc.lhb';
```

Конструкция DATABASE STOP INCREMENT должна ссылаться на <спецификацию файла>, указанную в конструкции DATABASE START INCREMENT.

8) Опция DEVICE задает 4-х символьное логическое имя устройства, на котором должен создаваться файл архива. Устройство должно быть определено в системной таблице $$$DEVICE. Если имя устройства не задано, по умолчанию используется SY00. Если SY00 не определена, архив создается в каталоге запуска ядра СУБД ЛИНТЕР.
9) Опция REWRITE разрешает удалять существующий архивный файл и создавать новый с тем же именем.
10) <Комментарий> задает текст комментария к архивному файлу.
11) <Пароль> задает пароль архивного файла.
12) Конструкция VOLUMES <размер тома> [K|M] разрешает разбивать архивный файл на отдельные тома указанного <размера тома>: К – в Кбайтах, М – Мбайтах. Если конструкция VOLUMES <размер тома> [K|M] не указана будет создан единый архив.

```
BACKUP DATABASE START INCREMENT FILE 'inc.lhb' VOLUMES 100k;
```

13) Опция ASYNC задает асинхронное выполнение процесса архивирования.

```
BACKUP DATABASE ASYNC FILE 'File_name.lhb' REWRITE COMMENT
 'Increment archive of DB SALE';
```

Общие правила

Примечания

1. По возможности использовать для архивирования БД утилиту lhb, а не SQL-запрос BACKUP DATABASE.
2. Не использовать для архивирования БД SQL-запрос BACKUP DATABASE, если в БД есть фразовые индексы.

1) Общие правила архивирования изложены в документе «Архивирование и восстановление базы данных», раздел «Оперативное архивирование».
2) Если задана опция ASYNC, пользователю сразу будет возвращен код завершения, указывающий на результат выполнения SQL-запроса (успешный или неуспешный запуск процесса архивирования). В дальнейшем информацию о текущем состоянии процесса архивирования можно будет получать из системной таблицы $$$INKERNBACK (документ «Архивирование и восстановление базы данных», раздел «Механизм асинхронного архивирования»).

Остановить оперативное архивирование (BACKUP STOP)

Функция

Определение оператора прерывания асинхронного архивирования БД в оперативном режиме.

Спецификация

[1] <остановить архивирование>::=
BACKUP STOP {ALL [SINCE <дата-время литерал>]
[UNTIL <дата-время литерал>]}
|<номер процесса>
[2] <номер процесса>::= <целочисленный литерал>

Синтаксические правила

1) Опция `ALL` задает останов всех процессов архивирования (как тех, которые запущены пользователем, подавшим эту команду, так и других, запущенных иными пользователями БД).
2) Конструкция `SINCE <дата-время литерал>` задает останов процессов архивирования, активизированных после указанной <даты-времени>.
3) Конструкция `UNTIL <дата-время литерал>` задает останов процессов архивирования, активизированных до указанной <даты-времени>.
4) `<Номер процесса>` – системный идентификатор процесса архивирования. `<Номер процесса>` возвращается в поле `ROWID` блока управления запросом `CBL` (см. документ «Интерфейс нижнего уровня», раздел «Блок управления запросом CBL») или в переменной `ROWIDPCI_` (см. документ «Встроенный SQL», раздел «Псевдопеременные встроенного языка») при успешном запуске процесса архивирования.

Общие правила

1) Прервать можно только асинхронное архивирование.
2) Прервать процесс архивирования может либо пользователь, который его инициировал, либо пользователь с правами DBA (права DBA позволяют прерывать любые активные процессы архивирования, в том числе и процессы других пользователей).

Пример

Останов всех запущенных в указанном диапазоне времени асинхронных процессов архивирования:

```
BACKUP STOP ALL SINCE '09.07.2002:12:00' UNTIL '09.07.2002:14:00';
```

Репликация данных

Объявление сервера репликации (CREATE NODE/ SERVER)

Функция

Определение оператора добавления сервера репликации в системную таблицу серверов репликации.

Спецификация

[1] <объявление сервера репликации>::=
CREATE [IF NOT EXISTS | OR REPLACE] {NODE | SERVER} <имя сервера>
[LOCAL | REMOTE]
[2] <имя сервера>::= <идентификатор>

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

Синтаксические правила

1) <Имя сервера> – идентификатор длиной не более 8 символов.
2) Опция OR REPLACE заставляет удалять существующее в БД описание сервера и создавать его под тем же именем, но с другими параметрами.
3) Опция IF NOT EXISTS отменяет выполнение оператора, если указанный сервер уже существует в БД.
4) Одновременное использование опций IF NOT EXISTS и OR REPLACE запрещено.

Общие правила

1) В БД-источнике репликации должна быть системная таблица SERVERS (см. документ «Репликация данных», раздел «Конфигурирование БД»).
2) <Имя сервера> должно совпадать с одним из имен ЛИНТЕР-серверов из файла сетевой конфигурации nodetab для данного ЛИНТЕР SQL-сервера.
3) Для создания сервера репликации необходимо иметь права администратора БД-источника.

Удаление сервера репликации (DROP NODE/SERVER)

Функция

Определение оператора удаления сервера репликации из системной таблицы серверов репликации.

Спецификация

[1] <удаление сервера репликации>::=
DROP {NODE | SERVER} <имя сервера> [CASCADE]

Синтаксические правила

1) <Имя сервера> – идентификатор длиной не более 8 символов.

Общие правила

1) <Имя сервера> должно присутствовать в системной таблице SERVERS (см. документ «Репликация данных», раздел «Конфигурирование БД»).
2) Для удаления сервера репликации необходимо иметь права администратора БД-источника.
3) Если задан режим CASCADE, то также удаляются все репликационные правила, которые ссылаются на удаляемый сервер.
4) Если режим CASCADE не задан, то удаление сервера, на который есть ссылки в правилах репликации, выполняться не будет.

Создание правила репликации (CREATE REPLICATION RULE)

Функция

Определение оператора репликации данных.

Спецификация

[1] <создание правила репликации>::=
CREATE [IF NOT EXISTS | OR REPLACE] REPLICATION RULE <имя правила>
FOR [<имя схемы>.]<имя таблицы-источника> [TO [<имя схемы>.]<имя таблицы-приемника>]
ON {NODE | SERVER} <имя узла>
[USER <имя пользователя>] [PASSWORD <пароль>]
[ENABLE | DISABLE]
[SYNC | ASYNC]
[PRIORITY {SECOND | FIRST | NEW | OLD | WEIGHT | DEFAULT}]
[CALCULATE {NONE | MAX | MIN | AVG | DIFFERENCE | DEFAULT}]
[COLUMN (<имя столбца>[, …])
[PRIORITY {SECOND | FIRST | NEW | OLD | WEIGHT | DEFAULT}]
[CALCULATE {NONE | MAX | MIN | AVG | DIFFERENCE | DEFAULT}]
[...]]

Синтаксические правила

1) <Имя правила> задается идентификатором длиной не более 66 символов.
2) Опция OR REPLACE заставляет удалять существующее в БД правило репликации и создавать его под тем же именем, но с другими параметрами.
3) Опция IF NOT EXISTS отменяет выполнение оператора, если указанное правило репликации уже существует в БД.
4) Одновременное использование опций IF NOT EXISTS и OR REPLACE запрещено.
5) <Пароль> – пароль пользователя БД-приемника.
6) В качестве <имени таблицы-источника> и <имени таблицы-приемника> должны указываться имена базовых таблиц. Использовать имена представлений и системных таблиц не допускается.
7) Правило репликации создается от имени пользователя, выполняющего создание правила.
8) <Имя пользователя> задает имя пользователя на удаленном узле. Этот пользователь обязательно должен быть пользователем удаленной БД в момент создания правила.
9) ENABLE – правило доступно, DISABLE – правило недоступно для использования.
10) SYNC – режим синхронной, ASYNC – асинхронной репликации. При асинхронном режиме тиражирование данных выполняется по мере возможности.
11) По умолчанию используются значения ENABLE, ASYNC.
12) Конструкции PRIORITY и CALCULATE задают правила разрешения конфликтов при тиражировании данных. Правила задаются для всей таблицы в целом.
13) Если используется конструкция COLUMN (<имя столбца>[, …]), то для указанных столбцов могут быть установлены индивидуальные правила разрешения конфликтов (см. документ «Репликация данных», пункт «Создание правила репликации»).

Общие правила

1) Структуры локальной и удаленной таблиц должны быть идентичны.
2) В БД-источнике репликации должны находиться системные таблицы SERVERS, $$$REPL и $$$EXTREEPL.

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

3) Репликация таблиц, содержащих данные типа EXTFILE, не допускается.

4) Локальная таблица (и, соответственно, удаленная) должна обязательно иметь одно или многостолбцовый индекс с атрибутом PRIMARY KEY.

5) Создавать правило репликации имеет право владелец тиражируемой таблицы или пользователь с привилегиями SELECT, DELETE, INSERT, UPDATE на эту таблицу.

Изменение правила репликации (ALTER REPLICATION RULE)

Функция

Определение оператора изменения правила репликации.

Спецификация

[1] <изменение правила репликации> : : =
ALTER REPLICATION RULE <имя правила>
[PASSWORD <пароль>]
[ENABLE | DISABLE]
[PRIORITY {SECOND | FIRST | NEW | OLD | WEIGHT | DEFAULT}]
[CALCULATE {NONE | MAX | MIN | AVG | DIFFERENCE | DEFAULT}]
[COLUMN (<имя столбца>[, …])
[PRIORITY {SECOND | FIRST | NEW | OLD | WEIGHT | DEFAULT}]
[CALCULATE {NONE | MAX | MIN | AVG | DIFFERENCE | DEFAULT}]
[...]]

Синтаксические правила

1) <Имя правила> – идентификатор длиной не более 66 символов, уникальный в БД.

2) <Пароль> – пароль пользователя БД-приемника.

3) По умолчанию используются значения ENABLE и ASYNC.

Общие правила

1) <Имя правила> должно ссылаться на правило, которое уже присутствует в системной таблице $$$REPL.

2) Изменять правило репликации имеет право владелец тиражируемой таблицы или пользователь с привилегиями SELECT, DELETE, INSERT, UPDATE на эту таблицу.

3) Описание конструкций PRIORITY и CALCULATE см. в пункте «Создание правила репликации».

Удаление правила репликации (DROP REPLICATION RULE)

Функция

Определение оператора удаления существующего правила репликации.

Спецификация

[1] <удаление правила репликации> : : =
DROP REPLICATION RULE <имя правила>

Общие правила

1) <Имя правила> должно ссылаться на правило, которое уже присутствует в системной таблице $$$REPL.
2) Удалить правило репликации имеет право владелец тиражируемой таблицы или пользователь с привилегиями SELECT, DELETE, INSERT, UPDATE на эту таблицу.

Синхронизация таблиц репликации (SYNCHRONIZE REPLICATION RULE)

Функция

Определение оператора синхронизации таблицы-источника и таблицы-приемника. Данная конструкция используется обычно после того, как создано новое правило репликации. Перед тем, как начать его использовать, необходимо обеспечить идентичность участвующих в репликации таблицы-источника и таблицы-приемника.

Спецификация

[1] <синхронизация таблиц репликации> **::=**
SYNCHRONIZE REPLICATION RULE <имя правила>

Общие правила

1) <Имя правила> должно ссылаться на правило, которое уже присутствует в системной таблице $$$REPL.
2) Синхронизировать таблицы имеет право владелец тиражируемой таблицы или пользователь с привилегиями SELECT, DELETE, INSERT, UPDATE на эту таблицу.

© Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

Расширение SQL

Геометрические типы данных

Описание геометрических типов данных приведено в документе «Геометрические типы данных».

Полнотекстовый поиск данных

Описание полнотекстового поиска данных приведено в документе «Полнотекстовый поиск в базе данных».

Процедурный язык

Описание процедурного языка приведено в документе «Процедурный язык».

Расширенные средства защиты данных

Описание расширенных средств защиты данных приведено в документе «Администрирование комплекса средств защиты данных».

SQL-интерфейсы

Нижний уровень

Описание интерфейса нижнего уровня приведено в документе «Интерфейс нижнего уровня».

Прикладной уровень

Описание интерфейса прикладного уровня приведено в документе «Прикладной интерфейс».

Описание объектной библиотеки прикладного уровня приведено в документе «Объектная библиотека прикладного интерфейса».

Встроенный SQL

Описание встроенного SQL приведено в документе «Встроенный SQL».

ODBC

Описание ODBC-драйвера приведено в документе «ODBC-драйвер».

JDBC

Описание JDBC-драйвера приведено в документе «JDBC-драйвер».

Perl

Описание Perl-интерфейсов приведено в документе «Perl-интерфейсы».

PHP

Описание PHP-интерфейсов приведено в документе «PHP-интерфейсы».

Python

Описание Python-интерфейса приведено в документе «Python-интерфейс».

Qt

Описание Qt-интерфейса приведено в документе «Qt-интерфейс».

Ruby

Описание Ruby-интерфейса приведено в документе «Ruby-интерфейс».

TCL/TK

Описание TCL/TK-интерфейса приведено в документе «TCL/TK-интерфейс».

ADO.NET

Описание ADO.NET-интерфейса приведено в документе «Поставщики данных ADO.NET».

Функции

Функции, реализованные в СУБД ЛИНТЕР, предназначены для работы со значениями столбцов и/или произвольными выражениями. Они допустимы в любых операторах SQL в тех местах, где синтаксическая конструкция позволяет использовать выражение.

Все функции SQL можно разделить на две группы:

1) встроенные функции;
2) пользовательские функции.

Встроенные функции являются неотъемлемой частью транслятора языка SQL СУБД ЛИНТЕР и не зависят от конкретного экземпляра БД. Приложение, использующее только встроенные функции, является переносимым приложением.

В качестве пользовательских функций в СУБД ЛИНТЕР выступают хранимые процедуры, возвращающие скалярное значение (не массив значений). Скалярная функция всегда возвращает одно значение. Результатом вызова пользовательской функции без аргументов является константа.

Описания пользовательских функций хранятся непосредственно в БД, поэтому обращение к ним возможно только в конкретном экземпляре БД. При переносе приложения, содержащего пользовательские функции, на другую БД, все используемые функции должны быть экспортированы в эту БД. Приложение, использующее пользовательские функции, уже не является переносимым приложением (переносимое приложение должно быть самонастраивающимся, т.е. проверять наличие требуемых функций и при их отсутствии самостоятельно создавать их в БД).

Механизм пользовательских функций позволяет разработчикам приложений расширять стандартный набор встроенных функций с учетом специфики приложений. Использование пользовательских функций повышает производительность клиент-серверных приложений за счет переноса обработки данных непосредственно на сервер БД и уменьшения сетевого трафика. Это является более эффективным решением, чем загрузка из БД всех необходимых для обработки строк таблицы и последующее выполнение расчетов на клиентской машине.

Разработанный набор пользовательских функций для некоторой предметной области позволяет разработчикам приложений в этой предметной области повышать качество разработки за счет использования готовых решений. В данном случае приложения приближаются к объектно-ориентированному виду.

Спецификация пользовательской функции включает:

1) имя схемы;
2) имя функции;
3) список входных параметров и их тип.

В БД может быть несколько функций с одинаковым именем, но созданных в разных схемах. Если при вызове имя схемы явно не указано, то вызывается функция с схемы соответствующей имени владельца, под которым выполняется приложение.

Функция идентифицируется ее схемой, именем функции, количеством и типом параметров. Это называется прототипом функции. Уникальной в экземпляре БД должна быть комбинация <имя схемы> + <имя функции>.

Символьные функции

Конкатенация (CONCAT)

Функция

Слияние (объединение) символьных строк.

Спецификация

Варианты:

1) [1] <синтаксис>::= CONCAT (<строка>, <строка>[, …])
2) [1] <синтаксис>::= <строка> [{|| | +} <строка>]
   [2] <строка>::= <символьное выражение>

Синтаксические правила

1) Аргументы функции должны иметь приводимые друг к другу строковые типы данных.
2) <Символьное выражение> может быть задано <SQL-параметром>, который должен содержать спецификацию типа данных параметра.

```
select concat (? (char(10)), ? (char(5)), :param (char(10)));
123
456
789

|123456789|

select ? (char(10)) + ? (char(5)) || :param (char(10));
123
456
789

|123456789|
```

Общие правила

1) Результатом выполнения функции является объединение символьных строк.
2) Если одно из <символьных выражений> имеет NULL-значение, результатом будет также NULL-значение.
3) Если один из аргументов имеет тип VARCHAR, то тип результата приводится к типу VARCHAR(n), где n – сумма длин объединяемых строк.
4) Если в строковом выражении конкатенации встречается пустая строка '', то она воспринимается как выражение типа CHAR(1) с длиной 1, т.е. одиночный пробел. Если необходимо, чтобы эта пустая строка воспринималась как выражение длины 0, то нужно привести ее к типу VARCHAR: CAST ('' AS VARCHAR).

Примечания

1. Если ядро СУБД запущено с ключом /COMPATIBILITY=STANDARD, то для значений типа CHAR(n) будут усекаться только концевые пробелы крайней строки.

2. Если ядро СУБД запущено с ключом /COMPATIBILITY=ORACLE, то результатом конкатенации символьных значений с NULL-значением будет исходная символьная строка (а не NULL-значение).

Примечание
Ключ /COMPATIBILITY=ORACLE поддерживается со сборки 6.0.17.92.

Примеры

```
select '123       ' || cast '123' as varchar(3);

|123       123|

select cast '123       ' as varchar(5) ||
       cast '123' as varchar(3);

|123  123|

select to_char(rownum) || ' ' || rtrim(firstnam) + ' ' +
  ltrim(name) from person order by name;

| 1 SYLVIA ADKINSON    |
| 2 ED ADKINSON        |
| 3 ETHEL ADKINSON     |
| 4 FRANCOISE ADKINSON |
| 5 VIRGINIA ADKINSON  |
…

update tst set vc=cast vc + '123' as varchar(50);

select concat('Сегодня: ', to_char(sysdate,'dd.mm.yyyy'));

|Сегодня: 14.12.2007|
```

Перевод начальной буквы слова в заглавную (INITCAP)

Функция

Перевод первой буквы каждого слова строки в заглавную.

Спецификация

[1] <синтаксис>::= INITCAP (<строка>)

Синтаксические правила

1) <Символьное выражение> может быть задано <SQL-параметром>, который должен содержать спецификацию типа данных параметра.

```
select initcap(? (char(50)));
```


© Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
aaa bbb 123 cc ddd

|Aaa Bbb 123 Cc Ddd                                    |
```

Общие правила

1) Функция возвращает строку того же типа и той же длины.
2) Разделителями слов в <строке> являются все коды со значением не больше кода пробела.
3) Если аргумент является NULL-значением, возвращается NULL-значение.

Пример

```
select initcap(lower(to_char(rownum) || ' ' || rtrim(firstnam) +
  ' ' + ltrim(name))) from person order by name;

| 1 Sylvia Adkinson    |
| 2 Ed Adkinson        |
| 3 Ethel Adkinson     |
| 4 Francoise Adkinson |
| 5 Virginia Adkinson  |
| 6 Richie Adkinson    |
| 7 Virginia Adkinson  |
```

Поиск подстроки (INSTR/POSITION)

Функция

Поиск подстроки в заданной строке.

Спецификация

Варианты:

[1] <синтаксис>::=
INSTR (<строка>, <подстрока>[, <начало поиска>[, <номер вхождения>]])
[1] <синтаксис>::= POSITION (<подстрока> IN <строка>)
[2] <подстрока>::= <символьное выражение>
[3] <начало поиска>::= <числовое выражение>
[4] <номер вхождения>::= <числовое выражение>

Общие правила

1) <Символьное выражение> в аргументе может иметь следующие типы данных: CHAR, VARCHAR, NCHAR, NCHAR VARYING.
2) Типы данных <строки> и <подстроки> должны быть приводимыми.
3) Длина <подстроки> не должна быть более 4000.
4) <Начало поиска> задает начальную позицию для поиска <подстроки>. Отсчет начинается с единицы. Если <начало поиска> не задано, по умолчанию принимается значение 1.

5) <Номер вхождения> задает порядковый номер искомой подстроки. Отсчет начинается с единицы. Если <номер вхождения> не задан, по умолчанию принимается значение 1.

6) <Числовое выражение> в <начале поиска> или в <номере вхождения> должно иметь целый тип или приводиться к нему.

7) Функция POSITION эквивалента функции INSTR (<строка>,<подстрока>,1,1) или INSTR (<строка>,<подстрока>).

```
select
position('БД' in 'СУБД ЛИНТЕР'),
instr('СУБД ЛИНТЕР','БД',1,1),
instr('СУБД ЛИНТЕР','БД');

| 3 | 3 | 3 |
```

8) Выполняется поиск подстроки в строке, начиная с заданной позиции и с учетом указанного номера вхождения.

9) Информация о недопустимых значениях входных параметров не возвращается.

10) Все аргументы функции могут быть задано <SQL-параметром>, который должен содержать спецификацию типа данных параметра.

```
select instr(? (varchar(50)), ? (varchar(10)), ? (int), ? (int));
Мы едем, едем, едем в далёкие края
едем
5
2

| 16|
```

Возвращаемое значение

1) Номер позиции, с которой размещается найденная в <строке> заданная <подстрока>.

2) 0, если <строка> имеет нулевую длину, если подстрока не найдена, или входные параметры имеют логически недопустимые значения.

3) NULL-значение, если длина <подстроки> равна 0.

4) Тип возвращаемого значения – INT.

Примеры

```
select rownum,rtrim(firstnam)+' '+ltrim(name) from person
 where instr(firstnam,'EDWARD')<>0 order by name;

| 1 | EDWARD KING    |
| 2 | EDWARD TRAVIS  |
| 3 | EDWARD WOOLSEY |
| 4 | EDWARD WYLLIS  |

select phone,rtrim(firstnam)+' '+ltrim(name) from person
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
 where instr(phone,'99',4,2)<>0 order by name;

| 257-9999 |   | BILL SPIEGEL   |
| 713-9996 |   | GERALD SPIEGEL |
```

```
create table tab1 (i int, d dec, c char(5));
create table tab2 (vc varchar(10));
insert into tab1 values(0,2.7,'12345');
insert into tab1 values(-3,1.2, '59202');
insert into tab2 values('ab23cd3456');
insert into tab2 values('cda4978ee5');

select vc,
       c,
       d,
       abs(i+2) as "abs(i+2)",
       substr(c,d, abs(i+2)) as "substr",
       instr(vc,substr(c,d,abs(i+2))) as "instr"
  from tab1,tab2;

| VC          | C     | D   | abs(i+2)| substr | instr |
| ab23cd3456 | 12345 | 2.7 | 2       | 23     | 3     |
| cda4978ee5 | 12345 | 2.7 | 2       | 23     | 0     |
| ab23cd3456 | 59202 | 1.2 | 1       | 5      | 9     |
| cda4978ee5 | 59202 | 1.2 | 1       | 5      | 10    |
```

Проверка вхождения подстроки по шаблону (REGEXP_LIKE)

Функция

Поиск в строке подстроки по шаблону регулярного выражения.

Спецификация

[1] <синтаксис>::=
REGEXP_LIKE (<строка>, <шаблон>[, <сопоставление>])
[2] <шаблон>::= <символьный литерал>
[3] <сопоставление>::= 'c' | 'i'

Общие правила

1) <Символьное выражение> в <строке> может иметь следующие типы данных: CHAR, VARCHAR, NCHAR, NCHAR VARYING.
2) <Символьный литерал> в <шаблоне> должен быть регулярным выражением.
3) <Сопоставление> – модификатор, изменяющий стандартный механизм сопоставления символьных данных:
   - 'c' – сопоставление, чувствительное к регистру символов (по умолчанию);
   - 'i' – сопоставление, нечувствительное к регистру символов;

4) В состав <шаблона> могут входить метасимволы привязки, квантификаторы и операторы повтора, предопределенные символьные классы, группы выражений, метасимволы ссылок (таблица 5).

Таблица 5. Элементы регулярного выражения <шаблона>

Метасимволы привязки	
Метасимвол	**Описание**

^	Привязать выражение к началу строки:
	`select regexp_like('Привет, как дела?', '^Привет');`
	`|true|`
	`select regexp_like('Как дела?  привет', '^Привет');`
	`|false|`

$	Привязать выражение к концу строки:
	`select regexp_like('Привет, как дела?', 'Привет $');`
	`|false|`
	`select regexp_like('Как дела?  привет', 'привет $');`
	`|true|`

Примечание
Привязка строки в функции REGEXP_LIKE несколько отличается от предиката сопоставления LIKE: предиката выражения 'ab' эквивалентно выражению '%ab%', выражение '^ab$' эквивалентно 'ab'.

Квантификаторы и операторы повтора	
Квантификатор	**Описание**

*	Встречается 0 и более раз
	`select REGEXP_LIKE('test111', '1111*');`
	`|false|`
	`select REGEXP_LIKE('11123345', '1111*');`
	`|true|`

?	Встречается 0 или 1 раз `select REGEXP_LIKE('test1', '1111?');`

	`|false|`
	
	`select REGEXP_LIKE('11123345', '1111?');`
	
	`|true|`

+	Встречается 1 и более раз
	`select REGEXP_LIKE('test1', '5+');`
	
	`|false|`
	
	`select REGEXP_LIKE('11123345', '5+');`
	
	`|true|`

{m}	Встречается ровно m раз
	`select REGEXP_LIKE('test1', '3{2}');`
	
	`|false|`
	
	`select REGEXP_LIKE('11123345', '3{2}');`
	
	`|true|`

{m,}	Встречается по крайней мере m раз
	`select REGEXP_LIKE('test1', '3{2,}');`
	
	`|false|`
	
	`select REGEXP_LIKE('11123345','3{2,}');`
	
	`|true|`

{m, n}	Встречается по крайней мере m раз, но не более n раз
	`select REGEXP_LIKE('test1', '3{2,1}');`
	
	`|false|`
	
	`select REGEXP_LIKE('11123345','3{2,2}');`
	
	`|true|`

Предопределенные символьные классы	
Класс	**Описание**
[:alpha:]	Буквы
[:lower:]	Буквы в нижнем регистре
[:upper:]	Буквы в верхнем регистре

[:digit:]	Цифры
[:alnum:]	Буквы и цифры
[:space:]	Пробелы
Альтернативное сопоставление и группировка выражений	
Метасимвол	**Описание**
|	Альтернатива. Разделяет альтернативные варианты, часто используется с оператором группировки ()
( )	Группа. Группирует подвыражения для альтернативы, квантификатора или ссылочности
[char]	Список символов. Большинство метасимволов в списке символов представляют собой литеры, за исключением символьных классов и метасимволов ^ и -
[^char]	Список символов, которые не должны присутствовать в строке
Метасимвол ссылки	
Метасимвол	**Описание**
\digit	Обратная косая черта, за ней следует цифра от 1 до 9. Обратная косая черта связана с предыдущим сопоставлением с соответствующим номером заключенного в скобки подвыражения. Обратная косая черта может иметь другое значение в регулярном выражении; в зависимости от контекста она может означать также символ Escape или обратную ссылку: '\n' – ссылка на результат выражения в скобках. Например, обратная ссылка '\n': (abc|def)xy\1 соответствует строкам abcxyabc и defxydef, но не соответствует ни строке abcxydef, ни строке abcxy. Обратная ссылка позволяет осуществлять поиск строки, не зная заранее фактической строки. Например, выражение ^(.*)\1$ соответствует строке, состоящей из двух идущих подряд одинаковых строк
Метасимволы стандарта языка Perl	
Метасимвол	**Описание**
\d	Означает любую цифру. Эквивалент [0-9]. Например, выражение для проверки ввода телефонного номера ^\(\d{3}\) \d{3}-\d{4}$ С помощью этого шаблона ищется строка, которая начинается с трехзначного числа в скобках (поэтому используем перед символом скобка знак переключения \, чтобы интерпретировать (или) как скобку, а не как метасимвол группы), потом пробел, за которым следует ещё раз трехзначное число, дефис и в заключении четырехзначное число. Номер (123) 456-3456 будет найден, а номер (234) 3456-3234 будет отвергнут
\D	Означает любой символ, кроме цифр. Эквивалент [^0-9]. Например, \d\D может означать 2b или 2$, но не 22
\w	Означает любой алфавитно-цифровой символ, включая символ подчеркивания '_'. Эквивалент [a-zA-Z0-9_]. Например, проверка допустимого адреса электронной почты \w+@\w+(\.\w+)+ С помощью этого шаблона ищется любой

	алфавитно-цифровой символ повторяющийся один или более раз (т.е. просто слово), затем символ @, затем снова слово, а после этого ищется строка по шаблону, состоящему из символа (поэтому используем знак переключения \) и слова, и этот шаблон может повторятся один или несколько раз. Электронный адрес eldar52@mail.ru или eldar52@mail.org.ru будет найден, а eldar52@mail. будет отвергнут. Если известно, что допустимый адрес имеет следующий вид name@mail.domain.something, тогда изменим регулярное выражение на поиск шаблона точно 2 раза \w+@\w+(\.\w+){2}[$] {2} означает, что шаблон (\.\w+) должен встречаться именно 2 раза, а за ним должно следовать либо конец строки, либо пробел
\W	Наоборот (от \w); эквивалент [^a-zA-Z0-9_]
\s	Соответствует любому символу whitespace (пробелу, табуляции и пустой строке); эквивалент [\t\n\r\f\v]
\S	Соответствует любому не-whitespace символу; эквивалент [^ \t\n\r \f\v]

Возвращаемое значение

Значение типа BOOLEAN: подстрока найдена (TRUE) или не найдена (FALSE).

Примеры

1)

```
select regexp_like('axb', 'a.b'),
       regexp_like('xaybx', 'a.b'),
       regexp_like('abba', 'a.b');
|T|T|T|:

select regexp_like('aab', '^a.b$'),
       regexp_like('abb', '^a.b$'),
       regexp_like('abb', '^a.b$');
|T|T|T|:

select regexp_like('axxb', 'a.b'),
       regexp_like('xaybx', '^a.b$'),
       regexp_like('abby', '^a.b$');
|F|F|F|:


select regexp_like('thing of beauty', '^thing'),
       regexp_like('small thing', '^thing'),
       regexp_like('thing of beauty', 'thing$'),
       regexp_like('small thing', 'thing$');
|T|F|F|T|:

select regexp_like('ba', 'b(an)*a'),
```

```
       regexp_like('bana', 'b(an)*a'),
       regexp_like('banana', 'b(an)*a'),
       regexp_like('yourbananasplit', 'b(an)*a');
|T|T|T|T|:

select regexp_like('abcef', '^ab[cd]ef$'),
       regexp_like('abdef', '^ab[cd]ef$');
|T|T|:a

select regexp_like('ab123','[^[:digit:]]'),
       regexp_like('123xy','[^[:digit:]]'),
       regexp_like('007ab','[^[:digit:]]'),
       regexp_like('abcxy','[^[:digit:]]'),
       regexp_like('12345','[^[:digit:]]');
|T|T|T|T|F|:

select regexp_like('Anderson', 'Anders(o|e|a)n'),
       regexp_like('Andersen', 'Anders(o|e|a)n'),
       regexp_like('Andersan', 'Anders(o|e|a)n');
|T|T|T|:

select regexp_like('Don', 'n$'),
       regexp_like('Ron', 'n$'),
       regexp_like('Stivenson', 'n$'),
       regexp_like('Bone', 'n$');
|T|T|T|F|:

select regexp_like('Alex', '^A'),
       regexp_like('Petrov Alex', '^A');
|T|F|:

select regexp_like('eldar52@mail.ru', '\w+@\w+(\.\w+)+'),
       regexp_like('eldar52@mail.org.ru', '\w+@\w+(\.\w+)+'),
       regexp_like('eldar52@mail.', '\w+@\w+(\.\w+)+');
|T|T|F|:

select regexp_like('name@mail.domain.something', '\w+@\w+(\.\w+)
{2}$'),
       regexp_like('name@mail.domain.something.something2', '\w+@
\w+(\.\w+){2}$'),
       regexp_like('name@mail.domain.something.something2', '\w+@
\w+(\.\w+){3}$'),
       regexp_like('name@mail.domain.something', '\w+@\w+(\.\w+)
{2}[$ ]');
|T|F|T|T|:
```

2)

```
CREATE OR REPLACE TABLE
  contacts(l_name VARCHAR(30), p_number VARCHAR(30));
insert into contacts values ('michael','(345) 345-4567');
insert into contacts values ('michael2','345 345-4567');
select * from contacts where
  not REGEXP_LIKE(p_number, '^\(\d{3}\) \d{3}-\d{4}$');

select regexp_like('$$$SYSRL', '[\$\[\]\^\-]{3}'),
       regexp_like('ABC[$]DEF', '[\$\[\]\^\-]{3}'),
       regexp_like('FOO[]$$D', '[\$\[\]\^\-]{3}'),
       regexp_like('FOO-^-D', '[\$\[\]\^\-]{3}'),
       regexp_like('[$$]', '^[\$\[\]\^\-]{3}$');
|T|T|T|T|F|:

select regexp_like('abefff', 'aB(cD)*eF+', 'i'),
       regexp_like('ABCDEF', 'aB(cD)*eF+', 'i'),
       regexp_like('ABCDE', 'aB(cD)*eF+', 'i');
|T|T|F|:
```

3)

```
create or replace table regtest(name varchar(30), birthdate
 varchar(10), city varchar(20));
insert into regtest values('Иванов','01011982','Воронеж');
insert into regtest values('Зайцев','01111998','Киев');
insert into regtest values('Петров','01011988','Москва');

! выборка из таблицы записей, которые содержат
! дату рождения в заданном формате REGEXP [0-9]{8}

select * from regtest where regexp_like(birthdate,'[0-9]{8}');
! результат:
! Зайцев,01111998,Киев
! Иванов,01011982,Воронеж
! Петров,01011988,Москва

! добавление записи с нестандартным форматом даты
insert into regtest values('Волков','010A1988','Дмитров');
select * from regtest where regexp_like(birthdate,'[0-9]{8}');
! результат
! Зайцев,01111998,Киев
! Иванов,01011982,Воронеж
! Петров,01011988,Москва

select regexp_like('_', '^\w$');
```

```
|T|:

select regexp_like('2b', '\d\D'),
       regexp_like('2$', '\d\D'),
       regexp_like('22', '\d\D');
|T|T|F|:

select regexp_like('bce','(a|b)c(d|e)');
|T|:

select regexp_like('xayfa','x(a|b(c|d))y(e|f)\1'),
       regexp_like('xayff','x(a|b(c|d))y(e|f)\2'),
       regexp_like('xayfe','x(a|b(c|d))y(e|f)\2');
|T|T|F|:

select regexp_like('abcxyabc','(abc|def)xy\1'),
       regexp_like('defxydef','(abc|def)xy\1'),
       regexp_like('abcxydef','(abc|def)xy\1'),
       regexp_like('abcxy','(abc|def)xy\1');
|T|T|F|F|:

select regexp_like('forfor','(.*)\1'),
       regexp_like('forfot','(.*)\1');
|T|F|:
```

4)

```
create or replace table tab25574_phone_2
  (id int, fio char(40), mobile_phone char(15));
insert into tab25574_phone_2
  values (101, 'Hhh F.G.', '+7-900-900-9090');
insert into tab25574_phone_2
  values (102, 'Aaa B.C.', '8-700-700-7070');
insert into tab25574_phone_2
  values (102, 'Xx Y.Y.', '8-500-500-5050');

! Выбираются по 3 записи для нижеследующих 6 запросов
select * from tab25574_phone_2
  where REGEXP_LIKE(fio, '[A-Z]{1}[a-z]{1,} ([A-Z]\.){2}');
select * from tab25574_phone_2
  where REGEXP_LIKE(fio, '([A-Z]\.){2}');
select * from tab25574_phone_2
  where REGEXP_LIKE(fio, '[A-Z]{1}[a-z]{1,}');
select * from tab25574_phone_2
  where REGEXP_LIKE(fio, '([A-Z]){1}([a-z]){1,}');
select * from tab25574_phone_2
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
  where REGEXP_LIKE(fio,
   '([[:upper:]]){1}([[:lower:]]){1,} ([[:upper:]]\.){2}');
select * from tab25574_phone_2
  where REGEXP_LIKE(fio,
   '[[:upper:]]{1}([[:lower:]]){1,} ([[:upper:]]\.){2}');
```

Выборка подстроки по шаблону (REGEXP_SUBSTR)

Функция

Поиск в строке подстроки по заданному шаблону.

Спецификация

[1] <синтаксис>::=
REGEXP_SUBSTR (<строка>, <шаблон>
[, <начало поиска>[, <номер вхождения>][, <сопоставление>]])

Общие правила

1) <Символьное выражение> в <строке> может иметь следующие типы данных: CHAR, VARCHAR, NCHAR, NCHAR VARYING.
2) <Символьный литерал> в <шаблоне> должен быть регулярным выражением с форматом, аналогичным формату регулярного выражения функции REGEXP_LIKE.
3) Длина <подстроки> не должна быть более 4000.
4) <Начало поиска> задает начальную позицию для поиска <подстроки> в символах. Отсчет начинается с единицы. Если <начало поиска> не задано, по умолчанию принимается значение 1.
5) <Номер вхождения> задает порядковый номер искомой подстроки. Отсчет начинается с единицы. Если <номер вхождения> не задан, по умолчанию принимается значение 1.
6) <Числовое выражение> в <начале поиска> или в <номере вхождения> должно иметь целый тип или приводиться к нему.
7) <Сопоставление> – модификатор, изменяющий стандартный механизм сопоставления символьных данных:
   - 'c' – сопоставление, чувствительное к регистру символов (по умолчанию);
   - 'i' – сопоставление, нечувствительное к регистру символов.

Возвращаемое значение

1) Подстрока, удовлетворяющая заданному <шаблону> или NULL-значение, если подстрока не найдена. Тип возвращаемого значения определяется типом первого аргумента функции.

Примеры

1)

```
Выделение ссылки на сайт:

SELECT REGEXP_SUBSTR ('See http://relex.ru/ru/documentation/ for
 details', 'https?://[^/]*');
```

```
Результат выполнения примера:
|http://relex.ru                                  |
```

2)

```
Выделение подстрок из строки с разделителем ('|'):

select
  REGEXP_SUBSTR('one|two|three|four|five', '[^\|]+', 1, 1),
  REGEXP_SUBSTR('one|two|three|four|five', '[^\|]+', 1, 2),
  REGEXP_SUBSTR('one|two|three|four|five', '[^\|]+', 1, 3),
  REGEXP_SUBSTR('one|two|three|four|five', '[^\|]+', 1, 4),
  REGEXP_SUBSTR('one|two|three|four|five', '[^\|]+', 1, 5);

Результат выполнения примера:

|one|two|three|four|five|
```

3)

```
create or replace table tab25574_phone03 (
  id int,
  fio char(40),
  mobile_phone char(15),
  check(REGEXP_LIKE(mobile_phone, '[^a-zA-z]'))
  );

! Ошибка
insert into tab25574_phone03 values(1,'aaa','ABC');

! Нормальное завершение
insert into tab25574_phone03 values(1,'aaa','123');

!30|T|:
select REGEXP_LIKE('http://www.linter.ru', 'http://([[:alnum:]]+
\.?){3,4}');
|T|:

select REGEXP_LIKE('http://www.linter.ru/', 'http://([[:alnum:]]+
\.?){3,4}/?');
|T|:

select REGEXP_SUBSTR('http://www.linter.ru/documentation',
 'http://([[:alnum:]]+\.?){3,4}/?');
|http://www.linter.ru/|
```

4)

```
create or replace table regtest(dt char (50));
insert into regtest values('Иванов, 01011982, Воронеж');
insert into regtest values('Зайцев, 01111998, Киев');
insert into regtest values('Петров, 01011988, Москва');
```

```
insert into regtest values('Волков, 010A1988, Дмитров');
select REGEXP_SUBSTR(t.dt,'[0-9]{8}') rr,t.dt rr from regtest t;

select regexp_substr('first field, second field, third field', ',
 [^,]*,');

|01011982   |Иванов, 01011982, Воронеж                          |
|01111998   |Зайцев, 01111998, Киев                             |
|01011988   |Петров, 01011988, Москва                           |
|NULL       |Волков, 010A1988, Дмитров                          |

select regexp_substr(str, '[^\.]+', 1, 1 ) str1,
       regexp_substr(str, '[^\.]+', 1, 2 ) str2,
       regexp_substr(str, '[^\.]+', 1, 3 ) str3,
       regexp_substr(str, '[^\.]+', 1, 4 ) str4,
       regexp_substr(str, '[^\.]+', 1, 5 ) str5,
       regexp_substr(str, '[^\.]+', 1, 6 ) str6,
       regexp_substr(str, '[^\.]+', 1, 7 ) str7,
       regexp_substr(str, '[^\.]+', 1, 8 ) str8,
       regexp_substr(str, '[^\.]+', 1, 9 ) str9,
       regexp_substr(str, '[^\.]+', 1, 10) str10,
       regexp_substr(str, '[^\.]+', 1, 11) str11,
       regexp_substr(str, '[^\.]+', 1, 12) str12
  from (select '01000.7961400000.0.0.0.001.0.K0101.0.0.0.0' str);

|01000|7961400000|0|0|0|001|0|K0101|0|0|0|0|
```

5)

```
CREATE OR REPLACE TABLE t1 (data VARCHAR(50));
INSERT INTO t1 VALUES ('FALL 2014');
INSERT INTO t1 VALUES ('2014 CODE-B');
INSERT INTO t1 VALUES ('CODE-A 2014 CODE-D');
INSERT INTO t1 VALUES ('ADSHLHSALK');
INSERT INTO t1 VALUES ('FALL 2004');

SELECT * FROM t1 WHERE TO_NUMBER(REGEXP_SUBSTR(data, '\d{4}')) >=
 2014;

|FALL 2014                                          |
|2014 CODE-B                                        |
|CODE-A 2014 CODE-D                                 |

delete from t1;
INSERT INTO t1 VALUES ('ArtADB1234567e9876540');
SELECT REGEXP_SUBSTR(data, '[A-Z][a-z]+', 1, 1) col1,
       REGEXP_SUBSTR(data, '[A-Z]+', 1, 2) col2,
       REGEXP_SUBSTR(data, '[0-9]+', 1, 1) col3,
```

```
       REGEXP_SUBSTR(data, '[a-z]+', 1, 2) col4,
       REGEXP_SUBSTR(data, '[0-9]+', 1, 2) col5
  FROM t1;

|Art |ADB |1234567 |e  |9876540 |

delete from t1;

INSERT INTO t1 VALUES ('978/955086/GZ120804/10-FEB-12');
INSERT INTO t1 VALUES ('97/95508/BANANA/10-FEB-12');
INSERT INTO t1 VALUES ('97/95508/"APPLE"/10-FEB-12');
SELECT REGEXP_SUBSTR(data, '[^/"]+', 1, 3) AS element3 FROM t1;

|GZ120804                                          |
|BANANA                                            |
|APPLE                                             |

drop table t1;
select regexp_substr('We are driving south by south
 east','south');
|south                          |

SELECT regexp_substr('655-236-4567', '-[^-]+' );
|-236        |
SELECT regexp_substr('655-236-4567', '-[^\-]+' );
|-236        |
SELECT regexp_substr( '655-236-4567', '-[^-]+-' );
|-236-       |

SELECT regexp_substr('Employee Name and Age: Adam, Dana 28', ':');
|:                                  |
SELECT regexp_substr('Employee Name and Age: Adam, Dana 28', ':
[^,]+');
|: Adam                             |
SELECT regexp_substr('Employee Name and Age: Adam, Dana 28',
 '[[:digit:]]+' );
|28                                 |

SELECT REGEXP_SUBSTR('Hey! Yababa dababa do!','(ab)\1');
|abab                  |

SELECT REGEXP_SUBSTR('ab12cd','[0-9]+');
|12    |

SELECT
  REGEXP_SUBSTR('We are trying to make the subject easier.',
```

```
  'tr(y(ing)?|(ied)|(ies))');
|trying                                    |

SELECT REGEXP_SUBSTR('system/pwd@orabase:1521:sidval','[^:]+', 1,
 3);
|sidval                          |

select REGEXP_SUBSTR('1:3,4:6,8:10,3:4,7:6,11:12','[^:]+', 1, 1);
|1                           |

select REGEXP_SUBSTR('1:3,4:6,8:10,3:4,7:6,11:12','[^,]+', 3, 1);
|3                           |
```

6)

```
CREATE OR REPLACE TABLE test (name VARCHAR(10), version
 VARCHAR(15));
INSERT INTO test values ('A', '1');
INSERT INTO test values ('B', '12.1.0.2');
INSERT INTO test values ('B', '8.2.1.2');
INSERT INTO test values ('B', '12.0.0');
INSERT INTO test values ('C', '11.1.2');
INSERT INTO test values ('C', '11.01.05');

select REGEXP_SUBSTR(version, '\d+', 1, 1) from test;

|1              |
|12             |
|8              |
|12             |
|11             |
|11             |

select REGEXP_SUBSTR(version, '\d+', 1, 2) from test;

|NULL               |
|1              |
|2              |
|0              |
|1              |
|01             |

select regexp_like('abac','(a.){2}'),
       regexp_like('abbc','(a.){2}');

|T|F|

select REGEXP_LIKE('sdfsdsfsd.as.sdsd@jhkk.d.rl',
```

```
  '^[a-z0-9](\.?[a-z0-9_\-]){0,}@[a-z0-9\-]+\.([a-z]{1,6}\.)?[a-z]
{2,6}$');

|T|

select substring('This image is nice'
  similar 'T%\"i[[:alpha:]]+e\"%is [[:alnum:]]+' escape '\');

|image|
```

Корректировка подстроки (INSERT/OVERLAY)

Функция

Корректировка подстроки в заданной строке (удаление подстроки или ее замена).

Спецификация

Варианты:

[1] <синтаксис>::=
INSERT (<строка>, <позиция>, <длина>, <подстрока>)

[1] <синтаксис>::=
OVERLAY (<строка> PLACING <подстрока>
FROM <позиция> [FOR <длина>])

[2] <позиция>::= <числовое выражение>

[3] <длина>::= <числовое выражение>

Общие правила

1) <Символьное выражение> в аргументе может иметь следующие типы данных: CHAR, VARCHAR, NCHAR, NCHAR VARYING, символьное BLOB-значение.
2) Типы данных <строки> и <подстроки> должны быть приводимыми.
3) <Числовое выражение> должно быть целочисленным значением или приводиться к нему.
4) Длина <подстроки> не должна быть более 4000.
5) <Позиция> задает позицию в символах заменяемой подстроки в <строке>. Отсчет начинается с единицы.
6) <Длина> задает количество заменяемых в <строке> символов.
7) <Подстрока> задает заменяющее в <строке> значение.
8) Все аргументы функции могут быть заданы <SQL-параметром>, который должен содержать спецификацию типа данных параметра.

```
select insert (? (char(20)),? (int), ?(int), ? (char(5)));
123 456 789
5
3
abcde

|123 abcde 789   |
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
select OVERLAY  ('123 456 789' PLACING 'abcde' FROM 5 FOR 7);

|123 abcde       |
```

9) Начиная с <позиции>, удаляется <длина> символов, и на место удаленных вставляются символы <подстроки>.

```
select insert('12345',2,3, 'abcd');

| 1abcd5 |

select overlay('12345' placing 'abcd' from 2 for 3);

|1abcd5 |
```

10) В функции insert позиция вставки <подстроки> не должна превышать правую границу <строки>.

```
select insert('12345',5,0, 'abcd');

| 1234abcd5 |

select insert('12345',5,1, 'abcd');

| 1234abcd |
```

Удаление подстроки:

```
select insert('12345',1,3, '');

| 45 |
```

11) В функции overlay позиция вставки <подстроки> может быть за правой границей <строки>. В этом случае добавляемая <подстрока> просто присоединяется к <строке>.

```
select overlay('12345' placing 'abcd' from 20);

|12345abcd|
```

12) В обеих функциях количество заменяемых символов, начиная от заданной <позиции>, может выходить за правую границу <строки>. В этом случае в <строке> удаляются все символы, начиная с <позиции> и до конца <строки>.

```
select insert('12345',5,3, 'abcd');

|1234abcd |

select overlay('12345' placing 'abcd' from 3 for 30);

|12abcd   |
```

13) Если аргумент FOR <длина> не задан (или задано отрицательное значение), в качестве <длины> используется длина <подстроки>.

```
select overlay('12345' placing 'abcd' from 2);
```

```
|1abcd |

select overlay('12345' placing 'abcd' from 2 for -10);

|1abcd |
```

14) Количество заменяемых символов может превышать количество удаляемых, т.е.:

```
select insert('12345',5,1, model) from auto;

|1234MERCURY COMET GT V8 |
|1234A-310 |
…
select insert(model, length(model), 1, 'Год выпуска: ') ||
  to_char(year + 1900, '9999') from auto;
```

15) Длина <подстроки> в байтах должна быть в точности равна количеству байтов, соответствующим символам <длины>.

```
Например, если заменяется часть строки из двух русских символов в
 кодировке UTF-8 (каждый занимает 2 байта), то её можно заменить
 либо на два других двухбайтовых символа, либо на 1 двухбайтовый
 символ и 2 однобайтных, либо на 4 однобайтных символа.
Пример (для утилиты inl):
create or replace table blb(bl blob character set "UTF-8");
insert into blb values (n'АБВГ');
select lenblob(bl,1), getblobstr(bl,1,4,1) from blb;

|          4|АБВГ        |

update blb set bl = insert(bl,2,2,'1234');

select lenblob(bl,1), getblobstr(bl,1,6,1) from blb;

|          6|А1234Г            |
```

16) Длина возвращаемого значения определяется как сумма длин <строки> и <подстроки> (ограничивается максимальным размером 4000 байт).
17) Если хотя бы один из входных аргументов у функций равен NULL-значению, возвращается NULL-значение.

Возвращаемое значение

1) <Строка> со вставленной <подстрокой>.
2) Код завершения при неправильном значении аргумента функции.

Замена всех подстрок (REPLACE)

Функция

Замена всех подстрок в заданной строке.

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

Спецификация

[1] <синтаксис>::=
REPLACE (<строка>, <подстрока1>[, <подстрока2>])
[2] <подстрока1>::= <символьное выражение>
[3] <подстрока2>::= <символьное выражение>

Общие правила

1) <Строка>, <подстрока1>, <подстрока2> должны иметь типы данных: CHAR, VARCHAR, NCHAR, NCHAR VARYING, символьное BLOB-значение.
2) Типы данных <строки>, <подстроки1> и <подстроки 2> должны быть приводимыми.
3) Длина <подстроки1>, <подстроки2> не должна быть более 4000.
4) Окончательная длина <строки> не должна быть более 4000.
5) <Подстрока1> задает удаляемое из <строки> значение.
6) <Подстрока2> задает вставляемое вместо удаленной <подстроки1> значение.
7) Если <подстрока2> не задана, по умолчанию используется пустая строка (т.е. в этом случае происходит фактически удаление <подстроки1>).

```
create or replace table tst (str char(20));
insert into tst values ('123456789abcdef');
select replace(str, '123','321') from tst;

|321456789abcdef     |

select replace(str, '123') from tst;

|456789abcdef        |
```

8) Длины <подстроки1> и <подстроки2> в байтах должна быть равны.
9) Все аргументы функции могут быть заданы <SQL-параметром>, который должен содержать спецификацию типа данных параметра.

```
select replace  (? (char(20)), ? (char(5)), ? (char(5)));
11 22 311 55
11
aa

|aa 22 3aa 55        |
```

Возвращаемое значение

1) Исходная <строка>, в которой все вхождения <подстроки1> заменены на <подстроку2>.
2) Если значение <подстроки1> в <строке> не найдено, <строка> возвращается без изменений.
3) Результирующая строка возвращается без концевых пробелов.
4) Если один из аргументов имеет значение NULL, результат будет NULL.

5) Если аргумент <строка> имеет нулевую длину, возвращается пустая строка.

```
select replace('','1','2');
```

Примеры

```
select replace('123452367238','23','abcdd');

| 1abcdd45abcdd67abcdd8 |

select replace('123452367238','23','a');

| 1a45a67a8 |
```

Удаление всех вхождений подстроки:

```
select replace('123452367238','23', '');

| 145678 |

select replace('123452367238','bb','abc');

| 123452367238 |
```

Замена символов строки (TRANSLATE)

Функция

Замена указанных символов строки другими символами.

Спецификация

[1] <синтаксис>::=
TRANSLATE (<строка>, <подстрока1>, <подстрока2>)

Общие правила

1) <Строка>, <подстрока1>, <подстрока2> должны иметь типы данных: CHAR, VARCHAR, NCHAR, NCHAR VARYING.
2) Типы данных <строки>, <подстроки1> и <подстроки2> должны быть приводимыми.
3) Длина <подстроки1>, <подстроки2> не должна быть более 4000.
4) Окончательная длина <строки> не должна быть более 4000.
5) <Подстрока1> задает набор заменяемых в <строке> символов.
6) <Подстрока2> задает новые значения заменяемых символов.

Примечание
Символы пробела, заданные в конце символьных выражений <строка>, <подстрока1>, <подстрока2> усекаются. Чтобы они принимались во внимание, необходимо использовать явное преобразование типа данных или не задавать пробелы в конце этих выражений.

Например, при выполнении данного запроса символ пробела не учитывается:

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
select translate('111-111 22','- ','');

| 111111 22 |
```

а в этих запросах – учитывается:

```
select translate('111-111 22',' -','');

| 11111122 |
select translate('111-111 22',cast '- ' as varchar,'');

| 11111122 |
```

7) Все аргументы функции могут быть заданы <SQL-параметром>, который должен содержать спецификацию типа данных параметра.

```
select translate(? (char(20)), ? (char(2)), ? (char(2)));
11 22 311 55
12
ab

|aa bb 3aa 55        |
```

Возвращаемое значение

1) Исходная <строка>, в которой каждый символ из <подстроки1> заменен на соответствующий ему символ из <подстроки2>. Например, если <подстрока1>='ac', а <подстрока2>='12', то каждый символ 'a' в исходной <строке> будет заменён на '1', а каждый символ 'c' в исходной <строке> – на '2'.
2) Если <подстрока1> длиннее <подстроки 2>, то все ее лишние символы удаляются из исходной <строки>, поскольку для них нет соответствующих символов в <подстроке2>.
3) Результирующая строка возвращается без концевых пробелов.
4) Если один из аргументов имеет значение NULL, результат будет NULL.
5) Если аргумент <строка> имеет нулевую длину, возвращается пустая строка.

```
select translate('','1','2');
```

6) Функция TRANSLATE теперь утраивает длину входного аргумента для записи ответа только для MBCS кодировок.

Примеры

1) Замена всех прописных букв строчными.

```
select translate(make,make, lower(make)) from auto;
ford
alpine
...
```

2) Исправление текста на кириллице, ошибочно набранного в латинице.

```
select translate('ошибка ddjlf',
```

```
'qwertyuiop[]asdfghjkl;"zxcvbnm,.',
'йцукенгшщзхъфывапролджэячсмитьбю');
|ошибка ввода   |
```

3) Удаление всех гласных из текста.

```
select translate(make, 'EYUIOAJ','') from auto;
FRD
LPN
MRCN MTRS
MSRT
…
```

4) Функция translate является расширением функции replace.

Запрос

```
select translate('abcdaefbgh', 'ab','12');
12cd1ef2gh
аналогичен запросу
select replace (replace('abcdaefbgh', 'a','1'), 'b','2');
12cd1ef2gh
```

Определение длины строки (в символах) (LENGTH)

Функция

Определение длины строки в символах.

Спецификация

[1] <синтаксис>::= [CHAR_ | CHARACTER_] LENGTH (<строка>)

Общие правила

1) <Строка> должно иметь тип CHAR, VARCHAR, NCHAR, NCHAR VARYING.
2) <Строка> может быть задана <SQL-параметром>, который должен содержать спецификацию типа данных параметра.

```
select char_length(? (char(20))), character_length(:param
 (char(10)));
Тестовая строка
Testing line

|         15|         10|
```

3) Для хранимых в таблицах значений CHAR, NCHAR длина считается по усечению концевых пробелов и UNICODE-пробелов соответственно.
4) Функции CHARACTER_LENGTH, CHAR_LENGTH и LENGTH эквивалентны.

```
select max(character_length(model)),
       max(char_length(model)),
       max(char_length(model)) from auto;
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
| 20 | 20 | 20 |
```

Возвращаемое значение

1) Для типа данных CHAR, NCHAR возвращается количество символов в <строке>, которое зависит от режима запуска ядра СУБД:
   - с ключом /COMPATIBILITY=STANDARD – концевые пробелы учитываются;
   - без ключа /COMPATIBILITY=STANDARD – концевые пробелы игнорируются.

```
create or replace table tab1(c char(50));
insert into tab1 values ('1234567890');

! с ключом  /COMPATIBILITY=STANDARD
select length(c) from tab1;
|           50|

! без ключа  /COMPATIBILITY=STANDARD
select length(c) from tab1;

|           10|
```

2) Для типа данных VARCHAR, NCHAR VARYING – реальное количество символов в <строке> (независимо от ключа запуска /COMPATIBILITY=STANDARD).
3) NULL, если <строка> имеет NULL-значение.
4) Если <строка> имеет тип данных NCHAR, NCHAR VARYING, то возвращаемое значение равно L/2 символам, где L – длина <строки> в байтах.
5) Тип возвращаемого значения – INT.

Пример

```
select rtrim(firstnam) + ' ' + ltrim(name)
  from person where length(name + firstnam) =
    (select max(length(name + firstnam)) from person);

| JEFFERSON SPARCK JONES |
| PHYLLIS PAPAYANOPOULOS |
| PHYLLIS PAPAYANOPOULOS |
| MAXWELL PAPAYANOPOULOS |

INL : number of rows shown: 4
```

Определение длины строки (в байтах) (OCTET_LENGTH)

Функция

Определение длины строки в байтах.

Спецификация

[1] <синтаксис>::= OCTET_LENGTH (<строка>)

Общие правила

1) <Строка> должна иметь тип CHAR, VARCHAR, NCHAR, NCHAR VARYING.
2) <Строка> может быть задана <SQL-параметром>, который должен содержать спецификацию типа данных параметра.

```
select octet_length(? (nchar(20)));
Тестовая строка

|         30|
```

Возвращаемое значение

1) Если <строка> имеет тип данных CHAR, VARCHAR, то возвращаемое значение аналогично функции LENGTH.
2) Если <строка> имеет тип данных NCHAR, NCHAR VARYING, то возвращаемое значение равно L*2 байтам, где L – длина <строки> в символах.
3) Тип возвращаемого значения – INT.

Пример

```
create table tab1 (ch char(10), vc varchar(10),
  nc nchar(10), nvc nchar varying(10));
insert into tab1(ch, vc, nc, nvc) values('12345', '12345',
  hex('67458821fc75'), hex('00de54326642'));
select octet_length(ch), octet_length(vc), octet_length(nc),
  octet_length(nvc) from tab1;

| 5 | 5 | 6 | 6 |
```

Перевод символов в нижний регистр (LOWER)

Функция

Перевод всех символов строки в нижний регистр.

Спецификация

[1] <синтаксис>::= LOWER (<строка>)

Общие правила

1) <Строка> может иметь тип данных CHAR, VARCHAR, NCHAR, NCHAR VARYING.
2) <Строка> может быть задана <SQL-параметром>, который должен содержать спецификацию типа данных параметра.

```
select lower(? (char(20)));
Testing Line

|testing line         |
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

Возвращаемое значение

1) <Строка>, в которой все символы имеют строчное представление, т.е. буквы алфавита A-Z, А-Я преобразованы в a-z, а-я.
2) Тип возвращаемого значения совпадает с типом аргумента.
3) Если аргумент NULL, результат NULL.

Пример

```
select lower('USA'),lower (user);

| usa | system |
```

Перевод символов в верхний регистр (UPPER)

Функция

Перевод всех символов строки в верхний регистр.

Спецификация

[1] <синтаксис>::= UPPER (<строка>)

Общие правила

1) <Строка> может иметь тип данных CHAR, VARCHAR, NCHAR, NCHAR VARYING.
2) <Строка> может быть задана <SQL-параметром>, который должен содержать спецификацию типа данных параметра.

```
select upper(? (char(20)));
Тестовая строка

|ТЕСТОВАЯ СТРОКА     |
```

Возвращаемое значение

1) <Строка>, в которой все символы имеют заглавное (прописное) представление, т.е. буквы алфавита a-z, а-я преобразованы в A-Z, А-Я.
2) Тип возвращаемого значения совпадает с типом аргумента.
3) Если аргумент NULL, то результат NULL.

Пример

```
select distinct substr(street,1,instr(street,' ')) + ' ' +
  lower(substr(street,instr(street, ' ') + 1, 2)) || '.'
  from person
  where
  upper(substr(street,instr(street, ' ') + 1,6)) = 'STREET';

| 17TH st.   |
| 49TH st.   |
```

```
| 8TH st.     |
| AMHURST st.|
| EUSTIS st.  |
| FIRST st.   |
| GARBLE st.  |
| HARRY st.   |
| PLACE st.   |
| QUICK st.   |
| SILVER st.  |
| SWEET st.   |
| TWINE st.   |
```

Дополнение строки слева (LPAD)

Функция

Дополнение строки слева заданными символами.

Спецификация

[1] <синтаксис>::=
LPAD (<строка>, <новая длина>[, <дополняемые символы>])
[2] <новая длина>::= <числовой литерал>
[3] <дополняемые символы>::= <символьное выражение>

Общие правила

1) В качестве <строки> можно использовать выражения типа CHAR, VARCHAR, NCHAR, NCHAR VARYING.

2) <Новая длина> должна быть беззнаковым <числовым литералом>.

3) Если <новая длина> больше исходной длины <строки>, то <строка> расширяется слева <дополняемыми символами> до <новой длины> <строки> (возможно, с повторением <дополняемых символов>).

```
select lpad('12345',10, '**');

| *****12345 |
```

4) Если тип данных <строки> CHAR, то концевые пробелы справа удаляются.

5) Если <дополняемые символы> не указаны, по умолчанию <строка> дополняется пробелами.

6) Если значение <новая длина> меньше исходной длины <строки>, то исходная <строка> усекается до заданной <новой длины> справа.

```
select lpad('12345',3, '**');

| 123 |
```

7) Если суммарная длина аргумента <дополняемые символы> и исходной длины <строки> больше, чем указанная <новая длина>, <строка> дополняется только частью аргумента <дополняемые символы>. В этом случае аргумент <дополняемые символы> усекается справа.

```
select lpad('12345',10,'abcdefgh');

| abcde12345 |
```

8) Все аргументы функции могут быть заданы <SQL-параметром>, который должен содержать спецификацию типа данных параметра.

```
select lpad(? (char(20)), ? (int), ? (char(10)));
Тестовая строка
30
12345678

|12345678  12345Тестовая строка|
```

Возвращаемое значение

1) <Строка>, дополненная слева указанными последовательностями символов. Длина <строки> – максимум из исходной длины <строки> и <длины>.
2) Тип возвращаемого значения совпадает с типом аргумента.
3) Если аргумент NULL, результат NULL.

Примеры

```
select substr(firstnam,1,1) || '.' || name,
  case length(ltrim(to_char(salary)))
  when 4 then lpad(ltrim(to_char(salary)), 5, '$')
  else lpad(ltrim(to_char(salary)), 6, '$') end from person;

| J. COLVILLE | $3010  |
| G. POORE    | $8000  |
| J. RAEBIGER | $5100  |
| V. PARK     | $3300  |
| L. MARKUSH  | $8800  |
| F. VAN DUYN | $5100  |
| M. WORLTON  | $2700  |
| S. MILLS    | $3300  |
| C. WELLS    | $16200 |
| J. FARRIS   | $9300  |
| L. STANWOOD | $4200  |
| E. WOOLSEY  | $10800 |
…
select lpad(' ' || to_char(sysdate,'dd.mm.yyyy TIME: hh:mi'),
  36, upper('Current date:'))
  from person
  where personid=2;

|CURRENT DATE: 18.09.1998 TIME: 17:05|
```

```
select rpad(c, 10, substr(vc, 1, 3)) from tst;
```

Дополнение строки справа (RPAD)

Функция

Дополнение строки справа заданными символами.

Спецификация

[1] <синтаксис>::=
RPAD (<строка>, <новая длина>[, <дополняемые символы>])

Общие правила

1) В качестве <строки> можно использовать выражения типа CHAR, VARCHAR, NCHAR, NCHAR VARYING.
2) Если <новая длина> больше исходной длины <строки>, то <строка> расширяется справа <дополняемыми символами> до <новой длины> <строки> (возможно, с повторением <дополняемых символов>).
3) Если <дополняемые символы> не указаны, по умолчанию <строка> дополняется пробелами.
4) Если значение <новая длина> меньше исходной длины <строки>, то исходная <строка> усекается до заданной <новой длины> справа.
5) Если суммарная длина аргумента <дополняемые символы> и исходной длины <строки> больше, чем указанная <новая длина>, <строка> дополняется только частью аргумента <дополняемые символы>. В этом случае аргумент <дополняемые символы> усекается справа.
6) Все аргументы функции могут быть заданы <SQL-параметром>, который должен содержать спецификацию типа данных параметра.

```
select rpad(? (char(20)), ? (double), ? (char(10)));
Тестовая строка
30.67
12345678

|Тестовая строка12345678 12345|
```

Возвращаемое значение

1) <Строка>, дополненная справа указанными последовательностями символов. Длина <строки> – максимум из исходной длины <строки> и <длины>.
2) Тип возвращаемого значения совпадает с типом аргумента.
3) Если аргумент NULL, результат NULL.

Левостороннее удаление символов (LTRIM)

Функция

Удаление заданных символов из строки слева.

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

Спецификация

[1] <синтаксис>::= LTRIM (<строка>[, <подстрока>])

Общие правила

1) Из <строки> удаляются слева символы, указанные в <подстроке>.
2) Если <подстрока> не задана, по умолчанию удаляются пробелы.
3) Если <строка> имеет тип данных CHAR(1), VARCHAR(1) и удаляется содержащийся в ней символ, то <строка> становится пустой: то есть длина строки становится равной 0.
4) Все аргументы функции могут быть заданы <SQL-параметром>, который должен содержать спецификацию типа данных параметра.

```
select ltrim(? (char(20)), ? (char(10)));
это тестовая строка
это

| тестовая строка    |
```

Возвращаемое значение

1) <Строка> с удаленными слева указанными символами.
2) Тип возвращаемого значения совпадает с типом аргумента.
3) Если аргумент NULL, результат NULL.

Пример

```
create or replace table tab1
  (c1 char(10), c2 char(10), c3 char(10), c4 char);
insert into tab1 values ('   123', '??123','123123456', '%');
select '*' || c1,length('*' || c1), '*' || ltrim(c1),
  length('*' || ltrim(c1)) from tab1
  union
select '*' || c2, length( '*' || c2), '*' || trim(c2, '?'),
  length('*' || trim(c2, '?')) from tab1
  union
select '*' || c3,length( '*' || c3), '*' || ltrim(c3, '123'),
  length('*' || ltrim(c3, '123')) from tab1
  union
select '*' || c4, length( '*' || c4), '*' || ltrim(c4, '%'),
  length('*' || ltrim(c4, '%')) from tab1
  union
select '*' || c4, length( '*' || c4), '*' || ltrim(c4, ''),
  length('*' || ltrim(c4,'')) from tab1;

|*   123    |7  |*123 |4 |
|*??123     |6  |*123 |4 |
|*123123456 |10 |*456 |4 |
|*%         |2  |*    |1 |
```

```
|*%          |2  |*%    |2 |
```

Правостороннее удаление символов (RTRIM)

Функция

Удаление заданных символов из строки справа.

Спецификация

[1] <синтаксис>::= RTRIM (<строка>[, <подстрока>])

Общие правила

1) Из <строки> удаляются справа символы, указанные в <подстроке>.
2) Если <подстрока> не указана, по умолчанию удаляются пробелы.
3) Если <строка> имеет тип данных CHAR(1), VARCHAR(1), и удаляется содержащийся в ней символ, то <строка> становится пустой и длина строки станет равной 0.
4) Все аргументы функции могут быть заданы <SQL-параметром>, который должен содержать спецификацию типа данных параметра.

```
select rtrim(? (char(20)), ? (char(10)));
Тестовая строка 24
24

|Тестовая строка     |
```

Возвращаемое значение

1) <Строка> с удаленными справа указанными символами.
2) Тип возвращаемого значения совпадает с типом аргумента.
3) Если аргумент NULL, результат NULL.

Пример

```
create or replace table tab1
  (c1 char(10), c2 char(10), c3 char(10), c4 char);
insert into tab1 values
  ('  123     ', '?123???','456123123', '%');
select c1 ||  '   *',length( c1 || '   *'), rtrim(c1) || '*',
 length( rtrim(c1) || '*') from tab1
 union
select c2 ||  '*', length( c2 || '*'), rtrim(c2, '?' ) ||  '*',
 length(rtrim(c2, '?') || '*') from tab1
 union
select c3 || '*', length( c3 || '*'), rtrim(c3, '123') || '*',
 length(rtrim(c3, '123') || '*') from tab1
 union
select c4 || '*', length(c4 || '*'), rtrim(c4, '%') || '*',
 length(rtrim(c4, '%') || '*') from tab1;
```


© Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
|  123   *    | 9  |  *123 | 6 |
| ?123???*    | 8  | ?123* | 5 |
| %*          | 2  | *     | 1 |
| 456123123*  | 10 | 456*  | 4 |
```

Удаление из строки крайних символов (TRIM)

Функция

Удаление из строки заданных крайних левых и/или правых символов.

Спецификация

Варианты:

1) [1] <синтаксис>::= TRIM (<строка>[, <подстрока>])
2) [1] <синтаксис>::=
   TRIM ([[LEADING | TRAILING | BOTH] [<подстрока>] FROM] <строка>)

Общие правила

1) Опция LEADING заставляет удалять крайние левые символы <подстроки> из <строки>.
2) Опция TRAILING заставляет удалять крайние правые символы <подстроки> из <строки>.
3) Опция BOTH заставляет удалять одновременно крайние левые и правые символы <подстроки> из <строки>.
4) Если опции LEADING, TRAILING и BOTH не заданы, удаляются одновременно крайние левые и правые символы <подстроки> из <строки>.
5) Если <подстрока> не указана, по умолчанию удаляются пробелы.
6) В первом варианте функции из <строки> удаляются справа (если найдены) и слева (если найдены) символы, указанные в <подстроке>.
7) Все аргументы функции могут быть заданы <SQL-параметром>, который должен содержать спецификацию типа данных параметра.

```
select trim(? (char(20)), ? (char(10)));
##Тестовая строка ######
##

|Тестовая строка     |

select TRIM (TRAILING ? (char(10))  FROM ? (char(20)));
##
##Тестовая строка ######

|##Тестовая строка   |
```

Возвращаемое значение

1) <Строка> с удаленными в соответствии с заданной опцией символами.

2) Тип возвращаемого значения совпадает с типом аргумента;

3) Если аргумент NULL, результат NULL.

```
select trim('*** Пользователи БД ***','*');

|Пользователи БД |
```

сравните

```
select rtrim(ltrim('*** Пользователи БД***','*'),'*');

|Пользователи БД |
```

```
select trim(trailing '*' from '*** Пользователи БД***');

|*** Пользователи БД |
```

```
select trim(leading substring('abcd' from 1 for 1) from substring
 ('aabcda' from 2 for 5));

|bcda   |
```

Выделение подстроки (SUBSTR/SUBSTRING)

Функция

Выделение подстроки из заданной строки.

Спецификация

Варианты:

1) [1] <синтаксис>::= SUBSTR (<строка>, <начало>[, <длина>])

2) [1] <позиционное выделение подстроки>::=
SUBSTRING (<строка> [{FROM|,} <начало>] [{FOR|,} <длина>])

3) [1] <выделение подстроки по шаблону>::=
SUBSTRING (<строка> SIMILAR <шаблон> ESCAPE <ESC-символ>)
[2] <шаблон>::=
<регулярное выражение1> <ESC-символ>
"<регулярное выражение2> <ESC-символ>"<регулярное выражение3>

Общие правила

1) В качестве <строки> можно использовать выражения типа CHAR, VARCHAR, NCHAR, NCHAR VARYING.

2) <Длина> подстроки должна задаваться в диапазоне от 0 до N, где N = length(<строка>) - <начало> + 1.

3) Формат <шаблона> регулярных выражений описан в пункте «Предикат сопоставления».

4) В вариантах 1 и 2 из <строки> выбирается подстрока заданной <длины>, начиная с позиции <начало>.

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

5) Если опция [FROM |, <начало>] не задана, по умолчанию принимается FROM 1. При этом обязательно должно использоваться ключевое слово FOR.

```
select substring('testing' for 4);

|test|
```

6) Если <длина> не задана, конечная позиция выбираемой подстроки определяется как максимум из двух значений: <начало> и длина <строки>.

7) Если <начало> больше длины <строки>, или вычисленная конечная позиция меньше 1, возвращается пустая строка.

Длина подстроки не задана:

```
select substring('123456', 5);
|56|

select substring('123456', 10);

|          |
```

Позиция <начала> больше длины <строки>:

```
select length(substring('123456', 8.2));

|          0|
```

Конечная позиция меньше 1:

```
select length(substring('123456', 2,-5));

|          0|
```

8) По умолчанию подстрока выбирается с позиции <начало> до конца исходной строки.

```
select substring('testing' from 3), substring('testing', 3);

| sting | sting |

select substring(cast 123456 as char(6), 4);

| 456 |
```

9) Не целочисленное значение параметров <начало> и <длина> усекается до целого значения.

```
select substring('123456', 5.7);

|56|
```

Конструкция

```
select substr(model, 3, 4) from auto;
```

и

```
select substring(model from 3 for 4) from auto;
```

эквивалентны.

```
| RCUR |
| 310  |
| TADO |
  …
```

10) Если <длина> не задана, то конечная длина подстроки определяется как максимум из двух значений: <начало> и длина <строки> (в случае типа данных VARCHAR конечным считается последний фактический символ).

```
select substr(model, 3) from auto;

| RCURY COMET GT V8 |
| 310               |
| TADOR STATION     |
…
```

11) Если <длина> равна 0, возвращается пустая строка.

12) <ESC-символ> в обязательном аргументе функции задает разделитель шаблона регулярного выражения.

    Предположим, что в качестве ESC-символа задан символ "x".

    Тогда символьная строка, задаваемая во втором операнде, должна иметь вид '<рег1>"рег2"рег3', где рег1, рег2 и рег3 являются регулярными выражениями.

    Функция пытается разделить <строку> на три раздела, первый из которых определяется путем сопоставления начала строки со строками, генерируемыми <рег1>, второй – путем сопоставления оставшейся части <строки> с <рег2>, и третий – путем сопоставления конца этой строки с <рег3>.

    В случае отсутствия в <шаблоне> двух пар, состоящих из следующих друг за другом ESC-символа и "", будет выдан код завершения 1125 («Неверный символ ESCAPE»).

```
select substring('This image is nice' similar 'T%\"i[[:alpha:]]+e
\"%is [[:alnum:]]+' escape '\');
```

Результатом будет строка 'image'.

```
select substring('This is string22' similar 'This is
 \"[[:ALPHA:]]+\"[[:DIGIT:]]+' escape '\');
```

Результатом будет строка 'string'.

13) Все аргументы функции могут быть заданы <SQL-параметром>, который должен содержать спецификацию типа данных параметра.

```
select substr (? (char(20)), ? (int), ? (double));
select substring (? (char(20)) from  ? (int) for ? (double));
Тестовая строка
10
```

```
3.67

|стр                    |
```

Возвращаемое значение

1) Возвращается либо подстрока заданной <длины>, либо подстрока с символами от позиции <начала> до конца <строки>, дополненная справа пробелами до заданной <длины>.
2) В случае использования регулярного выражения возвращается средняя часть <строки>.
3) Если тип данных <строки> CHAR, тип результата CHAR.
4) Если тип данных <строки> VARCHAR, тип результата VARCHAR.
5) Если значение хотя бы одного из операндов не определено (NULL) или если <строка> не подходит по <шаблону>, возвращается NULL-значение.
6) Если в качестве второго или третьего параметра указано отрицательное значение, возвращается код завершения 1036 - Значение аргумента в недопустимом диапазоне.

Пример

```
create table tab1 (c char(20));
insert into tab1 values(user);
insert into tab1 values('Систем');
select c,
       user as "user",
       substr(user, case instr(c, 'SYS')
                      when 0 then 4
                      else instr(c, 'SYS') end, 2) as "substr"
  from tab1;

| C       | User   | Substr |
| SYSTEM | SYSTEM | SY     |
| Систем | SYSTEM | TE     |
```

Выделение последних символов строки (RIGHT_SUBSTR)

Функция

Выделение заданного количества последних символов строки.

Спецификация

[1] <синтаксис>::= RIGHT_SUBSTR (<строка>, <количество>)
[2] <количество>::= <числовое выражение>

Общие правила

1) В качестве <строки> можно использовать выражения типа CHAR, VARCHAR, NCHAR, NCHAR VARYING.

2) Все аргументы функции могут быть заданы <SQL-параметром>, который должен содержать спецификацию типа данных параметра.

```
select right_substr (? (char(20)), ? (int));
Тестовая строка
6

|строка              |
```

Возвращаемое значение

1) Возвращается заданное <количество> символов, начиная с конца <строки>.
2) Если длина <строки> меньше заданного <количества>, то возвращается полностью исходная <строка>.

```
create table tab1 (ch varchar(20));
insert into tab1(ch) values('ФИО: Иванов П. А.');
select right_substr(ch, length(ch) - 5) from tab1;

|Иванов П. А. |
```

Дублирование строки (REPEAT_STRING)

Функция

Дублирование строки заданное число раз.

Спецификация

[1] <синтаксис>::= REPEAT_STRING (<строка>, <количество>)
[2] <количество>::= <числовой литерал>

Общие правила

1) В качестве <строки> можно использовать выражения типа CHAR, VARCHAR, NCHAR, NCHAR VARYING.
2) Аргумент <строка> может быть задан <SQL-параметром>, который должен содержать спецификацию типа данных параметра.

```
select 'Мы '+ repeat_string (? (char(20)), 3) + 'в далекие края.';
едем,

|Мы едем,едем,едем,в далекие края.   |
```

3) Результирующая длина <строки> не должна превышать максимально допустимую длину для типа данных исходной <строки> (например, 4000 для типа данных char).

Возвращаемое значение

Строка, являющаяся конкатенацией исходной <строки> заданное <количество> раз.

Примеры

```
select repeat_string('*',20);
```


© Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
|********************|
```

```
create table tab1 (c varchar(2) default 'xa');
insert into tab1 default values;
select repeat_string( c || '-', 2) || c || '!' from tab1;

|xa-xa-xa! |
```

Обратный порядок символов строки (REVERSE)

Функция

Исходная строка с обратным порядком символов.

Спецификация

[1] <синтаксис>::= REVERSE (<строка>)

Общие правила

1) В качестве <строки> можно использовать выражения типа CHAR, VARCHAR, NCHAR, NCHAR VARYING.
2) Аргумент <строка> может быть задан <SQL-параметром>, который должен содержать спецификацию типа данных параметра.

```
select reverse(? (char(10)));
12345

|54321     |
```

Возвращаемое значение

Исходная строка с обратным порядком символов.

Примеры

1)

```
select reverse ('abcd');

|dcba|
```

2) Подсчитать количество строк, у которых совпадает прямое и обратное значение

```
create or replace table words (name char(5));
insert into words values('пенал');
insert into words values('заказ');
insert into words values('казак');
insert into words values('замок');
insert into words values('шалаш');
select count(*) from words where name = reverse(name);

|                  3|
```

Символьное представление байта (CHR)

Функция

Получить символьное представление байта.

Спецификация

[1] <синтаксис>::= CHR (<числовое значение>)

Общие правила

1) <Числовое значение> должно быть целым положительным десятичным числом в диапазоне от 0 до 255 или приводиться к нему. Для <числового значения> в диапазоне от 128 до 255 выдается результат '?'.
2) Аргумент <числовое значение> может быть задан <SQL-параметром>, который должен содержать спецификацию типа данных параметра.

```
select chr(? (int));
46

|.|
```

Возвращаемое значение

Возвращается символьное представление заданного байта в кодировке БД, заданной по умолчанию, или пробел, если соответствующий символ отсутствует в текущей кодировке БД.

Пример

```
select chr(123) || to_char(sysdate) || chr(125);

|{27-OCT-02} |
```

Числовое представление символа (ASCII)

Функция

Получить числовое представление первого символа символьного значения.

Спецификация

[1] <синтаксис>::= ASCII (<символьное выражение>)

Общие правила

1) <Символьное выражение> должно быть в ASCII-кодировке.
2) <Символьное выражение> может быть NULL-значением.
3) Аргумент <символьное выражение> может быть задан <SQL-параметром>, который должен содержать спецификацию типа данных параметра.

```
select ascii(? (char(20)));
Тестовая строка
```


© Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
|        146|
```

Возвращаемое значение

Возвращается значение типа INTEGER первого символа <символьное выражение> или NULL, если <символьное выражение> является NULL-значением или имеет длину 0.

Примеры

1)

```
select ascii('z'), nvl(cast ascii(null) as char,'null'),
 ascii('SYSTEM');

|        122|NULL       |         83|
```

2)

```
! Подсчитать количество телефонов, начинающихся с цифры 2
create or replace table tst (tel char(9));
insert into tst values ('273-56-45');
insert into tst values ('450-66-30');
insert into tst values ('2-711-711');
select count(*) from tst where ascii(tel)=ascii('2');

|          2|
```

Символьное представление значения по умолчанию столбца (DEFTEXT)

Функция

Получить символьное представление значения по умолчанию столбца таблицы.

Спецификация

[1] <синтаксис>::=
DEFTEXT (<идентификатор таблицы>, <идентификатор столбца>[, <строка>])
[2] <идентификатор таблицы>::= целое положительное число
[3] <идентификатор столбца>::= целое положительное число

Синтаксические правила

1) <Идентификатор таблицы> задает системный идентификатор таблицы (поле $$$S11 системной таблицы $$$SYSRL), в которой находится нужный столбец.

   Получить системный идентификатор таблицы AUTO:

```
select $$$S11 from $$$SYSRL where $$$S13 = 'AUTO';

|100 |
```

2) <Идентификатор столбца> задает системный идентификатор нужного столбца (поле $$$S22 системной таблицы $$$ATTRI).

Получить системный идентификатор столбца MODEL таблицы AUTO:

а) Получить системный идентификатор таблицы AUTO:

```
select $$$S11 from $$$SYSRL where $$$S13 = 'AUTO';

|100 |
```

б) Получить системный идентификатор столбца MODEL:

```
select $$$S22 from $$$ATTRI where $$$S21 = 100 and $$$S23='MODEL';

|2 |
```

3) <Строка> задает строку, которая должна выдаваться в случае, если DEFAULT-значение для столбца не задано или имеет NULL-значение. По умолчанию используется фраза «NULL».

Получить значение по умолчанию столбца MODEL таблицы AUTO:

```
select deftext(100,2,'unknown');

|unknown |
```

4) Аргументы <идентификатор таблицы> и <идентификатор столбца> могут быть заданы <SQL-параметром>, который должен содержать спецификацию типа данных параметра.

```
select deftext(? (int), ? (int), 'нет значения по умолчанию');
173
2

|нет значения по умолчанию |
```

Возвращаемое значение

1) Возвращаемое значение имеет тип VARCHAR(2048).
2) DEFAULT-значения столбца возвращаются в двойных кавычках.
3) Если длина DEFAULT-значения столбца больше 2048, это значение обрезается.
4) В случае если DEFAULT-значение обрезано, возвращается значение без кавычек.
5) Для типов BYTE/VARBYTE значение возвращается в шестнадцатеричном виде (HEX-строка). Соответственно, для этих типов максимальное число выведенной информации о DEFAULT-значении равно 1024, т.к. 1 байт соответствует 2 текстовым символам.
6) Если DEFAULT-значение не установлено или равно NULL, возвращается фраза NULL без двойных кавычек.
7) Для столбцов с атрибутом GENERATED BY DEFAULT AS или DEFAULT в качестве <значимого выражения> возвращается строка NULL. Это связано с тем, что <значимое выражение> в данном случае вычисляется непосредственно в момент добавления новой записи (т.е. заранее неизвестно). Получить текстовый вид <значимого выражения> по умолчанию можно с помощью функции LINTER_DICT_INFO, например:

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
create or replace table q1(j int, i int default 10);
alter table q1 alter column i set default (j + 9);
select LINTER_DICT_INFO(2,$$$S11,2)
  from LINTER_SYSTEM_USER.$$$SYSRL where $$$S13='Q1';

|("J"+9)    |
```

Пример

```
create table tab1 (i int default 100);
select cast rtrim($$$S13) as varchar(66), /*таблица*/
       cast rtrim($$$S23) as varchar(66), /*столбец*/
       deftext($$$S11, $$$S22) /*значение по умолчанию*/
  from LINTER_SYSTEM_USER.$$$SYSRL, LINTER_SYSTEM_USER.$$$ATTRI
 where $$$S13 = 'TAB1' and $$$S11=$$$S21;

|TAB1 |I |"100" |
```

Получить список ключевых слов СУБД ЛИНТЕР (LINTER_KEYWORDS)

Функция

Получить список ключевых слов СУБД ЛИНТЕР.

Спецификация

[1] <синтаксис>::= LINTER_KEYWORDS ()

Возвращаемое значение

1) Возвращаемое значение имеет тип `VARCHAR(4000)` и содержит список ключевых слов СУБД ЛИНТЕР, разделенных запятыми.

Математические функции

Общая рекомендация

Аргументы всех математических функций могут задаваться в виде <SQL-параметра>, который должен содержать спецификацию типа данных параметра, например,

```
select abs(? (double));
-67.41

|                67.41|

select mod(? (double), ? (int));
236.78
23
```

```
|      6.78000000000003|
```

СУБД ЛИНТЕР поддерживает два типа приближенных вещественных данных: REAL и DOUBLE (DOUBLE PRECISION), различающихся между собой точностью представления данных (тип данных REAL имеет меньшую точность по сравнению с DOUBLE). При математических вычислениях с использованием одновременно разных вещественных типов данных точность промежуточных (или окончательных) результатов может снижаться. Поэтому рекомендуется при использовании функций от аргументов типа DOUBLE PRECISION хранить значения аргументов этих функций также в столбцах типа DOUBLE PRECISION, а не REAL.

Пример

Сравнение точности результатов при использовании вещественных типов данных разной точности.

```
create or replace table t_double (d double precision);
create or replace table t_real(r real);
insert into t_real values (1e+20);
insert into t_real values (999.4);
insert into t_double select * from t_real;

select * from t_double;

 D
 -
| 1.00000002004088e+20|
|     999.400024414063|
```

```
create or replace table t1_double (d1 double precision);
create or replace table t2_double (d2 double precision);
insert into t2_double values (1e+20);
insert into t2_double values (999.4);
insert into t1_double select * from t2_double;

select * from t1_double;

 D1
 -
|                1e+20|
|                999.4|
```

```
create or replace table tst (r real);
insert into tst values (12.35635);
insert into tst values (98.61324);
insert into tst values (1e+20);

select ceil(r), floor(r) from tst;

|    13|    12|
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
|    99|    98|
| 1e+20| 1e+20|
```

Из примера видно, что результат округления с недостатком оказался больше исходного значения. Для правильного результата нужно использовать тип данных DOUBLE:

```
create or replace table tst (d double);
insert into tst values (12.35635);
insert into tst values (98.61324);
insert into tst values (1e+20);
select ceil(d), floor(d) from tst;

|                  13|                  12|
|                  99|                  98|
|                1e+20|                1e+20|
```

Примечание

Для сравнения вещественных чисел с заданной точностью рекомендуется использовать следующую конструкцию:

```
select abs(a-b)/greatest(abs(a),abs(b)), abs(a-b) <= DBL_EPSILON *
 greatest(abs(a),abs(b)) from test_tbl;
```

где DBL_EPSILON (или FLT_EPSILON для FLOAT) – заданная точность сравнения. Пример:

```
create or replace table test_tbl(a float, b float);
insert into test_tbl
 values(0.000585955393034, 0.000585955393035);
insert into test_tbl
 values(0.00058595539303496, 0.00058595539303497);
insert into test_tbl
 values(0.000585955393034964, 0.000585955393034965);
insert into test_tbl
 values(0.0005859553930349654444, 0.0005859553930349654446);

select abs(a-b)/greatest(abs(a),abs(b)) relative_difference,
       abs(a-b) <= 1e-12 * greatest(abs(a),
       abs(b)) comparison_result from test_tbl;

|relative_difference  |comparison_result|
|1.70673074399179e-12 |F                |
|1.68378683546467e-14 |T                |
|1.66528368342661e-15 |T                |
|0                    |T                |

select abs(a-b)/greatest(abs(a),abs(b)) relative_difference,
       abs(a-b) <= 1e-14 * greatest(abs(a),
       abs(b)) comparison_result from test_tbl;
```

```
|relative_difference  |comparison_result|
|1.70673074399179e-12 |F                |
|1.68378683546467e-14 |F                |
|1.66528368342661e-15 |T                |
|0                    |T                |

select abs(a-b)/greatest(abs(a),abs(b)) relative_difference,
       abs(a-b) <= 1e-16 * greatest(abs(a),
       abs(b)) comparison_result from test_tbl;

|relative_difference  |comparison_result|
|1.70673074399179e-12 |F                |
|1.68378683546467e-14 |F                |
|1.66528368342661e-15 |F                |
|0                    |T                |
```

Вычисление абсолютного значения (ABS)

Функция

Вычисление абсолютного значения числа.

Спецификация

[1] <синтаксис>::= ABS (<числовое выражение>)

Возвращаемое значение

1) Абсолютное значение числа. Тип возвращаемого значения совпадает с типом `<числового выражения>`.
2) Если аргумент `NULL`, результат `NULL`.

Пример

```
select abs (5.7), abs(4-65.2), abs(avg(salary) - max(salary))
  from person;

|5.7 |61.2 |40448.1643002029 |
```

Округление до большего целого значения (CEIL)

Функция

Округление числа до ближайшего большего целого значения.

Спецификация

[1] <синтаксис>::= CEIL (<числовое выражение>)

Возвращаемое значение

1) Результат округления (до целого) с избытком `<числового выражения>` – наименьшее целое значение, большее или равное аргументу.

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

2) Тип возвращаемого значения соответствует типу <числового выражения>.
3) Если аргумент равен NULL-значению, результат NULL-значение.

Округление до меньшего целого значения (FLOOR)

Функция

Округление числа до ближайшего меньшего целого значения.

Спецификация

[1] <синтаксис>::= FLOOR (<числовое выражение>)

Возвращаемое значение

1) Результат округления (до целого) с недостатком <числового выражения>.
2) Тип возвращаемого значения соответствует типу <числового выражения>.
3) Если аргумент равен NULL-значению, результат NULL-значение.

Пример

```
select ceil(6.7), floor(6.7), ceil(-6.7),
       floor(-6.7), ceil(avg(salary) - max(salary))
  from person;

|7 |6 |-6 |-7 |-40448 |
```

Остаток от деления (MOD)

Функция

Получить остаток от деления.

Спецификация

[1] <синтаксис>::=
MOD (<числовое выражение 1>, <числовое выражение 2>)
[2] <числовое выражение 1>::=
значение типа REAL или приводимое к нему в диапазоне [2^{53}-1, -2^{53}+1]
[3] <числовое выражение 2>::=
значение типа REAL или приводимое к нему

Возвращаемое значение

1) Возвращается остаток от деления <числового выражения 1> на <числовое выражение 2>.
2) Тип возвращаемого значения – DOUBLE.
3) Верхняя и нижняя границы возвращаемого значения равны (2^{53}-1) и (-2^{53}+1) соответственно.

Примеры

```
select mod(10.5, 3),
       mod(10.5, -3),
       mod(-10.5, -3),
```

```
       mod(1.0, 0.999999);

|1.5 |1.5 |-1.5 |1.00000000002876e-006 |
select distinct mod(year+1900,1950) from auto;

|20 |
|21 |
```

Тригонометрические функции (COS/SIN/TAN)

Функция

Вычисление тригонометрических функций.

Спецификация

[1] <синтаксис>::= COS (<числовое выражение>)
[2] <синтаксис>::= SIN (<числовое выражение>)
[3] <синтаксис>::= TAN (<числовое выражение>)
[4] <числовое выражение>::= угол, выраженный в радианах

Возвращаемое значение

1) Значение косинуса (COS), синуса (SIN) и тангенса (TAN) <числового выражения> в радианах.
2) Тип возвращаемого результата – DOUBLE.
3) Если аргумент NULL, результат NULL.
4) Если значение аргумента вне диапазона $(-2^{53}+1)$; $(2^{53}-1)$, возвращается код завершения 1036 «Значение аргумента в недопустимом диапазоне».

Обратные тригонометрические функции (ACOS/ASIN/ ATAN/ATAN2)

Функция

Вычисление обратных тригонометрических функций.

Спецификация

[1] <синтаксис>::= ACOS (<числовое выражение>)
[2] <синтаксис>::= ASIN (<числовое выражение>)
[3] <синтаксис>::= ATAN (<числовое выражение>)
[4] <синтаксис>::=
ATAN2 (<числовое выражение 1>, <числовое выражение 2>)

Синтаксические правила

1) Тип данных <числового выражения> должен быть DOUBLE или приводиться к нему.

Возвращаемое значение

1) Значение обратного косинуса (ACOS), обратного синуса (ASIN), обратного тангенса (ATAN) <числового выражения> в радианах и значение обратного

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

тангенса частного <числовое выражение 1>/<числовое выражение 2> (ATAN2).

2) Тип возвращаемого результата – DOUBLE.

3) Если аргумент NULL, результат NULL.

Гиперболические функции (COSH/SINH/TANH)

Функция

Вычисление гиперболических функций.

Спецификация

[1] <синтаксис>::= COSH (<числовое выражение>)
[2] <синтаксис>::= SINH (<числовое выражение>)
[3] <синтаксис>::= TANH (<числовое выражение>)

Возвращаемое значение

1) Значение гиперболического косинуса (COSH), синуса (SINH) и тангенса (TANH) <числового выражения> в радианах.

2) Тип возвращаемого результата – DOUBLE.

3) Если аргумент NULL, результат NULL.

Вычисление экспоненты (EXP)

Функция

Вычисление экспоненциального значения.

Спецификация

[1] <синтаксис>::= EXP (<числовое выражение>)

Возвращаемое значение

1) Экспоненциальное значение <числового выражения>.

2) Тип возвращаемого результата – DOUBLE.

3) Если аргумент NULL, результат NULL.

Вычисление логарифмических значений (LN/LOG)

Функция

Вычисление логарифмических значений.

Спецификация

[1] <синтаксис>::= LN (<числовое выражение>)
[2] <синтаксис>::= LOG (<основание>, <числовое выражение>)
[3] <основание>::= <числовой литерал>

Синтаксические правила

1) <Основание> должно быть беззнаковым <числовым литералом>.

Возвращаемое значение

1) Натуральный логарифм (LN) и логарифм <числового выражения> по заданному <основанию> (LOG).
2) Тип возвращаемого результата – DOUBLE.
3) Если аргумент NULL, результат NULL.

Вычисление степени (POWER)

Функция

Вычисление степени числа.

Спецификация

[1] <синтаксис>::= POWER (<числовое выражение>, <экспонента>)
[2] <экспонента>::= <числовое выражение>

Возвращаемое значение

1) <Числовое выражение>, возведенное в степень <экспоненты>.
2) Тип возвращаемого результата – DOUBLE.
3) Если аргумент NULL, результат NULL.

Округление значения (ROUND)

Функция

Округление значения с заданной точностью.

Спецификация

[1] <синтаксис>::= ROUND (<выражение>, <точность>)
[2] <выражение>::=
<числовое выражение> | <выражение типа «дата-время»>
[3] <точность>::=
целочисленное значение | <элемент формата «дата-время»>
[4] <элемент формата «дата-время»>::=
{'Y'|'Q'|'M'|'D'|'DY'|'DAY'|'HH'|'HH12'|'HH24'|'MI'}

Синтаксические правила

1) <Выражение типа «дата-время»> должно иметь тип DATE.
2) <Точность> задает точность округления целочисленного значения:
   - при положительном значении – точность округления после десятичной точки (количество цифр после десятичной точки). Для целочисленных значений не используется;
   - при отрицательном значении – точность округления перед десятичной точкой, т.е. при «-1» – до ближайшего целого десятка («-2» – до сотни, «-3» тысячи и т.д.).



Примечание
При округлении до значения 1e-13 числа считаются равными.

© Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

3) <Элемент формата «дата-время»> задает точность округления значения типа «дата-время» (до какого элемента даты-времени должно выполняться округление).
4) Округление по году и месяцу имеет свою специфику, например,

```
select ROUND( sysdate, 'Y' );
```

возвращает вместе с годом и начальные день и месяц этого года:

```
|01.01.2005:00:00:00.00|
```

Для исключения начальных месяца и дня из «округляемого» года необходимо использовать конструкцию:

```
select to_char( ROUND( sysdate, 'Y' ), 'YYYY');
```

Аналогично для «округляемого» месяца:

```
select to_char( ROUND( sysdate, 'M' ), 'MM.YYYY');
```

5) Все позиции (после запятой), требуемые к заполнению символьным форматом второго аргумента и выходящие за пределы вычисленной точности, заполняются символом '0'.

Возвращаемое значение

1) <Выражение>, округленное до заданной точности.
2) Если <числовое выражение> не может быть округлено с заданной точностью перед десятичной точкой, то возвращается ноль (например, задано округление 45.67 до тысяч – round(45.67,-3)).
3) Тип возвращаемого результата: для <числового выражения> DOUBLE, для <выражение типа «дата-время»> DATE.
4) Результат при указании <элемента формата «дата-время»> 'D' зависит от наличия ключа /COMPATIBILITY=ORACLE в команде запуска ядра СУБД:
   - значение DATE, округленное до текущего дня, если ключ не задан;
   - значение DATE, округленное до ближайшего дня начала недели, если ключ задан.
5) Результат при указании <элемента формата «дата-время»> 'DY' или 'DAY' будет округлен до ближайшего дня начала недели.
6) Если аргумент NULL, результат NULL.

Примеры

```
select round(453.9847,0),
       round(453.9847,2),
       round(453.9847,7),
       round(-453.9847,3);

|454 |453.98 |453.9847 |-453.985 |

select round(453.9847,-1),
       round(453.9847,-2),
       round(453.9847,-3),
       round(456.9847,-1),
```

```
       round(-456.9847,-1);

|450 |500 |0 |460 |-460 |
select sysdate, round(sysdate, 'hh');

|29.08.2005:11:17:43.00|29.08.2005:11:00:00.00|
select sysdate, round(sysdate, 'mi');

|29.08.2005:11:17:43.00|29.08.2005:11:18:00.00|
select sysdate, round(sysdate, 'd');

|29.08.2005:11:17:43.00|29.08.2005:00:00:00.00|
select sysdate, round(sysdate, 'm');

|29.08.2005:11:17:43.00|01.09.2005:00:00:00.00|
select sysdate, round(sysdate, 'y');

|29.08.2005:11:17:43.00|01.01.2006:00:00:00.00|
```

Усечение представления значения (TRUNC)

Функция

Усечение значения с заданной точностью.

Спецификация

[1] <синтаксис>::= TRUNC (<выражение>[, <точность>])

Синтаксические правила

1) При усечении <числового выражения> округление не выполняется.
2) <Точность> задает точность усечения: при положительном значении усечение выполняется после десятичной точки (количество цифр после десятичной точки). Отрицательное значение – точность округления перед десятичной точкой, т.е. при -1 – до ближайшего целого десятка, -2 – до сотни, -3 до тысячи и т.д.



Примечание
При усечении до значения 1e-13 числа считаются равными.

3) <Элемент формата «дата-время»> задает точность усечения значения типа «дата-время» (до какого элемента даты-времени должно выполняться усечение).
4) Для <выражения типа «дата-время»> аргумент <точность> можно не указывать. В этом случае усечение выполняется до текущего дня даты.

```
select sysdate, trunc(sysdate,'d'), trunc(sysdate);

|17.08.2015:10:55:11.79|17.08.2015:00:00:00.00|
17.08.2015:00:00:00.00|
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

Возвращаемое значение

1) <Выражение>, усеченное до заданной точности.
2) Тип возвращаемого результата: для <числового выражения> DOUBLE, для <выражение типа «дата-время»> DATE.
3) Результат при указании <элемента формата «дата-время»> 'D' зависит от наличия ключа /COMPATIBILITY=ORACLE в команде запуска ядра СУБД:
   - значение DATE, усеченное до текущего дня, если ключ не задан;
   - значение DATE, усеченное до ближайшего дня начала недели, если ключ задан.
4) Результат при указании <элемента формата «дата-время»> 'DY' или 'DAY' будет усечен до ближайшего дня начала недели.
5) Если аргумент NULL, результат NULL.

Примеры

```
select round(6.785,1),
       trunc(6.785,1),
       round(-6.785,5),
       trunc(-6.785,0),
       abs(round((avg(salary) - max(salary)),2) -
 trunc((avg(salary) - max(salary)),0)),
       trunc(abs(round((avg(salary) - max(salary)),2) -
 trunc((avg(salary) - max(salary)),0)),5)
  from person;

|6.8 |6.7 |-6.785 |-6 |0.160000000003492 |0.16 |
```

```
select sysdate, round(sysdate, 'hh'),trunc(sysdate, 'hh');

|29.08.2005:11:38:27.00|29.08.2005:12:00:00.00|
29.08.2005:11:00:00.00|
```

```
select sysdate, round(sysdate, 'mi'),trunc(sysdate, 'mi');

|29.08.2005:11:38:27.00|29.08.2005:11:38:00.00|
29.08.2005:11:38:00.00|
```

```
select sysdate, round(sysdate, 'd'),trunc(sysdate, 'd');

|29.08.2005:11:38:27.00|29.08.2005:00:00:00.00|
29.08.2005:00:00:00.00|
```

```
select sysdate, round(sysdate, 'm'),trunc(sysdate, 'm');

|29.08.2005:11:38:27.00|01.09.2005:00:00:00.00|
01.08.2005:00:00:00.00|
```

```
select sysdate, round(sysdate, 'y'),trunc(sysdate, 'y');
```

```
|29.08.2005:11:38:27.00|01.01.2006:00:00:00.00|
01.01.2005:00:00:00.00|
```

Определение знака числа (SIGN)

Функция

Определение знака числа.

Спецификация

[1] <синтаксис>::= SIGN (<числовое выражение>)

Возвращаемое значение

1) -1 – <числовое выражение> отрицательное.
2) 1 – <числовое выражение> положительное.
3) 0 – <числовое выражение> равно нулю.
4) Тип возвращаемого результата – INT.

Вычисление квадратного корня (SQRT)

Функция

Вычисление квадратного корня числа.

Спецификация

[1] <синтаксис>::= SQRT (<числовое выражение>)

Возвращаемое значение

1) Корень квадратный из <числового выражения>.
2) Тип возвращаемого результата – DOUBLE.
3) Если аргумент NULL, результат NULL.

Псевдослучайное значение (RAND)

Функция

Получить псевдослучайное число.

Спецификация

[1] <синтаксис>::= RAND ([<числовое выражение>])

Синтаксические правила

1) <Числовое выражение> должно иметь тип данных REAL или приводиться к нему.
2) <Числовое выражение> задает создание последовательности псевдослучайных чисел, генерируемых на его основе.
3) Если <числовое выражение> положительное число, то оно и является основанием последовательности, если отрицательное – основанием последовательности является текущая дата.

© Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

Общие правила

1) Последовательность псевдослучайных значений инициализируется:
   - при первом выполнении функции RAND с аргументом;
   - при первом после запуска СУБД ЛИНТЕР выполнении функции RAND без аргумента. В этом случае за основание последовательности псевдослучайных значений принимается 1.
2) Для получения очередного псевдослучайного значения функция RAND должна вызываться без аргумента.
3) Функция RAND с одним и тем же положительным значением <числового выражения> будет всегда генерировать одинаковую последовательность псевдослучайных значений.

Возвращаемое значение

Псевдослучайное значение типа REAL в диапазоне от 0 до 1.

Пример

Инициирование последовательности и получение первого псевдослучайного значения из последовательности с данным основанием:

```
select rand(5);

|     0.430684101037068|
```

Получение псевдослучайного значения:

```
select rand();

|     0.430684101037068|
```

Инициирование последовательности и получение первого псевдослучайного значения из последовательности с другим основанием:

```
select rand(79);

|     0.404723714294249|
```

Получение псевдослучайного значения:

```
select rand();

|     0.528760920990613|
```

Воспроизведение псевдослучайных значений для последовательности с первоначальным основанием:

```
select rand(5);

|     0.430684101037068|
```

Получение псевдослучайного значения:

```
select rand();
```

```
|     0.0522177750487895|
```

Функции обработки списков

Под списком понимается конечный набор значений (например, из разных столбцов текущей строки).

Определение максимального значения в списке (GREATEST)

Функция

Определение максимального значения в заданном списке.

Спецификация

[1] <синтаксис>::=
GREATEST (<1-й элемент списка>[, <2-й элемент списка> [...]])
[2] <элемент списка>::= <значимое выражение>

Синтаксические правила

1) Типы данных всех элементов списка должны быть совместимы.
2) <Элементы списка> могут иметь числовой, символьный или «дата-время» тип данных.
3) Запрещается одновременное использование в <элементах списка> строковых значений с фиксированной и переменной длиной: либо все должны быть с фиксированной длиной, либо все – с переменной длиной.

```
Недопустимая конструкция:
select greatest( cast 'aaa' as varchar, cast 'abc' as char);

Допустимая конструкция:
select greatest( cast 'aaa' as varchar, cast 'abc' as varchar);
```

4) Все аргументы функции могут задаваться в виде <SQL-параметра>, который должен содержать спецификацию типа данных параметра.

```
select greatest(year, ? (int), ? (int)) from auto limit 1;
50
40

|       71|

select greatest (to_date('20.07.2015', 'dd.mm.yyyy'),
                 to_date('21.07.2015', 'dd.mm.yyyy'));

|21.07.2015:00:00:00|
```

Возвращаемое значение

1) Максимальное из значений в списке.

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

2) Тип возвращаемого результата – DOUBLE, если элементы списка имеют числовые типы данных.

3) Если список состоит из строковых (байтовых) значений, то длина результата равна максимальной из длин элементов списка.

Примеры

```
select count(*) from auto
 where make='FORD'
   and year=greatest(1, select max(year) from auto);

| 60|
```

Определение минимального значения в списке (LEAST)

Функция

Определение минимального значения в заданном списке.

Спецификация

[1] <синтаксис>::=
LEAST (<1-й элемент списка>[, <2-й элемент списка> [...]])

Синтаксические правила

1) Типы данных всех элементов списка должны быть совместимы.

2) <Элементы списка> могут иметь числовой, символьный или «дата-время» тип данных.

3) Запрещается одновременное использование в <элементах списка> строковых значений с фиксированной и переменной длиной: либо все должны быть с фиксированной длиной, либо все – с переменной длиной.

```
Недопустимая конструкция:
select least( cast 'aaa' as varchar, cast 'abc' as char);

Допустимая конструкция:
select least( cast 'aaa' as varchar, cast 'abc' as varchar);
```

4) Все аргументы функции могут задаваться в виде <SQL-параметра>, который должен содержать спецификацию типа данных параметра.

```
select least(year, ? (int), ? (int)) from auto limit 1;
50
40

|       40|
```

Возвращаемое значение

1) Минимальное из значений в списке.

2) Тип возвращаемого результата – DOUBLE, если элементы списка имеют числовые типы.

3) Если список состоит из строковых (байтовых) значений, то длина результата равна максимальной из длин элементов списка.

Пример

```
select greatest(count(*), min(salary), avg(salary), max(salary)),
  least(count(*), min(salary), avg(salary), max(salary))
  from person;

|96000 |986 |
```

```
select age,salary, greatest(age,salary/1000) from person;

 AGE SALARY
|30 |10800 |30 |
|48 |51000 |51 |
|52 |37000 |52 |
|41 |11000 |41 |
|32 |51000 |51 |
|34 |36000 |36 |
 …
```

Функции преобразования

Преобразование символьной шестнадцатеричной строки в строку байт (HEXTORAW/HEX)

Функция

Преобразование символьной шестнадцатеричной строки в строку байт.

Спецификация

[1] <синтаксис>::= {HEXTORAW | HEX}(<символьное выражение>)

Синтаксические правила

1) <Символьное выражение> – строка, содержащая символьное представление шестнадцатеричных цифр (цифры 0-9, буквы A-F).
2) Длина <символьного выражения> должна быть кратна 2.
3) Аргумент функции HEXTORAW может быть задан <SQL-параметром>, который должен содержать спецификацию типа данных параметра.

```
select hextoraw(? (char(10))), hextoraw(:param (char(6)));
```

4) Аргумент функции HEX может быть задан символьной строкой, выражением символьного типа, содержащим конкатенацию, функцию, возвращающую символьное значение и т.д.

Возвращаемое значение

Байтовая строка длиной N, если исходное <символьное выражение> имело длину 2*N.

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

Примеры

```
select cast hextoraw('de56df36') as char;

|"V¬6 |
```

```
select 'acd0554f',
       length('acd0554f'),
       hextoraw('acd0554f'),
       length(hextoraw('acd0554f'));

|acd0554f|          8| AC D0 55 4F|          4|
```

Преобразование значения в шестнадцатеричное представление (RAWTOHEX)

Функция

Преобразование значения в символьное шестнадцатеричное представление.

Спецификация

[1] <синтаксис>::= RAWTOHEX (<значимое выражение>)

Синтаксические правила

1) <Значимое выражение> – значение любого допустимого типа данных.
2) Аргумент функции может быть задан <SQL-параметром>, который должен содержать спецификацию типа данных параметра.

```
select rawtohex(? (boolean));
true

|01|
```

Возвращаемое значение

Символьная строка (тип CHAR) длиной 2*N, если исходное <значимое выражение> имело длину N.

Пример

```
select rawtohex('de56df36'), rawtohex((34+256)/4.2);

|6465353664663336 |024C04134C044500000000000000000 |
```

Преобразование значимого выражения в символьное представление (TO_CHAR)

Функция

Преобразование числового значения или значения типа «дата» из внутреннего представления в символьное.

Спецификация

[1] <синтаксис>::=
TO_CHAR (<значимое выражение>[, <символьный формат>[, <NLS-спецификация>]])
[2] <значимое выражение>::=
<числовое выражение> | <выражение типа «дата-время»>
[3] <NLS-спецификация>::=
<NLS_NUMERIC_CHARACTERS> | <NLS_DATE_LANGUAGE>
[4] <NLS_NUMERIC_CHARACTERS>::=
'NLS_NUMERIC_CHARACTERS=<символьный литерал>'
[5] <NLS_DATE_LANGUAGE>::=
'NLS_DATE_LANGUAGE={RUSSIAN | ENGLISH}'
[6] <символьный формат>::= <символьный литерал>

Синтаксические правила

1) <Числовое выражение> должно иметь значение типа SMALLINT, INTEGER, BIGINT, DECIMAL, REAL или DOUBLE.
2) При необходимости производится преобразование числового аргумента к типу данных DECIMAL.
3) При первом параметре функции to_char типа DATE при работе в режиме /COMPATIBILITY=ORACLE посторонние алфавитно-цифровые символы в строке формата при форматном выводе значений даты и времени вызывают выдачу кода завершения. При работе в прочих режимах /COMPATIBILITY эти символы копируются в выходную строку, без выдачи кода завершения.

Примечание

Ключ /COMPATIBILITY=ORACLE поддерживается со сборки 6.0.17.92.

4) <Выражение типа «дата-время»> должно иметь значение типа DATE.
5) Аргументы функции могут быть заданы <SQL-параметром>.

```
select to_char(? (int)), to_char(:param (date), ?);
+451
12/02/2015
MM/DD/YYYY

|451         |12/02/2015|
```

6) Для числовых значений <символьный формат> может содержать следующие элементы:
   - S – позиция для знака, должна быть одна в начале или в конце, выводятся и плюс и минус. При указании модификатора знак + выводится, в противном случае – нет. Для отрицательных чисел знак числа выводится всегда, независимо от присутствия или отсутствия в шаблоне модификатора S, для целых чисел нули после точки не формируются. Если в формате для строкового преобразования числа нет позиции для знака S, то в режиме /COMPATIBILITY=STANDARD автоматически добавляется одна позиция в длину результата, а в обычном режиме не добавляется (если знак из-за этого не помещается, выдается признак ошибки - заполнение символами '*');
   - 9 – позиция для цифры. Ведущие нули не выводятся за исключением числа 0;
   - 0 – позиция для цифры с обязательным выводом ведущих или концевых нулей;

© Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

- `.` – позиция для десятичной точки (должна быть одна);
- `D` – позиция для десятичной точки или ее заменителя, который берется из параметра `NLS_NUMERIC_CHARACTER`;
- `,` – позиция для разделителя-запятой, не может быть первой или идти после десятичной точки;
- `G` – позиция для разделителя-запятой или ее заменителя, который берется из параметра `NLS_NUMERIC_CHARACTER`;
- `FM` – признак «плотного вывода». Усекаются ведущие пробелы и концевые нули, т.е. вывод выравнивается по левой границе (по умолчанию это не делается и вывод в режиме /COMPATIBILITY=ORACLE выравнивается по правой границе);
- `EEEE` – представление числа с мантиссой и порядком. Для данного формата не поддерживаются символы `G` и `D`, а также третий аргумент `NLS_NUMERIC_CHARACTERS`.

```
select to_char(415000035);

|415000035|

select TO_CHAR (415000035,'999999999');

|   415000035|

select TO_CHAR (41500.0035,'999,999.999');

|  41,500.004|

select TO_CHAR (415000035,'S00999999999');

|  +00415000035|

select TO_CHAR (round(415000035,6),'9.99999999EEEE');

|4.15000035E+008 |

select to_char(1.0e0/3,'9.9999999999EEEE');

|3.3333333333E-001 |

SELECT TO_CHAR(415000035, 'FM99999999990D9999999'),
       TO_CHAR(415000035, '99999999990D9999999');

|415000035.0000000  |415000035.0000000  |
```

7) Форматы по умолчанию числовых типов:

Тип данных	**Формат по умолчанию**
SMALLINT	99999

Тип данных	Формат по умолчанию
INT	9999999999
BIGINT	9999999999999999999
REAL	99999999.99 (округление)
DOUBLE	99999999.99 (округление)
DECIMAL	99999999999999999999.9999999999

8) <Символьный литерал> в опции NLS_NUMERIC_CHARACTERS задается в виде двух символов 'XY', где:
   - X – определяет символ, используемый для различения целой и дробной части числового значения (по умолчанию точка);
   - Y – определяет разделитель групп цифр числового значения (по умолчанию запятая).

```
select TO_CHAR (9223372036857, '9999999G999G999D999999',
  'NLS_NUMERIC_CHARACTERS=.,');

| 9223372,036,857.000000|

select TO_CHAR (9223372036857, '9999999G999G999D999999',
  'NLS_NUMERIC_CHARACTERS=:;');

| 9223372:036:857;000000|

select TO_CHAR (-1231.5, 'S9G999D9999',
  'NLS_NUMERIC_CHARACTERS=''|?''');

| -1?231|5000|
```

9) <Символьный формат> для <выражения типа «дата-время»> может содержать следующие элементы:
   - YYYY | IYYY – номер года (4 цифры);
   - YY | IY – номер года (2 последние цифры, диапазон 00-99);
   - Q – квартал года (1 для января-марта, 2 для апреля-июня, 3 для июля-сентября, 4 для октября-декабря);
   - MM – номер месяца в году (2 цифры, диапазон 01-12);
   - MON – сокращенное название месяца (3 буквы);
   - MONTH – полное название месяца (8 букв);
   - RM – вывод номера месяца римскими цифрами (I-XII);
   - J – номер дня по юлианскому календарю;
   - D – номер дня в неделе (1 цифра, диапазон 1-7);
   - DD – номер дня в месяце (2 цифры, диапазон 01-31);
   - DY – сокращенное название дня недели (3 буквы);
   - DAY – полное название дня недели (12 букв);

- DDD – номер дня в году (3 цифры, диапазон 001-366);
- HH12 – количество часов (2 цифры, диапазон 00-12), обычно комбинируется вместе с одним из форматов am, pm, AM, PM;
- am, pm, AM, PM, a.m., p.m., A.M., P.M. – время до полудня (после полудня). A.M. автоматически заменяется на P.M. (и наоборот), в зависимости от заданного времени;
- HH, HH24 – количество часов (2 цифры, диапазон 00-23);
- MI – количество минут (2 цифры, диапазон 00-59);
- SS – количество секунд (2 цифры, диапазон 00-59);
- SSSSS – число секунд, прошедших от 00:00:00 текущих суток;
- FF – количество тиков (2 цифры, диапазон 00-99);
- FFF или MS – количество тысячных долей секунды (3 цифры) – только при использовании в функции TO_DATE, делится на 10 для перевода в тики.
- FM – признак «плотного вывода» наименований дней недели и месяцев года (то есть усекаются пробелы в конце имен). Применяется в виде префикса к шаблону полного дня недели или полного месяца года, т.е. FMDAY или FMMONTH.

```
select to_char(to_date('21.03.2016', 'dd.mm.yyyy'), 'month',
  'NLS_DATE_LANGUAGE=RUSSIAN' ) + '- месяц года.';

|март     - месяц года.|

select to_char(to_date('21.03.2016', 'dd.mm.yyyy'), 'fmmonth',
  'NLS_DATE_LANGUAGE=RUSSIAN' ) + '- месяц года.';

|март - месяц года.|
```

Примечание
Представление значений элементов формата нельзя усекать, т.е. номер месяца 2 надо записывать в виде 02.

10) Запрещено использование в <символьном формате> для <выражения типа «дата-время»> цифр и латинских букв, которые не задают разрешенный формат вывода одного из компонентов даты.
11) Входным форматом «дата-время» по умолчанию, который распознается автоматически (т.е. без указания формата преобразования), являются:
    - DD.MM.[YY]YY[:[HH:[MI:[SS.[FF]]]]];
    - MM/DD/[YY]YY[:[HH:[MI:[SS.[FF]]]]];
    - DD-MON-[YY]YY[:[HH:[MI:[SS.[FF]]]]];
    - YYYY-MM-DD[:[HH:[MI:[SS.[FF]]]]].

    Если символьные значения типа DATE представляются в ином виде, необходимо явно задавать формат преобразования.

```
create table tab1(dt date);
```

```
insert into tab1 values(sysdate);
insert into tab1 values('28-apr-50');
insert into tab1 values('28.04.51');
insert into tab1 values('04/28/52');
insert into tab1 values('28-apr-1953');
insert into tab1 values(to_date('28|04|1954','dd|mm|yyyy'));

select * from tab1;

|30.04.2003:12:03:27.00 |
|28.04.1950:00:00:00.00 |
|28.04.1951:00:00:00.00 |
|28.04.1952:00:00:00.00 |
|28.04.1953:00:00:00.00 |
|28.04.1954:00:00:00.00 |

insert into tab1 values(to_date('23.10.1997:02:27:38:78',
  'dd.mm.yyyy:hh:ss:mi:ff'));
insert into tab1 values(to_date('23.10.1997:02:27:38:178',
  'dd.mm.yyyy:hh:ss:mi:ms'));

select to_char(dt, 'dd.mm.yyyy:hh:mi:ss:fff') from tab1;

…
|23.10.1997:02:38:27:780|
|23.10.1997:02:38:27:170|
```

12) Выходные символьные строки значений типа `DATE` по умолчанию (т.е. при отсутствии явно заданного формата преобразования) представляются всегда в виде DD.MM.YYYY:HH:MI:SS.FF.

```
select sysdate;

|30.04.2003:12:03:27.25|
```

13) При конструировании формата преобразования допустимо использовать обозначения элементов формата, перечисленные выше, в любых комбинациях (кроме ограничений, накладываемых функцией `TO_DATE`) и задавать любые символы и/или строки символов, не совпадающие с обозначениями элементов формата, в качестве разделителей элементов формата. При распознавании формата регистр букв не различается, т.е. обозначения вида DD, dd, Dd и dD равнозначны. При использовании формата преобразуемое входное символьное значение должно в точности соответствовать формату:

Символьное представление	**Формат**
16.02.1998:10:35	DD.MM.YYYY:HH:MI
16.02.98:10:35	dd.mm.yy:HH:MI
16-02-98 10:35	DD-mm-YY hh:mi
10 час 30 мин 16/02/1998	HH час MI мин DD/MM/YYYY
В год 2000 месяца 02	В год YYYY месяца MM

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

Символьное представление	Формат
дня 16 10.35.47	дня DD hh.mi.ss
13.45.37	hh12.mi.ss a.m.

14) При вводе года по формату YY к нему добавляется 2000, если значение меньше или равно 37, иначе – 1900. При выводе выдаются две последние цифры без каких-либо проверок.

15) Третьим параметром для <выражения типа «дата-время»> является <NLS_DATE_LANGUAGE>. Значение третьего параметра определяет язык вывода названия месяцев и дней недели. Значение регистронезависимое, по умолчанию ENGLISH.

```
select to_char(sysdate, 'dd mon iyyy day'),
  to_char(sysdate, 'dd mon iyyy day',
    'NLS_DATE_LANGUAGE=russian'),
  to_char(sysdate, 'dd mon iyyy day',
    'NLS_DATE_LANGUAGE=ENGLISH');

|15 nov 2016 tuesday|15 ноя 2016 вторник|15 nov 2016 tuesday|
```

Возвращаемое значение

1) Символьное представление <значимого выражения> осуществляется в соответствии с <символьным форматом> преобразования.
2) Если формат представления недостаточен, выводится последовательность в виде *****.
3) Формат вывода выражений CAST … AS CHAR для выражения типа DECIMAL (если количество значащих цифр после запятой меньше точности) зависит от ключа /COMPATIBILITY=STANDARD запуска ядра СУБД:
   - ключ задан – значение после запятой будет дополнено нулями до значения точности;
   - ключ не задан – значение после запятой не будет дополнено нулями до значения точности.

```
Примеры.
1) Ядро СУБД запущено с ключом /COMPATIBILITY=STANDARD:
select cast (cast 1.5 as DECIMAL(15,3)) as char;

|1.500 |

2) Ядро СУБД запущено без ключа /COMPATIBILITY=STANDARD:
select cast (cast 1.5 as DECIMAL(15,3)) as char;

|1.5 |
```

Примеры

Пусть значение столбца dat будет 2 часа 27 минут 38 секунд 78 тиков 23 октября 1997, тогда:

```
to_char(dat) /* по умолчанию*/
```

```
|23-oct-97 |

to_char(dat,'dd-mm-yyyy/hh24:mi:ss')

|23-10-1997/02:27:38 |

to_char(dat,'Дата отчета: dd/mm/yy')

|Дата отчета: 23/10/97 |

to_char(dat,'hh24.mi')

|02.27 |

select sysdate, to_char(sysdate, 'hh12.mi.ss  a.m.');

|06.06.2003:14:07:14.00 |02.07.14  p.m. |
```

Преобразование символьного значения во внутреннее представление (TO_NUMBER)

Функция

Преобразование символьного числового значения во внутреннее представление.

Спецификация

[1] <синтаксис>::= TO_NUMBER (<символьное выражение>)

Синтаксические правила

1) <Символьное выражение> должно задавать символьное представление числового значения любого допустимого типа.
2) Аргумент функции может быть задан <SQL-параметром>, который должен содержать спецификацию типа данных параметра.

```
select to_number(? (char(5)));
+567.54

|                  567.54|
```

Возвращаемое значение

1) Числовое значение <символьного выражения> во внутреннем представлении.
2) Тип возвращаемого значения:
   - DOUBLE: если ядро запущено без ключа /COMPATIBILITY=ORACLE;
   - DECIMAL: если ядро запущено с ключом /COMPATIBILITY=ORACLE.
3) При ошибке преобразования возвращается код завершения 1042 («Ошибка при преобразовании строки в вещественное число»), который при необходимости можно игнорировать с помощью функции NULLIFERROR.

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

Примечание
Ключ /COMPATIBILITY=ORACLE поддерживается со сборки 6.0.17.92.

Примеры

```
create table tab1 ("День" int, "Месяц" int, "Год" int);
insert into tab1 values (cast to_number(to_char(sysdate,'dd'))
 as int, cast to_number(to_char(sysdate,'mm')) as int, cast
 to_number(to_char(sysdate,'yyyy')) as int);

select * from tab1;

|15 |4 |2003 |
select to_number('-45'+'.666');

|               -45.666|
```

Преобразование символьного представления типа «дата-время» во внутреннее (TO_DATE/TO_TIMESTAMP)

Функция

Преобразование символьного представления значения типа «дата-время» во внутреннее.

Спецификация

[1] <синтаксис>::=
TO_DATE | TO_TIMESTAMP (<символьное выражение>[, <символьный формат>])

Синтаксические правила

1) TO_TIMESTAMP является синонимом TO_DATE.
2) <Символьное выражение> должно задавать символьное значение типа «дата-время».
3) <Символьное выражение> и <символьный формат> могут быть заданы <SQL-параметром>, который должен содержать спецификацию типа данных параметра и его маску соответственно.

```
select to_date(:param_value (char(10)), :param_mask);
28-05-2030
dd-mm-yyyy

|28.05.2030:00:00:00.00|
```

4) Описание элементов <символьного формата> см. в описании функции TO_CHAR.
5) Если <символьный формат> не задан, по умолчанию используется DD-MON-YY.

```
select to_date('28-apr-03'), to_date('28-apr-45');
```

```
|28.04.2003:00:00:00.00 |28.04.1945:00:00:00.00 |
```

Возвращаемое значение

1) <Символьное выражение> во внутреннем представлении.
2) Тип возвращаемого значения – DATE.

Пример

```
create table tab1(d date);
insert into tab1 values (to_date('28-05-2003', 'dd-mm-yyyy'));
insert into tab1 values (to_date('28-apr-03'));
```

Преобразование значения в XML-формат (XML)

Функция

Преобразование значения в XML-формат.

Спецификация

[1] <синтаксис>::= XML (<элемент>[, <элемент> ...])
[2] <элемент>::= <идентификатор столбца> |<символьный литерал>

Возвращаемое значение

1) Значение <элемента> в XML-формате, т.е. в обрамлении тега, имя которого совпадает, по возможности, с <идентификатором столбца>.
2) Тип возвращаемого значения – VARCHAR(n), где n – сумма длин тегов и максимальных длин текстовых преобразований полей.

```
select xml(make, model, year) from auto;

|<MAKE>FORD</MAKE>            |<MODEL>MERCURY COMET GT V8</MODEL>|
<YEAR>71</YEAR>|
|<MAKE>ALPINE</MAKE>          |<MODEL>A-310</MODEL>              |
<YEAR>70</YEAR>|
|<MAKE>AMERICAN MOTORS</MAKE>|<MODEL>MATADOR STATION</MODEL>    |
<YEAR>71</YEAR>|
...
```

3) Если аргументы функции содержит литералы, выражения, повторяющиеся имена столбцов либо столбцы с не алфавитно-цифровыми именами, то вместо проблемных имен выводятся имена в формате «COLUMN_NUMBER_nnn».

```
select xml(sysdate, to_char(34.7,'99.9'));

|<COLUMN_NUMBER_1>20.06.2005:10:57:54:00</COLUMN_NUMBER_1>|
<COLUMN_NUMBER_2>34.7</COLUMN_NUMBER_2>|
```

4) Если аргумент функции имеет NULL-значение, то для него возвращаемое значение не формируется.

```
select xml(make, null, make) from auto fetch first 1;
```

```
|<MAKE>FORD</MAKE>|<COLUMN_NUMBER_3>FORD</COLUMN_NUMBER_3>|
```

5) <Элемент> может быть задан <SQL-параметром>, который должен содержать спецификацию типа данных параметра.

```
select xml(? (char(20)), ?(char(20)))from auto limit 1;
make
year

|<COLUMN_NUMBER_1>make</COLUMN_NUMBER_1>
<COLUMN_NUMBER_2>year</COLUMN_NUMBER_2>|
```

Преобразование значения в HTML-формат (HTML)

Функция

Преобразование значения в HTML-формат.

Спецификация

[1] <синтаксис>::= HTML (<элемент>[, <элемент> ...])

Возвращаемое значение

1) <Значение> в HTML-формате.
2) Тип возвращаемого значения – VARCHAR(4000).

```
select html(make, model, year) from auto limit 1;

|<tr><td>FORD</td><td>MERCURY COMET GT V8</td><td>71</td></tr>|
```

```
select html(sysdate, to_char(34.7,'99.9'));
<tr><td>20.06.2005:11:21:13:00</td><td>34.7</td></tr>
```

3) NULL-значения выделяются курсивом (тег <I>).

```
select html(make, null, make) from auto fetch first 1;

<tr><td>FORD</td><td><i>NULL</i></td><td>FORD</td></tr>
```

4) Аргументы функции могут быть заданы <SQL-параметром>, который должен содержать спецификацию типа данных параметра.

```
select html(? (char(20))) from auto where year=:param limit 1;
model
71

|<tr><td>model</td></tr>                |
```

Преобразование из одной кодировки в другую (CONVERT)

Функция

Преобразование символьной (байтовой) строки в соответствие с заданной кодовой страницей.

Спецификация

[1] <синтаксис>::=
CONVERT (<строка>, <выходная кодировка>, [<входная кодировка>])
[2] <строка>::= <значимое выражение>
[3] <выходная кодировка>::= <символьный литерал>
[4] <входная кодировка>::= <символьный литерал>

Синтаксические правила

1) Тип данных <значимого выражения> <строки> должен быть CHAR, VARCHAR, NCHAR, NVARCHAR, BYTE, VARBYTE.
2) <Выходная кодировка> и <входная кодировка> – символьные литералы, задающие имена кодовых страниц. Не допускается использование параметров, выражений и имен столбцов.
3) Если <входная кодировка> не задана, используется значение по умолчанию: текущая кодировка <строки> (обычно совпадает с кодировкой канала) для значений строковых типов и кодировка системного словаря – для значений байтовых типов.
4) Если <выходная кодировка> и <входная кодировка> не заданы, то перекодировка выполняется из кодировки системного словаря СУБД в кодировку канала.

```
create or replace table
  "ZZZТаблица с большим именемXXX" (i int, c char(10));
create or replace public synonym s_tab_2
  for "ZZZТаблица с большим именемXXX";
insert into
  "ZZZТаблица с большим именемXXX" values (1,'DEFAULT');
select * from "ZZZТаблица с большим именемXXX";
select * from s_tab_2;

Если теперь в командном интерфейсе выполнить команду
show S_TAB_2
то имя синонима будет показано как
PUBLIC.SYNONIM FOR "SYSTEM"." ZZZ????????? ? ??????? ??????XXX"

Чтобы получить корректное имя синонима, можно использовать функцию
 CONVERT:
select convert(getraw($$$S14,30,66))
  from LINTER_SYSTEM_USER.$$$SYSRL
 where $$$S13 like 'S_TAB_2'
   and $$$S12 = -1;

|ZZZТаблица с большим именемXXX|
```

5) Доступны все кодировки, включенные в системную таблицу $$$CHARSET и дополнительно Unicode-кодировки с именами UTF-8 и UCS2.
6) Аргумент <строка> может быть задан <SQL-параметром>, который должен содержать спецификацию типа данных параметра.

```
select convert(? (char(20)), 'CP866', 'CP1251');
яЁштхЄ
```


© Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
|привет                |
```

Общие правила

1) При посылке данных клиентскому приложению перекодированная строка автоматически преобразуется в кодировку клиентского приложения, поэтому запрос, извлекающий результат работы функции CONVERT, возвращает значения, перекодированные в кодировку клиентского приложения.

```
select convert('яЁштхЄ', 'CP1251', 'CP1251'),
       convert('привет', 'CP1251', 'CP866'),

 convert('&#9576;&#9573;&#9556;&#9579;&#9532;&#9560;','CP1251','KOI8-
R'),
       convert(cast Hex('3F044004380432043504420') as
 nchar,'CP1251','UCS2'),
       convert(Hex('EFF0E8E2E5F2'),'CP1251','CP1251');

|привет|привет|привет|привет|привет|
```

2) Если требуется получить строку в кодировке, отличной от кодировки клиента, следует воспользоваться приведением к байтовому типу:

```
select cast convert('привет', 'CP866', 'CP866') as byte,'*',
       cast convert('привет', 'CP1251', 'CP866') as byte,'*',
       cast convert('привет', 'KOI8-R', 'CP866') as byte,'*',
       cast convert('привет', 'UCS2', 'CP866') as byte,'*';

| AF E0 A8 A2 A5 E2|*| EF F0 E8 E2 E5 F2|*| D0 D2 C9 D7 C5 D4|*|
 3F 04 40 04 38 04 32 04 35 04 42 04|*|
```

Т.е. несмотря на то, что все извлекаемые значения имеют одно и то же строковое представление, бинарное представление у них разное.

Возвращаемое значение

1) Если <выходная кодировка> не является Unicode-кодировкой, функция возвращает строку типа VARCHAR.
2) Если <выходная кодировка> является Unicode-кодировкой, функция возвращает строку типа NVARCHAR.

Примеры

> **Примечание**
> Примеры выполнены с помощью утилиты inl в среде ОС Windows с кодировкой клиента CP866.

Структура-описатель значения <строки> содержит информацию о кодировке.

1)

```
select convert('яЁштхЄ', 'CP1251', 'CP1251'),
```

```
       convert('привет', 'CP1251', 'CP866'),
       convert('&#9576;&#9573;&#9556;&#9579;&#9532;&#9560;',
 'CP1251', 'KOI8-R'),
       convert(cast Hex('3F044004380432043504') as nchar,
 'CP1251', 'UCS2'),
       convert(Hex('EFF0E8E2E5F2'), 'CP1251', 'CP1251');

|привет|привет|&#9576;&#9573;&#9556;&#9579;&#9532;&#9560;|привет|
привет|
```

2)

```
select convert('a'||'b', 'CP1251', 'UCS2');

|?|
```

Функции бесформатного преобразования (GETBYTE/GETBYTEB)

Значение заданного байта

Функция

Получить значение заданного байта.

Спецификация

Варианты:

[1] <синтаксис>::= GETBYTE (<значимое выражение>, <смещение байта>)

[1] <синтаксис>::= GETBYTEB (<значимое выражение>, <смещение байта>)

[1] <смещение байта>::= целое положительное число

Синтаксические правила

1) <Смещение байта> – целое положительное число, задающее положение байта в <значимом выражении>. Нумерация байтов начинается с нуля.

2) Допустимые значения <смещения байта>:

Тип аргумента <значимое выражение>	Допустимый диапазон байтов <смещение байта>
CHAR(N)	0 – N-1
VARCHAR(N)	0 – N-1
BYTE(N)	0 – N-1
NCHAR(N)	0 – N-1
VARBYTE(N)	0 – N-1
NCHAR VARYING(N)	0 – N-1
DECIMAL (NUMERIC)	0 –15
BIGINT	0 – 7

© Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

Тип аргумента <значимое выражение>	Допустимый диапазон байтов <смещение байта>
INT	0 – 3
SMALLINT	0 – 1
REAL	0 – 3 (зависит от архитектуры)
DOUBLE	0 – 7 (зависит от архитектуры)
DATE	0 – 15
TRUE (FALSE)	0
BLOB	0 – 23 (значения из описателя BLOB-данных)

3) Тип <значимого выражения> не проверяется.
4) Аргументы <значимое выражение> и <смещение байта> могут быть заданы <SQL-параметром>, который должен содержать спецификацию типа данных параметра.

```
select getbyte(? (bigint),? (int));
56334289754
2

|        201|
```

Возвращаемое значение

1) Значение указанного байта в <значимом выражении>.
2) Тип возвращаемого значения: GETBYTE – INT, GETBYTEB – BYTE(1).
3) Если задано недопустимое смещение, фиксируется исключительная ситуация.

Пример

```
select getbyte('a',0),getbyte('A',0),getbyte(hex('FC0A'),1),
       getbyte(4567,1),
       getbyte(to_char(sysdate,'dd/mm/yyyy'),5)
  from person where personid=2;

|97 |65 |10 |17 |47 |
```

Значение заданного слова (GETWORD/GETWORDB)

Функция

Получить значение заданного слова.

Спецификация

Варианты:

1) [1] <синтаксис>::=
GETWORD (<значимое выражение>, <смещение слова>)
2) [1] <синтаксис>::=
GETWORDB (<значимое выражение>, <смещение слова>)

[1] <смещение слова>::= целое положительное число

Синтаксические правила

1) <Смещение слова> – целое положительное число, задающее положение слова в <значимом выражении>. Смещение начинается с нуля и отсчитывается в байтах.
2) Тип <значимого выражения> не проверяется.
3) Аргументы <значимое выражение> и <смещение слова> могут быть заданы <SQL-параметром>, который должен содержать спецификацию типа данных параметра.

```
select getword( ? (bigint), ? (int)), getwordb( ? (bigint), ?
 (int));
563342897540076
2
563342897540076
2

|      32075| 4B 7D|
```

Возвращаемое значение

1) Значение указанного слова (двух последовательных байт) в <значимом выражении>.
2) Тип возвращаемого значения: GETWORD – INT, GETWORDB – BYTE(2).
3) Если аргумент NULL, возвращается NULL.
4) Если задано недопустимое смещение, фиксируется исключительная ситуация.

Пример

Столбец $$$S14 первой строки системной таблицы $$$SYSRL содержит структуру, описывающую параметры настройки СУБД. Необходимо узнать максимальное количество одновременных подсоединений к БД (значение находится в структуре со смещением 24 байта):

```
select getword($$$s14,24) from $$$sysrl where rowid=1;

|100 |
```

Значение заданного длинного слова (GETLONG)

Функция

Получить значение заданного длинного слова.

Спецификация

[1] <синтаксис>::=
GETLONG (<значимое выражение>, <смещение слова>)

Синтаксические правила

1) <Смещение слова> – целое положительное число, задающее положение слова в <значимом выражении>. Смещение начинается с нуля и отсчитывается в байтах.

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

2) Тип <значимого выражения> не проверяется.

3) Аргументы <значимое выражение> и <смещение слова> могут быть заданы <SQL-параметром>, который должен содержать спецификацию типа данных параметра.

```
select getlong(? (bigint),? (int));
563342897540076
2


|    5995851|
```

Возвращаемое значение

1) Значение указанного длинного слова (четырех последовательных байт) в <значимом выражении>.

2) Тип возвращаемого значения – INT.

3) Если аргумент NULL, возвращается NULL.

4) Если задано недопустимое смещение, фиксируется исключительная ситуация.

Пример

Столбец $$$S14 первой строки системной таблицы $$$SYSRL содержит структуру, описывающую параметры настройки СУБД. Необходимо узнать логическое имя устройства, на котором расположен файл системного журнала (значение находится в структуре со смещением 50 байт).

```
select getstr(getlong($$$s14,50),0,4) from $$$sysrl where rowid=1;

|SY00 |
```

Значение заданного количества бит (GETBITS)

Функция

Получить значение заданного количества бит.

Спецификация

[1] <синтаксис>::=
GETBITS (<значимое выражение>, <смещение байта>, <смещение бита>, <количество бит>)
[2] <смещение байта>::= целое неотрицательное число
[3] <смещение бита>::= целое неотрицательное число
[4] <количество бит>::= целое положительное число

Синтаксические правила

1) <Смещение байта> – целое неотрицательное число, задающее положение того байта в <значимом выражении>, в котором находится начало требуемой битовой последовательности. Смещение начинается с нуля.

2) <Смещение бита> – целое неотрицательное число, задающее начало битовой последовательности относительно выбранного байта. Смещение начинается с нуля и задается в количестве бит. <Смещение бита> – значение из диапазона 0-7.

3) <Количество бит> – целое положительное значение в диапазоне от 1 до 32, задающее количество выбираемых бит.

4) Тип <значимого выражения> не проверяется.

5) Все аргументы функции могут быть заданы <SQL-параметром>, который должен содержать спецификацию типа данных параметра.

```
select GETBITS (? (byte(10)), :param (int), ? (smallint), ?
 (int));
0145ac77ff
2
3
8

|        245|
```

Возвращаемое значение

1) Значение указанной битовой последовательности. Тип возвращаемого значения – INT.

2) Если аргумент NULL, возвращается NULL.

3) Если задано недопустимое смещение, фиксируется исключительная ситуация.

Пример

```
GETBITS (hex('01450affcd02'), 2, 3, 8)
```

Результатом будет строка бит: 01010111(см. пояснение ниже).

Исходное значение	00000001	01000101	00001010	11111111	11001101	00000010
Смещение байта	0	1	2	3	4	5
Смещение бита			0123			
Результат	00000001	01000101	00001010	11111111	11001101	00000010

Столбец $$$S14 первой строки системной таблицы $$$SYSRL содержит структуру, описывающую параметры настройки СУБД. Необходимо узнать версию БД (значение находится в структуре со смещением 65 байт).

```
select getbits($$$s14,65,0,7) from $$$sysrl where rowid=1;
```

Выбор подмножества символов (GETSTR)

Функция

Выбор подмножества символов из любого допустимого значения.

Спецификация

[1] <синтаксис>::=
GETSTR (<значимое выражение>, <смещение>, <количество>)

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

[2] <смещение>::= <числовое выражение>

Синтаксические правила

1) Тип <значимого выражения> может быть произвольным.
2) <Смещение> задает начальную позицию выбираемого подмножества символов. Отсчет позиций начинается с нуля. <Числовое выражение>, задающее <смещение>, должно быть положительным целым числом.
3) <Количество> – целое положительное значение, задающее количество выбираемых символов.
4) Аргументы <значимое выражение> и <смещение> могут быть заданы <SQL-параметром>, который должен содержать спецификацию типа данных параметра.

```
select getstr(? (char(20)), :param (int), 2) from auto limit 1;
FORD
1

|OR|
```

Возвращаемое значение

1) Выбранная подстрока символов <значимого выражения>.
2) Тип возвращаемого значения – CHAR.
3) Если выбранный байт не может быть представлен в символьном виде, выводится пробел.
4) Если заданное <количество> превышает текущую размерность типа данных аргумента <значимого выражения>, фиксируется исключительная ситуация.

Примеры

```
select getstr(model, 0,20), getstr(sysdate, 0,4),
       getstr(user, 0,18), getstr(personid+100, 0,4) from auto;

|MERCURY COMET GT V8 | | |SYSTEM |e |
…
select getstr(model, 3, 10), getstr(sysdate, 0, 2),
  getstr(user, 4,6), getstr(personid+100, 2,1) from auto;

|CURY COMET GT V8 | | |EM | |
…
```

Выбор подмножества байт (GETRAW)

Функция

Выбор подмножества байт из любого допустимого значения.

Спецификация

[1] <синтаксис>::=
GETRAW (<значимое выражение>, <смещение>, <количество>)

[2] <количество>::= <числовой литерал>

Синтаксические правила

1) Тип <значимого выражения> может быть произвольным.
2) <Смещение> – целое положительное число, задающее начало выбираемого подмножества байт в <значимом выражении>. Смещение начинается с нуля.
3) <Количество> – целое положительное значение, задающее количество выбираемых байт.
4) Аргументы <значимое выражение> и <смещение> могут быть заданы <SQL-параметром>, который должен содержать спецификацию типа данных параметра.

```
select getraw(? (byte(10)), ? (int), 2);
aa0056fc
1
```

Возвращаемое значение

1) Выбранная подстрока байт <значимого выражения>.
2) Тип возвращаемого значения – BYTE.
3) Если для выбора задано больше байт, чем допускает <значимое выражение>, то лишние байты будут представлены двоичными нулями.

Примеры

```
create table tab1 (b varbyte(10));
insert into tab1 values (hex('aa0056fc'));

select getraw(b, 1,2) from tab1;

|0056 |

update tab1 set b= getraw(user, 1,10);
select getraw(b, 1,2) from tab1;

|5354 |

select cast getraw(b, 1,4) as char from tab1;

|STEM |
```

Функции для работы с BLOB-данными

С точки зрения поиска информации внутри BLOB-данных все возможные виды BLOB-данных (текст, графика, музыка, видеоизображение и т.п.) рассматриваются СУБД ЛИНТЕР одинаково – как набор слов.

Слово внутри BLOB-данных – это непрерывная последовательность из букв, цифр, знаков подчеркивания и дефисов длиной не более 239 знаков, все остальные знаки, встреченные в BLOB-значении (пробелы, знаки табуляции, графические символы и другие), считаются разделителями и в поиске не участвуют.

Более широкие возможности для работы с BLOB-данными, содержащими только текстовую информацию в различных форматах (DOC, PDF, RTF и т.д.), предоставляют средства полнотекстового поиска (см. документ «Полнотекстовый поиск в базе данных», раздел «Элементы языка»).

Еще один набор функций для работы с BLOB-данными предоставляет утилита `inl` (см. документ «Командный интерфейс», раздел «Работа с BLOB-данными»):

```
blob {insert | clear | append | get}
{rowid=<rowid записи>
<имя схемы>.<имя таблицы>.<имя столбца>
|column=<номер столбца>} {file=<имя файла>
|<BLOB-данные>};
```

Например,

```
username SYSTEM/MANAGER8
create or replace table test(i int, bl1 blob, bl2 blob);
insert into test(i,bl1,bl2) values(1,NULL,NULL);
blob insert column=2 333333ABCD;
blob append column=3 444444ABCD;
```

Пример

Пусть в качестве BLOB-данных записан следующий текст:

```
SELECT $$$S34,$$$S13 FROM $$$SYSRL,$$$USR WHERE $$$S12 =$$$S31
AND $$$S32 =0 AND ACCESS(USER,$$$S11) <> '------';
```

Этот текст содержит следующий набор слов (с точки зрения функций BLOB):

```
SELECT S34 S13 FROM SYSRL USR WHERE S12 S31 AND S32 0 ACCESS USER
 S11
```

Присутствующие в BLOB-значении символы «$», «,», «=», «(», «)», «<», «>», «'» не являются элементами слов и интерпретируются как разделители.

Каждое слово BLOB-значения имеет смещение – номер (от начала BLOB-значения) первого байта (знака) слова. Смещения начинаются с нуля.

Определение числа слов BLOB-значения (COUNTBLOB)

Функция

Определение числа слов BLOB-значения, соответствующих указанному шаблону.

Спецификация

[1] <синтаксис>::= COUNTBLOB (<имя столбца>, <шаблон поиска>)
[2] <шаблон поиска>::= <символьный литерал>|<UNICODE-литерал>

Синтаксические правила

1) `<Имя столбца>` должно соответствовать столбцу с BLOB-данными.
2) BLOB-значение должно состоять из набора слов (последовательность алфавитно-цифровых символов), разделителем которых может быть любой другой символ.

3) <Шаблон поиска> может содержать стандартные специальные символы: подчеркивание «_» представляет собой указатель на произвольный символ, процент «%» - указатель на подстроку (возможно, пустую).
4) В <шаблоне поиска> допустим <UNICODE-литерал>.
5) Поиск выполняется во всех BLOB-значениях, получаемых в <запросе выборки>.
6) В случае, если <шаблон поиска> имеет символьный тип, то выполняется поиск символьных строк; если <шаблон поиска> имеет UNICODE-тип, то выполняется поиск UNICODE-строк.
7) Значение и шаблон поиска приводятся в верхнему регистру.
8) Аргумент <шаблон поиска> может быть задан <SQL-параметром>, который должен содержать спецификацию типа данных параметра.

```
create or replace table tst (I int, bl blob);
insert into tst values (1, '11 222 334 55 11');
select countblob(bl, ? (char(2))) from tst;
%1

|         2|
```

Возвращаемое значение

1) Возвращается число найденных соответствий заданному шаблону.
2) Тип возвращаемого значения – INT.

Пример

```
SELECT FName,COUNTBLOB(TextBlob, 'стандарт%') FROM fb ORDER BY
 FName;
SELECT Author
 FROM Da WHERE COUNTBLOB(Da.ArticleText, 'Russian') >=10;

Скрипт tst.sql:
create or replace table test(id int, bl blob);
insert into test(id,bl) values(1, NULL);
blob insert column=2
4600690072007300740020004300750073007400 6F006D0065007200;
insert into test(id,bl) values(2, NULL);
blob insert column=2
31002000420065006400;
insert into test(id,bl) values(3, NULL);
blob insert column=2
6200620062006200;
!Найдено
select id from test where CountBLOB(bl,n'bbbb') > 0;

id
|  3|

!Ничего не найдено:
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
select id from test where CountBLOB(bl,'bbbb') > 0;

Выполнение
inl -u SYSTEM/MANAGER8 -f tst.sql
```

Поиск слова (FINDBLOB)

Функция

Поиск слова, соответствующего заданному шаблону.

Спецификация

[1] <синтаксис>::=
FINDBLOB (<имя столбца>, <шаблон поиска>, <номер вхождения>)

Синтаксические правила

1) <Имя столбца> должно соответствовать столбцу с BLOB-данными.
2) <Шаблон поиска> может содержать стандартные специальные символы: подчеркивание «_» представляет собой указатель на произвольный символ, процент «%» – указатель на подстроку (возможно, пустую).
3) В <шаблоне поиска> под элементом текста (словом) понимается любая последовательность символов, включающая только буквы, цифры, подчеркивание и дефис. Все остальные символы считаются разделителями.
4) В <шаблоне поиска> допустим <UNICODE-литерал>.
5) Поиск выполняется во всех BLOB-значениях, получаемых в <запросе выборки>.
6) В случае, если <шаблон поиска> имеет символьный тип, то выполняется поиск символьных строк; если <шаблон поиска> имеет UNICODE-тип, то выполняется поиск UNICODE-строк.
7) <Числовое выражение> в параметре <номер вхождения> – целое неотрицательное (больше нуля) значение типа INTEGER, задающее порядковый номер слова, которое должно быть найдено.
8) Значение и шаблон поиска приводятся в верхнему регистру.
9) Аргументы <шаблон поиска> и <номер вхождения> могут быть заданы <SQL-параметром>, который должен содержать спецификацию типа данных параметра.

```
create or replace table tst (I int, bl blob);
insert into tst values (1, '11 222 334 55 11');
select findblob(bl, ? (char(2)), ? (int)) from tst;
%1
2

|          15|
```

Возвращаемое значение

1) Возвращается смещение найденного слова или 0, если все слова BLOB-значения не соответствуют шаблону. Смещение начинается с 1.
2) При поиске в UNICODE-значения возвращается не номер символа, а номер байта, с которого начинается этот символ.

3) Тип возвращаемого значения – INT.

Примеры

```
SELECT FName,FINDBLOB(TextBlob, 'табл%', 1)
  FROM fb
 WHERE FINDBLOB(TextBlob, 'табл%', 2) >500 ORDER BY FName;

SELECT Author,
 FINDBLOB(ArticleText, 'Russian',
          COUNTBLOB(ArticleText, 'Russian')) AS LastWord
 FROM Da
 WHERE COUNTBLOB(Da.ArticleText, 'Russian') > 0
```

Поиск слова в текстовых BLOB-данных (FINDRTF)

Функция

Поиск слова, соответствующего заданному шаблону, в текстовых BLOB-данных.

Поиск выполняется в BLOB-данных в форматах RTF, PDF, DOC, XLS, PPT и PS.

Спецификация

[1] <синтаксис>::=
FINDRTF (<имя столбца>, <шаблон поиска>, <номер вхождения>)

Синтаксические правила

1) <Имя столбца> должно соответствовать столбцу с BLOB-данными.
2) <Шаблон поиска> может содержать стандартные специальные символы: подчеркивание «_» представляет собой указатель на произвольный символ, процент «%» – указатель на подстроку (возможно, пустую).
3) В качестве <шаблона поиска> можно использовать <UNICODE-литерал>.
4) Поиск выполняется во всех BLOB-значениях, получаемых в <запросе выборки>.
5) <Числовое выражение> в параметре <номер вхождения> – целое неотрицательное (больше нуля) значение типа INTEGER, задающее порядковый номер слова, которое должно быть найдено.
6) Аргументы <шаблон поиска> и <номер вхождения> могут быть заданы <SQL-параметром>, который должен содержать спецификацию типа данных параметра.

   Например,

```
Файл text.rtf:
ЛИНТЕР: назначение и основные возможности
ЛИНТЕР - это мощная система управления базами данных (СУБД).

create or replace table tst (i int, bl blob);
insert into tst(i,bl) values(1,NULL);
blob insert column=2 file=text.rtf;
select findrtf(bl,'%ЛИН%', 2) from tst;
select findrtf(bl,? (char(10)), ?(int)) from tst;
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
%ЛИН%
2

|          43|
```

Возвращаемое значение

1) Если BLOB-данные представлены в формате RTF (в соответствии со спецификацией формата RTF это определяется по наличию префикса \rtf в начале BLOB-данных), то сначала осуществляется декодирование символов кириллицы (символы латиницы в RTF не кодируются), а затем поиск нужной строки.
2) Если BLOB-данные представлены не в форматах PDF, DOC, XLS, PPT и PS, функция выполняется как FINDBLOB.
3) Возвращается смещение найденного слова или 0, если все слова BLOB-значения не соответствуют шаблону. Смещение начинается с 1.
4) Тип возвращаемого значения – INT.

Местоположение искомых элементов текста (GETTEXTPOS)

Функция

Предоставление списка позиций (местоположения) заданных элементов текстовых данных. Под элементом текста (словом) понимается любая последовательность символов, включающая только буквы, цифры, подчеркивание и дефис. Все остальные символы считаются разделителями.

Спецификация

[1] <синтаксис>::=
GETTEXTPOS (<имя столбца>, <поисковое выражение>[, <тип поиска>][, <начало поиска>][, <объем поиска>])
[2] <поисковое выражение>::=
<шаблон поиска>[| <шаблон поиска> ...]
[3] <шаблон поиска>::= <LIKE-шаблон>|<CONTAINS-шаблон>
[4] <LIKE-шаблон>::= <символьный литерал>
[5] <CONTAINS-шаблон>::= <символьный литерал>
[6] <тип поиска>::= 1 | 2 | 5 | 6
[7] <начало поиска>::= <значимое выражение>
[8] <объём поиска>::= <значимое выражение>

Синтаксические правила

1) <Имя столбца> должно соответствовать столбцу типа BLOB, EXTFILE, CHAR, VARCHAR, NCHAR, NCHAR VARYING.
2) <LIKE-шаблон> должен соответствовать спецификации <предиката подобия> (см. пункт «Предикат подобия») и может содержать стандартные специальные символы: подчеркивание «_» представляет собой указатель на произвольный символ, процент «%» – указатель на подстроку (возможно, пустую).

```
create or replace table test(c char(100));
insert into test(c)
  values ('11 22 333 11 4411 55 666 1177 811 1199');
select gettextpos(c,'11%',2,1,2) from test;
```

```
|0000000002 0000000013 1 2 11 2|
```

```
select gettextpos(c,'11%|22|4%',2,1,10) from test;

|0000000006 0000000000 1 2 4 2 11 2 14 4 26 4 35 4|
```

3) Следует учитывать различие между шаблоном подобия в <предикате подобия> и в <LIKE-шаблоне> данной функции. В <предикате подобия> шаблон подобия распространяется на весь массив символов указанного столбца, в то время как в данной функции он распространяется только на отдельные элементы текста (слова), например:

```
create or replace table tst (c varchar(500));
insert into tst(c) values ('_11_');
insert into tst(c) values ('_11 aa_');
insert into tst(c) values ('_11aa_');
При выполнении этого запроса находим все 3 записи:
select rowid from tst where c like '_11%_';
1
2
3
При выполнении нижеследующего запроса находим заданные элементы
 текста (слова)
только в 1 и 3 строках, т.к. во второй строке набор символов '_11
 aa_'
рассматривается функцией GetTextPos как два разных элемента текста
 (два слова),
каждый из которых не удовлетворяет шаблону поиска _11%_:

select gettextpos(c, '_11%_',2,1,length(c)) from tst;

|0000000001 0000000000 1 4                          |
|0000000000 0000000000                               |
|0000000001 0000000000 1 6                          |
```

4) <CONTAINS-шаблон> должен соответствовать спецификации <предиката фразового поиска> (см. документ «Полнотекстовый поиск в базе данных», пункт «Местоположение искомых элементов текста во внешнем файле») со следующими ограничениям: <поисковое выражение> не должно содержать логических условий («и», «или», «не равно», расстояние между искомыми словами и т.п.).

```
select gettextpos(c,'11*',1,1,10) from test;

|0000000004 0000000000 1 2 11 2 26 4 35 4|
```

Длина обоих шаблонов должна быть не больше 64 символов (иначе шаблон поиска усекается до 64 символов).

5) Если одновременно указывается несколько шаблонов поиска, то все они должны быть однотипными (либо только <LIKE-шаблоны>, либо <CONTAINS-шаблоны>).

6) <Тип поиска> это битовая маска, задающая атрибуты поиска:
   - 1 (001) – поиск по <CONTAINS-шаблону>;

- 2 (010) – поиск по <LIKE-шаблону>;

Значение 4 (100) – поиск с учетом регистра (обязательно объединение с одним из двух предыдущих атрибутов), т.е. <тип поиска> может так же принимать значения:

- 5 (101) – поиск по <CONTAINS-шаблону> с учетом регистра;
- 6 (110) – поиск по <LIKE-шаблону> с учетом регистра.

Из двух флагов-атрибутов CONTAINS (1) и LIKE (2) обязательно должен быть установлен ровно один.

7) Если <тип поиска> не задан, по умолчанию используется значение 1 (поиск по <CONTAINS-шаблону>), и в этом случае последующие аргументы функции не должны задаваться (будут использованы их значения по умолчанию).

Конструкция

```
select gettextpos(c,'11*') from test;
```

эквивалента

```
select gettextpos(c,'11*',1,1,0) from test;
```

8) <Начало поиска> задает номер позиции (значение типа INTEGER) в данных, начиная с которой необходимо выполнять поиск элементов по шаблону. Отсчет позиций начинается с 1. Аргумент необязательный; если не задан, по умолчанию принимается значение 1.

9) Если <начало поиска> не задано, по умолчанию используется 1 (поиск с начала), и в этом случае последующий аргумент функции не должен задаваться (будет использовано значение по умолчанию).

Конструкция

```
select gettextpos(c,'11*',1) from test;
```

эквивалента

```
select gettextpos(c,'11*',1,1,0) from test;
```

10) <Объем поиска> (значение типа INTEGER) ограничивает количество маркируемых элементов:

- 0 – маркировать все найденные элементы;
- n – выполнять прямой поиск; маркировать заданное (n) количество элементов;
- -n – выполнять обратный поиск; маркировать заданное (n) количество элементов.

Аргумент необязательный; если не задан, по умолчанию принимается значение 0.

11) Если аргумент <начало поиска> не задан или имеет значение 1, а значение аргумента <объем поиска> – отрицательное, то поиск выполняется с конца данных.

```
create or replace table test(c char(100));
insert into test(c)
 values ('25 a11 22 333 11bc 4411 55 666 1177 811 1199');
```

```
select gettextpos(c,'11*',1,1,-3) from test;

|0000000000 0000000000 |
(искомые элементы текста не найдены)

select gettextpos(c,'11*',1,length(c)/2,-3) from test;

|0000000001 0000000000 15 4|
(найден 1 элемент текста 11bc, находящийся на 15 позиции и имеющий
 длину 4
 символа).
```

12) Аргументы <поисковое выражение>, <начало поиска> и <объем поиска> могут быть заданы <SQL-параметром>, который должен содержать спецификацию типа данных параметра.

```
create or replace table test(c char(100));
insert into test(c)
 values ('11 22 333 11 4411 55 666 1177 811 1199');
select gettextpos(c,? (char(5)),2, ?(int),? (int)) from test;
11%
1
2

|0000000002 0000000013 1 2 11 2|
```

Возвращаемое значение

1) Тип возвращаемого значения – VARCHAR(2000).
2) Структура возвращаемой строки:

```
<количество><разделитель><продолжение поиска>[<описатель
 элемента>]…
<описатель элемента>::=
<разделитель><позиция элемента><разделитель><длина элемента>
```

где:

- <разделитель> – символ пробела;
- <количество> – количество промаркированных элементов текста. Поле имеет фиксированную длину – 10 символов;
- <продолжение поиска> – позиция, с которой необходимо продолжать сканирование данных (после последнего выданного маркированного элемента). Поле имеет фиксированную длину – 10 символов. Если сканирование данных выполнено полностью, значение поля будет равно 0;
- <описатель элемента> – описатель маркированного элемента текста. Элементы описателя имеют парное значение: <позиция элемента> <длина элемента>, которое представляет, соответственно, позицию найденного элемента и его фактическую длину.

```
create or replace table test(c char(256));
```

```
insert into test values ('RELEX is one of the leading Russian
 application and system software developers');
```

Запрос осуществляет поиск элементов текста, начинающихся с "develop" (с учетом регистра) или совпадающих с "russian" (без учета регистра).

```
select gettextpos(c,'#develop*|russian',1,1,0) from test;

| 0000000002 0000000000 29 7 69 10 |
```

Результат поиска:

найдено 2 элемента текста (<количество> равно 0000000002)

найдены все элементы (<продолжение поиска> равно 0000000000)

первый найденный элемент находится на 29 позиции и имеет длину 7 символов (слово Russian)

второй найденный элемент находится на 69 позиции и имеет длину 10 символов (слово developers).

Выборка текста (GETTEXT)

Функция

Получение заданной порции текстовых данных.

Спецификация

[1] <синтаксис>::= GETTEXT (<имя столбца>, <смещение>, <длина>)
[2] <длина>::= <числовой литерал>

Синтаксические правила

1) <Имя столбца> должно принадлежать столбцу, тип данных которого позволяет создавать для него фразовый индекс (см. документ «Полнотекстовый поиск в базе данных», пункт «Выборка порции текста внешнего файла»). Сам индекс при этом может и не существовать.
2) <Смещение> – целое положительное число, задающее начальную позицию требуемой порции текста. Первый символ текста имеет смещение 1.
3) <Длина> – размер требуемой порции текста в символах (целое число в диапазоне от 1 до 2000). Количество символов (и смещение) задаётся в символах оригинального документа. Если у пользователя стоит кодировка MBCS или UTF-8, то длина результата преобразования может превысить 4000 байт, разрешённых для столбца. В результате часть данных не будет передана. Нужно учитывать эту особенность в случае, когда извлекаются непрерывные куски текста несколькими порциями.
4) Аргумент <смещение> может быть задан <SQL-параметром>, который должен содержать спецификацию типа данных параметра.

```
create or replace table test(c char(100));
insert into test(c) values ('Мы едем, едем, едем в далёкие края');
select gettext(c, ? (int), 20) from test;
15
```

```
| едем в далёкие края                                                    |
```

Возвращаемое значение

1) Возвращается затребованная порция прошедшего через фразовый фильтр текста, определяемая указанными смещением и длиной.
2) В возвращаемом значении порция текста после последнего символа текста столбца заполняется пробелами.
3) Если размер требуемой порции текста превышает `2000` символов, то возвращается 2000 символов текста, а оставшаяся часть результата заполняется пробелами.
4) Для файлов типа `XML` и `HTML` функция возвращает только чистый текст (пары «атрибут-значение» игнорируются).
5) Значение `NULL` возвращается в случае, если один из параметров имеет значение `NULL`, если требуемый файл не существует (для столбца типа `EXTFILE`) или если в процессе получения требуемой порции текста произошла ошибка.
6) Правила выбора фразового фильтра совпадают с правилами, используемыми при создании фразового индекса со значениями флагов, принятыми по умолчанию для данного типа столбца (см. функцию `create phrase index` в документе «Полнотекстовый поиск в базе данных», пункт «Создание фразового индекса»).
7) Даже если в системной таблице `$$$CHARSET` не задана кодировка `866`, `1251` или `20866`, то перекодировка текста все равно будет выполнена правильно с помощью встроенных в СУБД ЛИНТЕР внутренних таблиц кодировки. В противном случае символы с кодами меньше `0x80` извлекаются как есть (подразумевается 7-битный `ASCII`), а остальные символы заменяются на '?'.

Примечание
Поскольку выдается текст, прошедший через фильтр, в нем могут отсутствовать знаки препинания, символы разметки.

Пример

Найти позицию третьего повторения фразы «баз данных» в документе с идентификатором `10`:

```
select instr(gettext(text_doc,1,1500),'баз данных',1,3) from "ph"
 where id_doc=10;
```

Дать порцию BLOB-значения (GETBLOB)

Функция

Дать порцию BLOB-значения.

Спецификация

[1] <синтаксис>::=
GETBLOB (<имя столбца>, <смещение порции>, <размер порции>)
[2] <смещение порции>::= <числовое выражение>
[3] <размер порции>::= <числовой литерал>

Синтаксические правила

1) При выполнении функции BLOB-данные рассматриваются как непрерывный массив байт, т.е. выделение слов игнорируется.

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

2) <Смещение порции> – целое неотрицательное значение типа INTEGER, задающее номер байта, с которого начинается порция данных. Отсчет выполняется с 1.

3) <Размер порции> – целое неотрицательное значение типа INTEGER не больше 4000, задающее размер порции в байтах.

4) Выбор порции выполняется во всех BLOB-значениях, получаемых в <запросе выборки>.

5) Аргумент <смещение> может быть задан <SQL-параметром>, который должен содержать спецификацию типа данных параметра.

```
create or replace table test(bl blob);
insert into test values(null);
blob insert column=1 333333ABCDc0ff4532;
select getblob(bl, ? (int), 5) from test;
7
5

| FF 45 32 AB CD|
```

Возвращаемое значение

1) Возвращается строка длиной <размер порции> (для строк с фиксированной длиной) или строка фактической длины (для строк с переменной длиной в случае, если размер BLOB-данных меньше запрошенной порции).

2) Если размер BLOB-данных меньше запрошенной порции, то возвращаемые строки фиксированной длины дополняются пробелами до запрошенной длины.

3) Если один из параметров имеет значение NULL или требуемая порция не найдена, возвращается значение NULL.

4) Тип возвращаемого значения BYTE (данные представлены в виде последовательности байт).

5) Если BLOB-данные имеют значение NULL, то возвращается значение NULL.

Примеры

```
SELECT FName, GETBLOB(TextBlob,1,60) FROM fb ORDER BY FName;

SELECT GETBLOB (ArticleText,FINDBLOB(ArticleText, 'Russian', 1),
 32) AS FirstWord
       GETBLOB (ArticleText, FINDBLOB(ArticleText, 'Russian',
 COUNTBLOB(ArticleText, 'Russian')), 32) AS LastWord
  FROM Da
 WHERE COUNTBLOB(Da.ArticleText, 'Russian') >0;
```

Дать порцию BLOB-значения в текстовом виде (GETBLOBSTR)

Функция

Дать порцию BLOB-значения в текстовом виде.

Спецификация

[1] <синтаксис>::=
GETBLOBSTR (<имя столбца>, <смещение порции>, <размер порции>[, <режим подсчёта>])
[2] <режим подсчёта>::= <числовой литерал>

Синтаксические правила

1) При выполнении функции BLOB-данные рассматриваются как непрерывный массив байт, т.е. выделение слов игнорируется.
2) <Смещение порции> – целое неотрицательное значение типа INTEGER, задающее номер байта, с которого начинается порция данных. Отсчет выполняется с 1. Если кодировка столбца UNICODE, то <смещение порции> должно быть нечётным (т.к. не допускается получение половины UNICODE-символа).
3) <Размер порции> – целое неотрицательное значение типа INTEGER не больше 4000, задающее размер порции в байтах. Если кодировка столбца UNICODE, то <размер порции> должен быть чётным (т.к. не допускается получение половины UNICODE-символа).
4) Выбор порции выполняется во всех BLOB-значениях, получаемых в <запросе выборки>.
5) <Режим подсчёта> задает режим подсчета длины:
   - 0 – в байтах (по умолчанию). Означает, что и <смещение порции>, и <размер порции> заданы в байтах;
   - 1 – в символах. Означает, что и <смещение порции>, и <размер порции> заданы в символах.
6) Аргумент <смещение порции> может быть задан <SQL-параметром>, который должен содержать спецификацию типа данных параметра.

```
create or replace table test(bl blob);
insert into test values(null);
blob insert column=1 333333ABCDc0ff4532;
select getblobstr(bl, ? (int), 10,1) from test;
2

|33л-└ E2  |
```

Возвращаемое значение

1) Возвращается текстовая строка длиной <размер порции>. Если требуемая порция не найдена, строка будет содержать пробелы.
2) Тип возвращаемого значения:
   - NCHAR: в случае кодировки UCS2;
   - CHAR: во всех остальных случаях, текст будет автоматически перекодирован из кодировки, указанной при создании BLOB-столбца, в кодировку канала.
3) Если BLOB-данные имеют NULL-значение, возвращается строка пробелов.
4) Если один из параметров имеет значение NULL, то результат будет NULL.

Примеры

```
create or replace table tst ("Информация о БД"  blob);
```

```
insert into tst ("Информация о БД")
values ('СУБД ЛИНТЕР – реляционная мобильная ...');

select getblob("Информация о БД",1,40) from tst;

| D1 D3 C1 C4 20 CB C8 CD D2 C5 D0 20 2D 20 F0 E5 EB FF F6 E8 EE
 ED ED E0 FF 20 EC EE E1 E8 EB FC ED E0 FF 20 2E 2E 2E 20|

select getblobstr("Информация о БД",1,40) from tst;|СУБД ЛИНТЕР –
 реляционная мобильная ... |

create or replace table blb(bl blob character set "CP866");
insert into blb values (n'12АБВГ');
select getblobstr(bl,3,2),getblobstr(bl,3,2,1) from blb;

|АБ|АБ|

create or replace table blb(bl blob character set "UCS2");
insert into blb values (n'12АБВГ');
select getblobstr(bl,5,4),getblobstr(bl,3,2,1) from blb;

|АБ|АБ|

create or replace table blb(bl blob character set "UTF-8");
insert into blb values (n'12АБВГ');
select getblobstr(bl,3,4),getblobstr(bl,3,2,1) from blb;

|АБ  |АБ    |
```

Определение длины BLOB-значения (LENBLOB)

Функция

Определение длины BLOB-значения.

Спецификация

[1] <синтаксис>::= LENBLOB (<имя столбца>[, <режим подсчёта>])

Синтаксические правила

1) Длина порции вычисляется для всех BLOB-значений, получаемых в `<запросе выборки>`.
2) `<Режим подсчёта>` задает режим подсчета длины:
   - `0` – в байтах (по умолчанию);
   - `1` – в символах.

Возвращаемое значение

1) Возвращается размер BLOB-значения (в байтах или символах).

2) Если BLOB-данные имеют NULL-значение, возвращается значение 0.

3) Тип возвращаемого значения – INT.

Примеры

```
create or replace table blb(bl blob character set "CP866");
insert into blb values (n'12АБВГ');
select lenblob(bl),lenblob(bl,1) from blb;

|          6|          6|
```

```
create or replace table blb(bl blob character set "UCS2");
insert into blb values (n'12АБВГ');
select lenblob(bl),lenblob(bl,1) from blb;

|         12|          6|
```

```
create or replace table blb(bl blob character set "UTF-8");
insert into blb values (n'12АБВГ');
select lenblob(bl),lenblob(bl,1) from blb;

|         10|          6|
```

Модификация подстроки BLOB-данных (SET/INSERT)

Функция

Модификация заданной подстроки BLOB-данных с помощью SQL-запросов UPDATE и UPDATE CURRENT.

Спецификация

[1] <модификация порции BLOB-данных>::=
SET <столбец>=<спецификация модифицируемой порции>
[2] <спецификация модифицируемой порции>::=
INSERT (<столбец>, <позиция>, <длина>, <подстрока>)
[3] <столбец>::= <идентификатор>
[4] <подстрока>::= <символьное выражение>|<байтовое выражение>

Синтаксические правила

1) <Символьное выражение> в аргументе может иметь следующий тип данных: CHAR, VARCHAR, NCHAR, NCHAR VARYING, BYTE, VARBYTE.

2) Длина <подстроки> не должна быть более 4000.

3) <Позиция> задает позицию в байтах заменяемой подстроки в <столбце>. Отсчет начинается с 1. Если BLOB-столбец имеет кодовую страницу UNICODE, то <позиция> не может быть чётной, в противном случае выдаётся код завершения 1036 («Значение аргумента в недопустимом диапазоне»).

4) <Длина> задает длину заменяемой подстроки в <столбце> в байтах. Если BLOB-столбец имеет кодовую страницу UNICODE, то <позиция> не может быть нечётной, в противном случае выдаётся код завершения 1036 («Значение аргумента в недопустимом диапазоне»).

5) <Подстрока> задает значение, вставляемое вместо удаленной подстроки.

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

6) Если <подстрока> имеет символьный тип данных (CHAR, VARCHAR, NCHAR, NCHAR VARYING), она автоматически перекодируется в кодовую страницу, заданную для BLOB-столбца. Если <подстрока> имеет байтовый тип данных, она будет использоваться «как есть».

7) Аргументы <позиция>, <длина> и <подстрока> могут быть заданы <SQL-параметром>, который должен содержать спецификацию типа данных параметра.

```
create or replace table test(bl blob character set "UCS2");
insert into test(bl) values('0123456789 aaa 0123456789');

select lenblob(bl), getblobstr(bl, 1, 60) from test;

|         50|0123456789 aaa 0123456789|

update test set bl=insert(bl, ? (int), ? (int), ? (char(2)));
3
2
aa

select lenblob(bl), getblobstr(bl, 1, 60) from test;

|         50|01aa456789 aaa 0123456789|
```

Общие правила

1) Начиная с <позиции>, в BLOB-столбце удаляется <длина> байт, и на их место вставляются символы или байты <подстроки>.

2) Если будет предпринята попытка заменить часть BLOB-данных на порцию, <длина> которой в байтах отличается от заменяемой части, то будет выдан код завершения 1721 («Неверное смещение в странице BLOB»).

Возвращаемое значение

Отсутствует. Если значения аргументов функции правильные, выполняется корректировка BLOB-данных, в противном случае BLOB-данные не изменяются.

Пример

```
create or replace table test(bl blob character set "UCS2");
insert into test(bl) values('0123456789 aaa 0123456789');
select lenblob(bl), getblobstr(bl, 1, 60) from test;

|         50|0123456789 aaa 0123456789|

update test set bl=insert(bl, 3, 2, 'aa');
select lenblob(bl), getblobstr(bl, 1, 60) from test;

|         50|01aa456789 aaa 0123456789|

update test set bl=insert(bl, 5, 3, HEX('310031003100'));
```

```
select lenblob(bl), getblobstr(bl, 1, 60) from test;

|         50|01aa111789 aaa 0123456789|
```

Примечание
В конструкции «bl=insert(bl, 3, 10, 'aa')» часть «bl=» является вспомогательной, то есть реального присвоения не происходит, а собственно модификация BLOB-данных осуществляется внутри функции INSERT.

Модификация подстрок BLOB-данных (SET/REPLACE)

Функция

Модификация всех заданных подстрок BLOB-данных с помощью SQL-запросов UPDATE и UPDATE CURRENT.

Спецификация

[1] <модификация подстрок порции BLOB-данных>::=
SET <столбец>=<спецификация модифицируемых подстрок>
[2] <спецификация модифицируемых подстрок>::=
REPLACE (<столбец>, <подстрока1>, <подстрока2>)

Синтаксические правила

1) <Подстрока1>, <подстрока2> должны иметь типы данных: CHAR, VARCHAR, NCHAR, NCHAR VARYING, BYTE, VARBYTE.
2) Длина <подстроки1>, <подстроки2> не должна быть более 4000.
3) <Подстрока1> задает удаляемое из <столбца> значение.
4) <Подстрока2> задает вставляемое вместо удаленной <подстроки1> значение.
5) Если <подстрока1> и <подстрока2> имеют символьный тип данных (CHAR, VARCHAR, NCHAR, NCHAR VARYING), они автоматически перекодируются в кодовую страницу, заданную для BLOB-столбца. Если <подстрока1> и <подстрока2> имеют байтовый тип данных, они будут использоваться «как есть».
6) Аргументы <подстрока1> и <подстрока2> могут быть заданы <SQL-параметром>, который должен содержать спецификацию типа данных параметра.

```
create or replace table test(bl blob);
insert into test(bl) values('Мы едем, едем, едем в далекие
 края.');
select lenblob(bl), getblobstr(bl, 1, 35) from test;

|         35|Мы едем, едем, едем в далекие края.|

update test set bl=replace(bl, :line1 (char(4)), :line2
 (char(4)));
едем
ЕДЕМ
select lenblob(bl), getblobstr(bl, 1, 35) from test;

|         35|Мы ЕДЕМ, ЕДЕМ, ЕДЕМ в далекие края.|
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

Общие правила

1) В текущей строке BLOB-столбца удаляются символы или байты всех <подстрок1>, и вместо них вставляются символы или байты <подстроки2>.
2) Если будет предпринята попытка заменить часть BLOB-данных на порцию, <длина> которой в байтах отличается от заменяемой части, то будет выдан код завершения 1721 («Неверное смещение в странице BLOB»).

Возвращаемое значение

Отсутствует. Если значения аргументов функции правильные, выполняется корректировка BLOB-данных, в противном случае BLOB-данные не изменяются.

Пример

```
create or replace table test(bl blob character set "UCS2");
insert into test(bl) values('0123456789 aaa 0123456789');
select lenblob(bl), getblobstr(bl, 1, 60) from test;

|         50|0123456789 aaa 0123456789|

update test set bl=replace(bl, '123456', 'замена');
select lenblob(bl), getblobstr(bl, 1, 60) from test;

|         50|0замена789 aaa 0замена789|

update test set bl=replace(bl, HEX('37003800'), HEX('78007800'));
select lenblob(bl), getblobstr(bl, 1, 60) from test;

|         50|0заменаxx9 aaa 0заменаxx9|
```

Примечание
В конструкции «set bl=replace(bl, '123456', 'замена')» часть «bl=» является вспомогательной, т.е. реального присвоения не происходит, а собственно модификация BLOB-значения осуществляется внутри функции REPLACE.

Функции для работы с типом DATE

Текущая дата в формате UNIX (UNIX_TIMESTAMP)

Функция

Предоставление текущей даты в формате UNIX – то есть в виде метки времени. Метка времени содержит дату в виде целого числа секунд, начиная с 1 января 1970 года (начало эпохи UNIX).

Спецификация

[1] <синтаксис>::= UNIX_TIMESTAMP ([<дата-время выражение>])

Синтаксические правила

1) Если аргумент задан, то он считается заданным в GMT.

2) Аргумент <дата-время выражения> может быть задан <SQL-параметром>, который должен содержать спецификацию типа данных параметра.

```
select unix_timestamp(? (date));
23.10.2013

|          1382486400|
```

Возвращаемое значение

1) Если аргумент не задан, возвращается временная метка текущей даты (количество секунд с 1970-01-01 00:00:00 GMT до текущей даты).

```
select unix_timestamp(),unix_timestamp(sysdate);

|          1197632318|          1197632318|
```

2) Если аргумент задан, возвращается временная метка заданной даты (количество секунд с 1970-01-01 00:00:00 GMT до заданной даты). Если дата оказывается меньшей 1970-01-01 00:00:00 GMT, возвращается отрицательное число.

```
select unix_timestamp(to_date('01.01.1970', 'dd.mm.yyyy')),
       unix_timestamp(to_date('01.01.0001', 'dd.mm.yyyy'));

|                   0|        -62135683200|
```

3) Тип возвращаемого значения – BIGINT.

Сокращенное название дня даты (DAYNAME)

Функция

Формирование сокращенного названия дня даты.

Спецификация

[1] <синтаксис>::= DAYNAME (<дата-время выражение>)

Синтаксические правила

1) Предполагается, что аргумент задан в среднем времени по Гринвичу (GMT).
2) Аргумент <дата-время выражения> может быть задан <SQL-параметром>, который должен содержать спецификацию типа данных параметра.

```
select dayname(? (date));
01.01.2013

|tue|
```

Возвращаемое значение

1) Возвращается трехсимвольное имя дня указанного <дата-время выражения>.
2) Тип возвращаемого значения – CHAR(3).

Примеры

```
select dayname(sysdate);
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
|wed |
select case dayname(sysdate) when 'wed' then 'Среда' end;

|Среда |
```

Сокращенное название месяца даты (MONTHNAME)

Функция

Выделение сокращенного названия месяца даты.

Спецификация

[1] <синтаксис>::= MONTHNAME (<дата-время выражение>)

Синтаксические правила

1) Предполагается, что аргумент задан в среднем времени по Гринвичу (GMT).
2) Аргумент <дата-время выражения> может быть задан <SQL-параметром>, который должен содержать спецификацию типа данных параметра.

```
select monthname(? (date));
01.01.2013

|jan|
```

Возвращаемое значение

1) Возвращается трехсимвольное имя месяца указанного <дата-время выражения>.
2) Тип возвращаемого значения – CHAR(3).

Пример

```
select monthname(sysdate+10);

|apr |
```

Выделение заданных элементов даты (DATESPLIT/ EXTRACT)

Функция

Выделение заданных элементов даты.

Спецификация

Варианты:

1) [1] <синтаксис>::= DATESPLIT(<дата-время выражение>, <параметр>)
   [2] <параметр>::= <символьный литерал>
2) [1] <синтаксис>::= EXTRACT(<элемент даты> FROM <дата-время выражение>)

[2] <элемент даты>::=
{YEAR | MONTH | DAY | HOUR | MINUTE | SECOND}

Синтаксические правила

1) <Дата-время выражение> должно быть представлено в одном из форматов по умолчанию.
2) <Дата-время выражение> может быть представлено в виде литерала типа <дата-время>.
3) <Параметр> определяет возвращаемое функцией значение.

   Допустимы следующие значения <параметра>:

Значение параметра	Возвращаемое значение
'D'	День месяца
'M'	Номер месяца
'QY'	Номер квартала
'Y'	Год
'DW'	Номер дня недели
'DY'	Номер дня в году
'WM'	Номер недели в месяце
'WY'	Номер недели в году
'ND'	Номер дня от начала нашей эры
'NW'	Номер недели от начала нашей эры
'NM'	Номер месяца от начала нашей эры
'HH'	Количество часов (диапазон 00-23)
'HH12'	Количество часов (диапазон 0-12)
'HH24'	Количество часов (диапазон 00-23)
'MI'	Количество минут
'SS'	Количество секунд
'FF'	Количество тиков

4) Функция EXTRACT добавлена для совместимости со стандартом SQL2008.
5) <Элемент даты> определяет возвращаемое функцией значение:
   - YEAR – год;
   - MONTH – месяц;
   - DAY – день;
   - HOUR – час;
   - MINUTE – минуты;
   - SECOND – секунды.
6) Аргументы <дата-время выражение> и <параметр> могут быть заданы <SQL-параметром>, который должен содержать спецификацию типа данных параметра.

```
select datesplit(? (date), ? (char(2)));
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
01.01.2013
qy

|          1|

select extract(hour from ? (date));
01.01.2013:12:45

|                  12|
```

Возвращаемое значение

Функция DATESPLIT:

1) Указанный элемент <значимого выражения типа DATE>.
2) Тип возвращаемого значения – INT.

Функция EXTRACT:

1) Значение DECIMAL со SCALE=0 для всех элементов даты, кроме SECOND.
2) Значение DECIMAL со SCALE=2 для элемента даты SECOND.

Примеры

```
select avg(datesplit(sysdate, 'Y') -
  datesplit('28.04.1950', 'Y')) from person;
```

Определить, сколько длилась Великая Отечественная война:

```
select distinct 'Великая Отечественная война продолжалась ' ||
  cast datesplit('09.05.1945', 'ND') -
    datesplit('22.06.1941', 'ND') as char(5) || ' дней';

|Великая Отечественная война продолжалась 1417 дней|

select datesplit (cast '04-11-2006' as date, 'qy');

|4|
select sysdate, extract(month from sysdate);

|12.08.2007:12:05:45.23|          8|
```

Сдвиг даты на заданный интервал (MULTIME)

Функция

Сдвиг даты на заданный интервал.

Спецификация

[1] <синтаксис>::=
MULTIME (<тип интервала>, <интервал>, <исходная дата>)

[2] <тип интервала>::= <числовое выражение>
[3] <интервал>::= <числовое выражение>
[4] <исходная дата>::= <дата-время выражение>

Синтаксические правила

1) Допустимые значения параметра <тип интервала>:

Тип интервала	Единицы интервала
1	Тики
2	Секунды
4	Минуты
8	Часы
16	Дни
32	Недели
64	Месяцы
128	Кварталы
256	Годы

2) <Числовое выражение> приводится к целочисленному значению.
3) Все аргументы могут быть заданы <SQL-параметром>, который должен содержать спецификацию типа данных параметра.

```
select multime(:hour (int), ? (int), ? (date));
8
4
01.01.2013:12:45

|01.01.2013:16:45:00.00|
```

Возвращаемое значение

1) Результат функции не должен превышать дату 31.12.9999 года (в случае сдвига вперед) и дату начала нашей эры (в случае сдвига назад).
2) Возвращается значение типа DATE, увеличенное (уменьшенное) по сравнению с <исходной датой> на заданный <интервал>.
3) Если <исходная дата> представлена только временем, и <интервал> задает дни, недели, месяцы, кварталы или годы, то она перед вычислением устанавливается к текущей дате.

Примеры

Пусть файл multime1.sql содержит операторы:

```
select sysdate;
select multime(8,2,sysdate);
select multime(16,-4.7,sysdate);
select multime(256,.95e2,sysdate);
select multime(8,ceil(2.67), sysdate);
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
SQL>_multime1

|07.09.2010:10:29:21.73 |

|07.09.2010:12:29:21.73 |

|03.09.2010:10:29:21.73 |

|07.09.2105:10:29:21.73 |

|07.09.2010:13:29:21.73 |
```

Вычисление интервала между двумя датами (DIVTIME)

Функция

Вычисление интервала между двумя датами.

Спецификация

[1] <синтаксис>::=
DIVTIME (<тип интервала>, <начальная дата>, <конечная дата>)
[2] <начальная дата>::= <дата-время выражение>
[3] <конечная дата>::= <дата-время выражение>

Синтаксические правила

1) Допустимые значения параметра <тип интервала> см. в описании функции MULTIME.
2) Все аргументы могут быть заданы <SQL-параметром>, который должен содержать спецификацию типа данных параметра.

```
select divtime(:day (int),? (date),? (date));
16
22.06.1941
09.05.1945

|       1417|
```

Возвращаемое значение

1) Возвращается значение типа INTEGER, представляющее разницу между конечной и начальной датами в единицах, указанных параметром <интервал>.
2) Округление происходит в меньшую сторону, например, если <интервал> = 256 (годы), а <начальная дата> больше <конечной даты> хотя бы на тик, будет возвращено значение -1.
3) Если один из параметров NULL, то результат будет NULL.

Примеры

```
select divtime(16, to_date('20:10:2002','DD:MM:YYYY'),
```

```
  to_date('30:10:2002','DD:MM:YYYY'));

|10 |

select divtime(2, to_date('20:10:2002','DD:MM:YYYY'),
  to_date('30:10:2002','DD:MM:YYYY'));

|864000 |
```

Вычисление количества дней в дате (TO_DAYS/ TIMEINT_TO_DAYS)

Функция

Вычисление количества дней в указанной дате.

Спецификация

[1] <синтаксис>::=
TO_DAYS | TIMEINT_TO_DAYS (<значимое выражение>)

Синтаксические правила

1) <Значимое выражение> должно иметь тип DATE или приводиться к нему.
2) Аргумент может быть задан <SQL-параметром>, который должен содержать спецификацию типа данных параметра.

```
select to_days(? (date));
01.01.0001

|                    1.0|
```

Возвращаемое значение

Возвращается значение типа DECIMAL, представляющее количество дней от начала летоисчисления до указанной даты.

Примеры

```
select timeint_to_days(sysdate), round( timeint_to_days(sysdate));

|731323.46201388 |731323 |

select round(timeint_to_days('01.01.0001'));

|1 |

select distinct year + 1900,
  round(timeint_to_days (to_date('01.01.' ||
  to_char(year + 1900, '9999'), 'dd.mm.yyyy'))) from auto;

|1970 |719163 |
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
|1971 |719528 |
```

Количество дней между двумя датами:

```
select round(timeint_to_days('01.05.2003') -
  timeint_to_days('28.04.2003'));

|3 |
```

Вычисление даты по количеству дней (FROM_DAYS/ TIMEINT_FROM_DAYS)

Функция

Вычисление даты по заданному количеству дней.

Спецификация

[1] <синтаксис>::=
FROM_DAYS | TIMEINT_FROM_DAYS (<значимое выражение>)

Синтаксические правила

1) <Значимое выражение> должно иметь целый или вещественный тип данных либо приводиться к нему.
2) Аргумент может быть задан <SQL-параметром>, который должен содержать спецификацию типа данных параметра.

```
select from_days(? (int));
2000

|23.06.0006:00:00:00.00|
```

Возвращаемое значение

Значение типа DATE, насчитывающее заданное количество дней.

Примеры

```
select timeint_from_days (100000);

|16.10.0274:00:00:00.00|

select from_days(100000)
 union
select from_days(123456.789)
 union
select from_days(timeint_to_days(sysdate));

|16.10.0274:00:00:00.00|
|05.01.0339:18:56:09.60|
|14.12.2007:14:31:25.00|
```

Последний день месяца (LAST_DAY)

Функция

Вычисление последнего дня месяца для указанной даты.

Спецификация

[1] <синтаксис>::= LAST_DAY (<значимое выражение>)

Синтаксические правила

1) <Значимое выражение> должно иметь тип DATE или приводиться к нему.
2) Аргумент может быть задан <SQL-параметром>, который должен содержать спецификацию типа данных параметра.

```
select last_day(? (date));
23.02.2012

|29.02.2012:00:00:00.00|
```

Возвращаемое значение

Возвращается значение типа DATE, представляющее дату последнего дня того месяца, который выбран из аргумента функции.

Примеры

```
select last_day(sysdate), to_char(last_day(sysdate),'dd');

|31.08.2006:10:24:12| 31|
select last_day (cast '12-JAN-06' as date);

|31.01.2006:00:00:00|
select cast to_char
  (last_day(to_date('12-02-2006', 'dd-mm-yyyy')),'dd') as int,
       cast to_char
  (last_day(to_date('12-02-2008', 'dd-mm-yyyy')),'dd') as int;

|28|29|
```

Дата очередного дня недели (NEXT_DAY)

Функция

Вычисление даты очередного дня недели.

Спецификация

[1] <синтаксис>::= NEXT_DAY (<значимое выражение>, <день недели>)

Синтаксические правила

1) <Значимое выражение> должно иметь тип DATE или приводиться к нему.

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

2) <День недели> – символьное выражение или приводимое к нему, которое должно иметь одно из следующих значений:

Название дня недели		
Полное	**Сокращенное**	**Полное**
Monday	Mon	Понедельник
Tuesday	Tue	Вторник
Wednesday	Wed	Среда
Thursday	Thu	Четверг
Friday	Fri	Пятница
Saturday	Sat	Суббота
Sunday	Sun	Воскресенье

3) Аргументы могут быть заданы <SQL-параметром>, который должен содержать спецификацию типа данных параметра.

```
select next_day(? (date), ? (char(3)));
01.01.2014
sun

|05.01.2014:00:00:00.00|
```

Возвращаемое значение

1) Возвращается значение типа DATE, соответствующее указанному <дню недели> после заданной даты.
2) Значение времени в возвращаемой дате совпадает со временем в исходной дате.
3) Если запрашиваемый день недели совпадает с днем недели в исходной дате, возвращается дата следующего (т.е. через 7 дней) дня недели.

Примеры

1)

```
select next_day(cast '04.08.2006' as date, 'fri');

|11.08.2006:00:00:00|
```

2) Определить дату первого понедельника следующего месяца:

```
select next_day(last_day(cast '19-08-2006' as date),'monday');

|04.09.2006:00:00:00|
```

3) Определить дату дня недели после текущей даты (требуемый день недели берётся из таблицы):

```
create or replace table tst(day_week char(3));
insert into tst values('mon');
insert into tst values('tue');
```

```
select case dayname(next_day(sysdate, day_week))
         when 'mon' then 'понедельник'
         when 'tue' then 'вторник'
         when 'wed' then 'среда'
         when 'thu' then 'четверг'
         when 'fri' then 'пятница'
         when 'sat' then 'суббота'
         when 'sun' then 'воскресенье'
         else null end,
       next_day(sysdate, day_week)
  from tst where rowid=2;

|вторник   |08.08.2006:10:23:12|
```

Помесячное изменение даты (ADD_MONTHS)

Функция

Арифметическое добавление месяцев к исходной дате.

Спецификация

[1] <синтаксис>::=
ADD_MONTHS (<значимое выражение>, <количество месяцев>)

Синтаксические правила

1) <Значимое выражение> должно иметь тип DATE или приводиться к нему.
2) <Количество месяцев> – численное значение типа INT, SMALLINT, BIGINT, NUMERIC, REAL, DOUBLE или приводимое к нему.
3) Аргументы могут быть заданы <SQL-параметром>, который должен содержать спецификацию типа данных параметра.

```
select sysdate as "Дата зачатия",
  add_months(sysdate, ? (int)) as "Дата рождения";
9

 Дата зачатия           Дата рождения
 ------------           -------------
|12.03.2014:11:11:23.32|12.12.2014:11:11:23.32|
```

Возвращаемое значение

1) Возвращается значение типа DATE, увеличенное (уменьшенное) на заданное <количество месяцев>.
2) При положительном значении аргумента <количество месяцев> формируется будущая дата, при отрицательном – прошлая по сравнению с исходной.

```
select add_months(cast '15-04-2006' as date, 3),
  add_months(cast '15-04-2006' as date, -3);
```


© Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
|15.07.2006:00:00:00|15.01.2006:00:00:00|
```

3) Если значение параметра <количество месяцев> не является целочисленным значением, оно усекается до целой части.

```
select add_months(cast '15-04-2006' as date,3.6),
       add_months(cast '15-04-2006' as date,-3.4),
       add_months(cast '15-04-2006' as date,.5);

|15.07.2006:00:00:00|15.01.2006:00:00:00|15.04.2006:00:00:00|
```

4) При добавлении месяцев номер дня в результирующей дате не меняется, за исключением тех случаев, когда он приходится на конец месяца.

```
select add_months(cast '31-01-2006' as date, 1),
       add_months(cast '28-02-1999' as date, 12),
       add_months(cast '31-05-2006' as date, -1);

|28.02.2006:00:00:00|29.02.2000:00:00:00|30.04.2006:00:00:00|
```

Операции над значениями типа DATE

Над значениями типа DATE разрешены следующие операции:

1) бинарная операция сложения;
2) бинарная операция вычитания.

В бинарных операциях один из операндов должен иметь значение не полной даты, а отдельного элемента значения даты: только год, или месяц, или день и т.д. Например, при добавлении к дате '22.05.1998' 5 лет получится дата '22.05.2003'; при добавлении 9 месяцев получится дата '22.02.1998'; при добавлении 10 дней получится дата '01.06.1998'. При сложении двух полных дат, например, '22.05.1998' и '01.12.2000', результат будет непредсказуем.

Если к значению типа DATE добавляется (вычитается) просто число (без указания формата), то это число принимается за количество дней.

Значение лет (без указания дней, часов, минут, секунд и т.п.) из 2 цифр воспринимается как относительное (т.е. месяцы остаются без изменений, дни ставятся в 0), из 4 цифр – как абсолютное (т.е. дни и месяцы, если их нет, ставятся в 1). Сравните:

```
select to_date('03.02.2011', 'DD.MM.YYYY') - to_date('05','yy');

|03.02.2006:00:00:00.00|

select to_date('03.02.2011', 'DD.MM.YYYY') -
 to_date('0005','yyyy');

|02.02.2007:00:00:00.00|
```

Элементы конструкций могут быть заданы <SQL-параметром>, который должен содержать спецификацию типа данных параметра, например,

```
select ? (date) - to_date(? (char(2)),'yy');
```

```
16.03.2015
05

|16.03.2010:00:00:00.00|
```

Примеры

```
select sysdate + to_date('05','yy') from person;

select sysdate;

|27.03.2003:15:33:42.00 |

select sysdate+3;

|30.03.2003:15:34:40.00 |

select sysdate+4.56;

|01.04.2003:05:01:40.00 |

select to_date('15.10.2007 10:20','DD.MM.YYYY HH:MI') + 1.1e0;

|16.10.2007:12:44:00.00|
```

Интервальное время

Интервальное время позволяет представить промежуток времени между двумя датами.

Интервальное время может быть двух типов: интервал лет и месяцев и интервал дней, часов, минут и секунд.

Элементы конструкций для вычисления интервального времени могут быть заданы <SQL-параметром>, который должен содержать спецификацию типа данных параметра.

Для интервального времени разрешены следующие операции:

1) вычисление интервала времени между значениями типа «дата-время».

```
a)
create or replace table test("Начальная дата" date,
  "Промежуточная дата" date, "Конечная дата" date);

insert into test ("Начальная дата", "Промежуточная дата",
  "Конечная дата")
values (to_date('01.01.2010 12:00:00', 'DD.MM.YYYY HH:MI:SS'),
        to_date('02.01.2010 08:00:00', 'DD.MM.YYYY HH:MI:SS'),
        to_date('10.01.2010 18:00:00', 'DD.MM.YYYY HH:MI:SS'));
select to_char("Промежуточная дата" -
               "Конечная дата", 'dd') || ' дней' from test;
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
|08 дней|

select to_char("Начальная дата" - "Промежуточная дата", 'hh') ||
       ' часов' from test;

|20 часов|

select to_char("Начальная дата" - "Конечная дата", '+dd.mm')
 from test;

|-09.01|

select to_char("Конечная дата" - "Начальная дата", 'dd')
 from test;

|09|

select to_char("Начальная дата" + "Конечная дата", '+dd')
 from test;

|+12|

select to_char("Промежуточная дата" + "Начальная дата",
       'dd.mi.ss') from test;

|03.00.00|

b)
select to_char(:current_date (date) + :start_date (date),
       'dd.mm.yyyy');
01.01.2015
01.03.2014

|01.03.4028|
```

2) сложение/вычитание интервалов времени с числовыми значениями и интервалами времени.

```
a)
select to_char(("Промежуточная дата" - "Начальная дата") +
  ("Конечная дата" - "Промежуточная дата"), 'dd') from test;

|09|

b)
Блокада Ленинграда длилась с 8 сентября 1941 года 872 дня.
Узнать, когда она закончилась.
```

```
select 'Снятие блокады Ленинграда ' ||
       to_char((to_date('08.09.1941', 'dd.mm.yyyy') + 872),
       'dd.mm.yyyy');

|Снятие блокады Ленинграда 28.01.1944|

c)
select to_char(:current_date (date) + ? (int),'dd.mm.yyyy');
01.01.2015
200

|20.07.2015|
```

3) умножение/деление интервалов времени на числовое значение и на интервал времени.

```
a)
Пятилетку – досрочно, за 4 года и 2 месяца
(начало пятилетки '01.01.1980', завершение по плану '31.12.1984')

select 'Завершим пятилетку досрочно, за 4 года и 2 месяца, к ' ||
 to_char(to_date('01.01.1980', 'dd.mm.yyyy') +
         (to_date ('31.12.1984', 'dd.mm.yyyy') -
          to_date ('01.01.1980', 'dd.mm.yyyy')) * 50/60,
         'dd.mm.yyyy');

|Завершим пятилетку досрочно, за 4 года и 2 месяца, к 01.03.1984|

Вариант:
select 'Завершим пятилетку досрочно, за 4 года и 2 месяца, к ' ||
 to_char(to_date('01.01.1980', 'dd.mm.yyyy') +
         to_date ('02.04','mm.yy'), 'dd.mm.yyyy');

|Завершим пятилетку досрочно, за 4 года и 2 месяца, к 01.03.1984|

b)
Отец ребенка родился 25.03.1976, мать 19.09.1980, ребенок
 07.12.2006. Во сколько раз больше разница в годах между рождением
 ребенка и отца (матери)?
select round(cast (to_date('25.03.1976', 'dd.mm.yyyy') -
       to_date('07.12.2006', 'dd.mm.yyyy')) /
       (to_date('19.09.1980', 'dd.mm.yyyy') -
       to_date('07.12.2006', 'dd.mm.yyyy')) as decimal,2);

|                   1.17|
```

4) указание знака "-" перед значением интервала времени.

```
select - to_date('01:00', 'HH:MI') + to_date('02:00', 'HH:MI');
```

```
|00.00.0000:01:00:00|
```

Разница дат в месяцах (MONTHS_BETWEEN)

Функция

Вычисляет разницу в месяцах между двумя датами (из первой вычитает вторую).

Спецификация

[1] <синтаксис>::=
MONTHS_BETWEEN (<значимое выражение1>, <значимое выражение2>)
[2] <значимое выражение1>::= <значимое выражение>
[3] <значимое выражение2>::= <значимое выражение>

Синтаксические правила

1) <Значимое выражение1> и <значимое выражение2> должны иметь тип DATE.

Возвращаемое значение

1) Результат имеет тип DECIMAL.
2) Если дни месяца в датах одинаковы или являются последними днями месяца, то разница будет целое число, иначе разница будет дробное число (считая, что в месяце 31 день).
3) Если один из параметров NULL, то результат будет NULL.

Примеры

```
select MONTHS_BETWEEN('01.04.2002', '01.01.2002');
| 3.0|

select MONTHS_BETWEEN('30.04.2002', '24.01.2002');

|3.2258064516|
```

Преобразование числового значения в интервал времени (NUMTODSINTERVAL)

Функция

Преобразует число указанных единиц времени в значение типа DATE.

Спецификация

[1] <синтаксис>::=
NUMTODSINTERVAL (<значимое выражение>, <строка>)

Синтаксические правила

1) <Значимое выражение> имеет числовой тип (при необходимости производится неявное преобразование этого аргумента к типу DECIMAL).

2) <Строка> имеет строковый тип и указывает на тип единиц «дата-время»: 'DAY', 'HOUR', 'MINUTE' или 'SECOND'.

Возвращаемое значение

1) Результат имеет тип DATE.
2) Если один из параметров NULL, то результат NULL.

Примеры

```
SELECT NUMTODSINTERVAL (100, 'HOUR');

|  04.01.0001:04:00:00|

SELECT NUMTODSINTERVAL(1234567890, 'minute');

| 24.04.2348:19:30:00|
```

Преобразование строкового числового значения в интервал времени (TO_DSINTERVAL)

Функция

Преобразует строковое представление значения типа «дата-время» в тип данных DATE.

Спецификация

[1] <синтаксис>::= TO_DSINTERVAL (<строка>)

Синтаксические правила

1) <Строка> – строка в формате 'DDDD HH24:MI:SS'. Значение дней может содержать до 7 цифр. При превышении максимального значения года (9999) будет выдано непредсказуемое значение.

Возвращаемое значение

1) Результат имеет тип DATE.
2) Если параметр NULL, то результат NULL.

Пример

```
select TO_DSINTERVAL('736625 22:58:55');

|  23.10.2017:22:58:55 |
```

Работа с часовыми поясами

Преобразование времени по Гринвичу к локальному времени (TO_LOCALTIME)

Функция

Преобразование времени по Гринвичу к локальному времени.

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

Спецификация

[1] <синтаксис>::=
TO_LOCALTIME (<дата-время выражение>[, <часовой пояс>])
[2] <часовой пояс>::=
<временная_зона1>[<временная_зона2>, <дата_перехода_на_летнее время>[,/ <время>]
<дата_перехода_на_зимнее время>[,/ <время>]]
[3] <временная_зона1>::= 'GMT <смещение>'
[4] <временная_зона2>::= 'GMT [<смещение>]' | 'DST [<смещение>]'
[5] <смещение>::= '[+|-]hh[:mm[:ss]]'
[6] <время>::= '[+|-]hh[:mm[:ss]]'
[7] <дата_перехода_на_летнее время>::= <дата>
[8] <дата_перехода_на_зимнее время>::= <дата>
[9] <дата>::= 'M<месяц>.<неделя>.<день_недели>'
[10] <месяц>::= 1-12
[11] <неделя>::= 1-5
[12] <день_недели>::= 0-6

Синтаксические правила

1) `<Дата-время выражение>` – значение, воспринимаемое как дата (время) нулевого часового пояса (по Гринвичу).

2) Если аргумент `<часовой пояс>` равен `NULL` или не задан, то по умолчанию преобразование выполняется к часовому поясу, установленному на компьютере.

```
select to_localtime(to_date('12.45.00', 'hh.mi.ss'));

|00.00.0000:16:45:00.00|
```

Примечание
В данном случае результат может оказаться неверным, т.к. для формирования времени функция использует *текущее* значение флага летнего времени в ОС, а не то, которое должно соответствовать дате, указанной в качестве аргумента `<дата-время выражение>`.

3) Параметр `<временная_зона1>` задает номер часового пояса для зимнего времени.

```
Получить зимнее локальное время в 3 часовой зоне (Москва):
select to_localtime(to_date('12.45.00', 'hh.mi.ss'), 'gmt+3');

|00.00.0000:15:45:00.00|
```

4) Параметр `<временная_зона2>` задает номер часового пояса для летнего времени. Обычно отличается от `<временной_зоны1>` на 1 час, т.е. `<временная_зона2>=<временная_зона1>+1`.

5) Для получения летнего времени необходимо указать дату и, при необходимости, время перехода на летнее и на зимнее время в этом часовом поясе.

```
Например, узнать локальное летнее время в 3 часовом поясе (дата
 перехода на
 летнее время 30 марта в 02.00 часа ночи, на зимнее 30 октября в
 02.00 часа ночи):
select to_localtime(to_date('14.30.00','hh.mi.ss'),' GMT+3:00:00
 GMT+04:00:00,
```

```
 M3.5.0/2:00:00, M10.5.0/02:00:00 ');

|00.00.0000:17:30:00.00|

select to_localtime(to_date('28.05.2012','DD.MM.YYYY'),
'GMT+3DST+1,M3.5.0/2,M10.5.0/2');
```

6) Конструкция DST[<смещение>] задает смещение относительно <временной_зоны1>.

```
select to_localtime
 (to_date('14.30.00','hh.mi.ss'),'GMT+3DST+1,M3.5.0/2,M10.5.0/2');
эквивалентно
select to_localtime(to_date('14.30.00','hh.mi.ss'),' GMT+3:00:00
 GMT+04:00:00,
 M3.5.0/2:00:00, M10.5.0/02:00:00 ');

|00.00.0000:17:30:00.00|
```

7) Если задана только <временная_зона1>, а <временная_зона2> не задана, то подразумевается, что переход на летнее/зимнее время отсутствует.

8) В случае если задана <временная_зона2>, но не задано <смещение> для 'GMT' или <смещение> для 'DST', подразумевается, что оба эти смещения отличаются от <смещения> из <временной_зоны1> на 1 час.

```
select to_char(sysdate,'hh:mm:ss'),
to_char(to_gmtime(sysdate, 'GMT+3'),'hh:mm:ss'),
to_char(to_gmtime(sysdate, 'GMT+3
 GMT,M3.5.0,M10.5.0'),'hh:mm:ss');

|11:08:44|08:08:44|07:08:44|
```

9) <Смещение> задает нужную часовую зону в прямом или обратном (в зависимости от <знака>) направлении от Гринвичского меридиана. Допустимое значение в интервале от '-14:00:00' до '+14:00:00'.

```
select to_char(sysdate,'hh:mm:ss'),
to_char(to_localtime(sysdate, 'gmt+10'),'hh:mm:ss');

|09:08:36|19:08:36|
```

10) Знаки пробела игнорируются как внутри <смещения>, так и между элементами GMT, <знак> и <смещение>.

```
select to_localtime(to_date('31.12.2125','DD.MM.YYYY'),'gmt + 1 1
 ');

|31.12.2125:11:00:00.00|
```

11) Знак положительного смещения можно не задавать (используется по умолчанию).

```
select to_localtime(to_date('31.12.2125','DD.MM.YYYY'),'GMT4');

|31.12.2125:04:00:00.00|
```

12) Префикс GMT регистронезависим.

```
select to_localtime(to_date('31.12.2125','DD.MM.YYYY'),'GmT-1');
```

13) Префикс M регистрозависим.

```
Правильная конструкция
select to_localtime(to_date('14.30.00','hh.mi.ss'),
  'GMT+3DST,M3.5.0,M10.5.0');
Неправильная конструкция
select to_localtime(to_date('14.30.00','hh.mi.ss'),
  'GMT+3DST,m3.5.0,m10.5.0');
```

14) Нули внутри конструкции <часовой пояс> игнорируются, т.е. 'GMT+3' равнозначно 'GMT+03'.

15) <Дата_перехода_на_летнее время>, <дата_перехода_на_зимнее время> – даты перехода на летнее и на зимнее время соответственно.

16) <Время> – время перехода на летнее или на зимнее время. Если оно не задано, по умолчанию используется значение '02:00:00'.

17) Знак '+' у <смещения> означает, что часовой пояс лежит к востоку от Гринвича, '-' – к западу от Гринвича.

Примечание
Часовой пояс Москвы '+3'/'+4' для зимнего/летнего времени соответственно.

18) В параметре <неделя> число 5 означает последний день недели в месяце, даже если в месяце всего 4 таких дня недели.

19) В параметре <день> отсчет дней начинается с воскресенья, т.е. 0 – воскресенье.

20) Сумма смещений из <временной зоны1> и конструкции DST <смещение> должна находиться в интервале от '-14:00:00' до '+14:00:00'.

21) Аргументы могут быть заданы <SQL-параметром>, который должен содержать спецификацию типа данных параметра.

```
select to_localtime(? (date),? (char(10)));
30.12.2015:23:00:00
GMT-2

|30.12.2015:21:00:00.00|
```

22) Поддерживаются следующие условные наименования временных зон (таблица 6).

Таблица 6. Условные наименования временных зон

Условное наименование	Значение	Комментарий
AST	GMT-4	Atlantic Standard Time
ADT	GMT-3	Atlantic Daylight Time
BST	GMT-11	Bering Standard Time
BDT	GMT-10	Bering Daylight Time
CST	GMT-6	Central Standard Time
CDT	GMT-5	Central Daylight Time

Условное наименование	Значение	Комментарий
EST	GMT-5	Eastern Standard Time
EDT	GMT-4	Eastern Daylight Time
HST	GMT-10	Alaska-Hawaii Standard Time
HDT	GMT-9	Alaska-Hawaii Daylight Time
MST	GMT-7	Mountain Standard Time
MDT	GMT-6	Mountain Daylight Time
NST	GMT-3.5	Newfoundland Standard Time
PST	GMT-8	Pacific Standard Time
PDT	GMT-7	Pacific Daylight Time
YST	GMT-9	Yukon Standard Time
YDT	GMT-8	Yukon Daylight Time

Для всех временных зон используются те же максимальное и минимальное смещение, что и для GMT: от '-14:00:00' до '+14:00:00'.

```
select to_gmtime(to_date('01.01.2015 00:00','DD.MM.YYYY
 HH:MI'),'HST+0');

|31.12.2014:14:00:00.00|

select to_gmtime(to_date('01.01.2015 00:00','DD.MM.YYYY HH:MI'),?
 (char(10)));
YDT+5

|31.12.2014:11:00:00.00|
```

Общие правила

1) Интервал времени между GMT и локальным временем вычисляется заново через каждые 30 секунд.
2) Если в момент выполнения функции происходит перевод времени (например, времени местного московского 2009-03-29* 02*:00:00 уже не существует), то возвращается ближайшее существующее.

```
select '2009-03-28 23:00' "gmt",
  TO_LOCALTIME(to_date('2009-03-28 23:00','YYYY-MM-DD HH:MI'),
    'GMT+3DST,M3.5.0,M10.5.0');

gmt
 ---
|2009-03-28 23:00|29.03.2009:03:00:00.00|
```

Возвращаемое значение

Значение типа DATE, преобразованное из даты по Гринвичу в локальную дату (время) заданного часового пояса в летний или зимний период времени.

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

Примеры

1) Получить разницу в часовых поясах между местным временем и временем по Гринвичу:

```
select datesplit(to_localtime(sysdate),'hh') -
 datesplit(sysdate,'hh');

|    3|
```

2)

```
select to_localtime(to_date('31.12.2125','DD.MM.YYYY'),'GMT-1');

|30.12.2125:23:00:00.00|
```

3)

```
select to_localtime(to_date('31.12.2125','DD.MM.YYYY'),'GMT+10')
union all
select to_localtime(to_date('31.12.2125','DD.MM.YYYY'),'GMT10');

|31.12.2125:10:00:00.00|
|31.12.2125:10:00:00.00|
```

4) Переход на зимнее время:

```
select to_localtime(to_date('27.03.2010','DD.MM.YYYY'),
' GMT+3:00:00  GMT+04:00:00, M3.5.0/2:00:00, M10.5.0/02:00:00 ');
|27.03.2010:03:00:00.00|
```

```
select to_localtime(to_date('27.03.2010','DD.MM.YYYY'),
  'GMT+3DST+1,M3.5.0/2,M10.5.0/2');

select to_localtime(to_date('27.03.2010','DD.MM.YYYY'),
  'GMT+3GMT,M3.5.0,M10.5.0');
```

Преобразование локального времени ко времени заданного часового пояса (TO_GMTIME)

Функция

Преобразование локального времени ко времени заданного часового пояса.

Спецификация

[1] <синтаксис>::=
TO_GMTIME (<дата-время выражение>[, <часовой пояс>])
[2] <часовой пояс>::=
<временная_зона1>[<временная_зона2>, <дата_перехода_на_летнее время>[,/ <время>]
<дата_перехода_на_зимнее время>[,/ <время>]]
[3] <временная_зона1>::= 'GMT <смещение>'
[4] <временная_зона2>::= 'GMT [<смещение>]' | 'DST [<смещение>]'
[5] <смещение>::= '[+|-]hh[:mm[:ss]]'
[6] <время>::= '[+|-]hh[:mm[:ss]]'
[7] <дата_перехода_на_летнее время>::= <дата>

[8] <дата_перехода_на_зимнее время>::= <дата>
[9] <дата>::= 'M<месяц>.<неделя>.<день_недели>'
[10] <месяц>::= 1-12
[11] <неделя>::= 1-5
[12] <день_недели>::= 0-6

Синтаксические правила

1) <Дата-время выражение> – значение, воспринимаемое как дата (время) локального часового пояса.
2) Если аргумент <часовой пояс> равен NULL или не задан, то по умолчанию преобразование выполняется к нулевому часовому поясу (по Гринвичу).

```
select sysdate, to_gmtime(sysdate, null);

|16.08.2010:11:05:20.81|16.08.2010:07:05:20.81|
```

Примечание
В данном случае результат может оказаться неверным, т.к. для формирования времени функция использует *текущее* значение флага летнего времени в ОС, а не то, которое должно соответствовать дате, указанной в качестве аргумента <дата-время выражение>.

3) Параметр <временная_зона1> задает номер часового пояса для зимнего времени.

```
Получить чему соответствует данное локальное время в 3 часовой
 зоне (Москва):
select to_GMTIME(to_date('12.45.00', 'hh.mi.ss'), 'gmt+3');

|00.00.0000:09:45:00.00|
```

4) Параметр <временная_зона2> задает номер часового пояса для летнего времени.

   Обычно отличается от <временной_зоны1> на 1 час, т.е. <временная_зона2>=<временная_зона1>+1.

5) Для получения летнего времени необходимо указать дату и при необходимости время перехода на летнее и на зимнее время в этом часовом поясе.

```
Например, узнать, чему соответствует данное локальное время в 3
 часовом поясе
 (дата перехода на летнее время 30 марта в 02.00 часа ночи, на
 зимнее время – 30 октября в 02.00 часа ночи).
select to_gmtime(to_date('14.30.00','hh.mi.ss'),
  'GMT+3:00:00 GMT+04:00:00, M3.5.0/2:00:00, M10.5.0/02:00:00');

|00.00.0000:11:30:00.00|
```

6) Конструкция DST[<смещение>] задает смещение относительно <временной_зоны1>.

```
select to_gmtime(to_date('14.30.00','hh.mi.ss'),
  'GMT+3DST+1,M3.5.0/2,M10.5.0/2');
эквивалентно
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
select to_gmcaltime(to_date('14.30.00','hh.mi.ss'),
  'GMT+3:00:00 GMT+04:00:00, M3.5.0/2:00:00, M10.5.0/02:00:00');

|00.00.0000:17:30:00.00|
```

7) Если задана только <временная_зона1>, а <временная_зона2> не задана, то подразумевается, что переход на летнее/зимнее время отсутствует.

8) <Смещение> задает нужную часовую зону в прямом или обратном (в зависимости от знака) направлении от локального часового пояса. Допустимое значение в интервале от '-14:00:00' до '+14:00:00'.

```
select to_char(sysdate,'hh:mm:ss'),  to_char(to_gmtime(sysdate,
 'gmt+10'),'hh:mm:ss');

|11:08:34|01:08:34|
```

9) Синтаксис аргумента <часовой пояс> аналогичен этому же аргументу в функции TO_LOCALTIME.

10) Если задана только <временная_зона1>, а <временная_зона2> не задана, то подразумевается, что переход на летнее/зимнее время отсутствует.

11) В случае если задана <временная_зона2>, но не задано <смещение> для 'GMT' или <смещение> для 'DST', подразумевается, что оба эти смещения отличаются от <смещения> из <временной_зоны1>на 1 час.

```
select to_char(sysdate,'hh:mm:ss'),
to_char(to_gmtime(sysdate, 'GMT+3'), 'hh:mm:ss'),
to_char(to_gmtime(sysdate, 'GMT+3 GMT,M3.5.0,M10.5.0'),
  'hh:mm:ss');

|11:08:44|08:08:44|07:08:44|
```

12) <Дата_перехода_на_летнее время>, <дата_перехода_на_зимнее время> – даты перехода на летнее и на зимнее время соответственно.

13) <Время> – время перехода на летнее или на зимнее время. Если оно не задано, по умолчанию используется значение '02:00:00'.

14) Знак '+' у <смещения> означает, что часовой пояс лежит к востоку от Гринвича, '-' – к западу от Гринвича.

Примечание
Часовой пояс Москвы '+3'/'+4' для зимнего/летнего времени соответственно.

15) В параметре <неделя> число 5 означает последний день недели в месяце, даже если в месяце всего 4 таких дня недели.

16) В параметре <день> отсчет дней начинается с воскресенья, т.е. 0 – воскресенье.

17) Сумма смещений из <временной_зоны1> и конструкции DST <смещение> должна находиться в интервале от '-14:00:00' до '+14:00:00'.

18) Аргументы могут быть заданы <SQL-параметром>, который должен содержать спецификацию типа данных параметра.

```
select to_gmtime(? (date),? (char(30)));
30.12.2015:23:00:00
```

```
GMT+3DST+1,M3.5.0/2,M10.5.0/2

|30.12.2015:20:00:00.00|
```

19) Поддерживаются условные наименования временных зон (таблица 6).

Общие правила

1) Интервал времени между GMT и локальным временем вычисляется заново через каждые 30 секунд.

Возвращаемое значение

Значение типа DATE, преобразованное из локальной даты в дату (время) заданного часового пояса в летний или зимний период времени.

Примеры

1)

```
Самолет вылетел из Иркутска 28.12.2007 в 15.45.00 по местному
 времени (8 часовой пояс) в Воронеж (3 часовой пояс).
Длительность полета 3 час 47 мин.
Узнать местное время прибытия самолета.
select
'Время отправления: '+
to_char(to_localtime(to_date('28.12.2007.15.45',
 'dd.mm.yyyy.hh.mi'), 'gmt+0'), 'dd.mm.yyyy.hh.mi')
union
select  'Время прибытия:    '+
to_char(to_gmtime(to_localtime(to_date('28.12.2007.15.45',
 'dd.mm.yyyy.hh.mi'), 'gmt+0')
+to_date('03','hh')
+ to_date('47','mi'),'gmt+5'),'dd.mm.yyyy.hh.mi');

|Время отправления: 28.12.2007.15.45|
|Время прибытия:    28.12.2007.14.32|
```

2)

```
Варианты задания часового пояса Москвы:
'GMT+03:00:00GMT+04:00:00,M3.5.0/02:00:00,M10.5.0/02:00:00'
'GMT+3DST+1,M3.5.0/2,M10.5.0/2'
'GMT+3DST,M3.5.0,M10.5.0'
```

Функции для работы с IP-адресами

Бесклассовая междоменная маршрутизация (Classless Inter-Domain Routing (CIDR)) – способ назначения и указания Internet-адресов, используемый в междоменной маршрутизации. Он более гибок, чем первоначальная система классов адресов, принятая в Internet-протоколе (IP). При использовании CIDR каждый IP-адрес имеет сетевой префикс, который определяет либо совокупность сетевых узлов, либо

индивидуальный узел. Длина сетевого префикса задаётся как часть IP-адреса и варьируется в зависимости от количества необходимых бит. IP-адрес пункта назначения или маршрут, который указывает несколько адресатов, имеет короткий префикс (менее точная адресация). Длинный префикс описывает узел сети более точно.

Внешнее представление сетевого адреса при использовании CIDR выглядит так:

```
192.30.250.00/18
```

«192.30.250.00» есть сам сетевой CIDR IP-адрес. «18» означает, что первые 18 бит есть сетевая часть адреса, оставшиеся 14 бит задают адреса хостов.

Внутреннее представление IP-адреса – BYTE(5). Из них 4 байта занимает собственно сетевой адрес, а 1 байт отводится под размер префикса. Значение префикса может быть в диапазоне от 1 до 32.

Аргументы функций могут быть заданы <SQL-параметром>, который должен содержать спецификацию типа данных параметра, например:

```
select cidrtoraw(? (char(16)));
192.30.250.00/18

| C0 1E FA 00 12|

select ip_set_masklen(? (byte(5)), ? (int));
5052200b0a
21

| 50 52 20 0B 15|
```

Преобразование внешнего представления CIDR IP-адреса во внутреннее (CIDRTORAW)

Функция

Преобразование внешнего представления CIDR IP-адреса во внутреннее.

Спецификация

[1] <синтаксис> : : = CIDRTORAW (<символьное выражение>)

Синтаксические правила

1) <Символьное выражение> должно иметь тип CHAR или VARCHAR с длиной от 9 до 18 символов включительно в формате IP-адреса.

Возвращаемое значение

1) Внутреннее представление CIDR IP-адреса. Тип возвращаемого значения – BYTE(5). Из них первые 4 байта занимает собственно сетевой адрес, 5-й байт – размер префикса.

Примеры

```
select cidrtoraw('80.82.32.00/10');
```

```
| 505220000A |
```

```
create or replace table tab1 (cidr varchar(18));
insert into tab1 values('80.82.32.00/10');
select cidrtoraw(cidr) from tab1;
```

Преобразование внутреннего представления CIDR IP-адреса во внешнее (RAWTOCIDR/IP_TEXT)

Функция

Преобразование внутреннего представления CIDR IP-адреса во внешнее.

Спецификация

[1] <синтаксис>::= {RAWTOCIDR |IP_TEXT} (<байтовое выражение>)

Примечание
Функция `IP_TEXT` реализована для совместимости с PostgreSQL 7.3.

Синтаксические правила

1) `<Байтовое выражение>` должно иметь тип BYTE(5).

Возвращаемое значение

1) Внешнее представление CIDR IP-адреса.
2) Тип возвращаемого значения – `VARCHAR(18)`.

Примеры

```
select rawtocidr (cidrtoraw('80.82.32.00/10')), ip_text
 (HEX('5052200B15'));

| 80.82.32.11/21| 80.82.32.11/21|
```

```
select rawtocidr(HEX('5052200B15'));

|80.82.32.11/21    |
```

```
create or replace table tab2 (cidr byte(5));
insert into tab2 values(hex('8082320010'));
select rawtocidr( cidr ) from tab2;

| 128.130.50.0/16 |
```

Определение длины префикса CIDR IP-адреса (CIDRLEN/IP_MASKLEN)

Функция

Определение длины префикса CIDR IP-адреса.

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

Спецификация
[1] <синтаксис>::= {CIDRLEN |IP_MASKLEN} (<байтовое выражение>)

> **Примечание**
> Функция IP_MASKLEN реализована для совместимости с PostgreSQL 7.3.

Синтаксические правила

1) <Байтовое выражение> должно иметь тип BYTE(5).

Возвращаемое значение

1) Длина префикса CIDR IP-адреса.
2) Тип возвращаемого значения – INT.

Примеры

```
select
cidrlen (HEX('8082320010')), ip_masklen (HEX('8082320010'));
select cidrlen (cast HEX('8082320010') as byte(5));

| 16 |16|
```

```
create or replace table tab2 (cidr byte(5));
insert into tab2 values(hex('8082320010'));
select cidrlen( cidr ) from tab2;

|16 |
```

Установка новой длины префикса CIDR IP-адреса (IP_SET_MASKLEN)

Функция

Установка новой длины префикса и формирование нового значения CIDR IP-адреса.

Спецификация
[1] <синтаксис>::=
IP_SET_MASKLEN (<байтовое выражение>, <числовое выражение>)

Синтаксические правила

1) <Байтовое выражение> должно иметь тип BYTE(5).
2) <Числовое выражение> должно иметь тип INTEGER, SMALLINT или BIGINT.
3) Значение <числового выражения> задает новую длину префикса CIDR IP-адреса и должно быть в диапазоне от 1 до 32.

Возвращаемое значение

1) Новый CIDR IP-адрес.
2) Тип возвращаемого значения – BYTE(5).

Примеры

```
select ip_set_masklen(hex('5052200b0a'), 21);

| 50 52 20 0B 15 |

select rawtocidr(ip_set_masklen(cidrtoraw('80.82.32.11/10'), 21));

|80.82.32.11/21|
```

Определение CIDR IP-адреса без длины префикса (CIDRPFX)

Функция

Определение CIDR IP-адреса без длины префикса.

Спецификация

[1] <синтаксис>::= CIDRPFX (<байтовое выражение>)

Синтаксические правила

1) <Байтовое выражение> должно иметь тип BYTE(5).

Возвращаемое значение

1) Сетевой IP-адрес без префикса. Все биты CIDR IP-адреса после префикса обнуляются.
2) Формирование сетевого адреса выполняется по следующему алгоритму:
   - шестнадцатеричное представление сетевого адреса преобразуется в двоичное представление, например, 5052200B15 преобразовать в двоичное (в данной функции – без префикса 15): 01010000 01010010 10000000 00001011
   - префикс имеет значение 15 (шестнадцатеричное), что равно значению 21 (десятичное), т.е. длина префикса в сетевом адресе начинается с 21 позиции в двоичном сетевом адресе: 01010000010100101000000 00001011
   - т.к. требуется получить сетевой адрес без префикса, то обнулятся все биты сетевого адреса, начиная с 21 бита: 0101 0000 1010 01010000000 00000000
   - полученное значение представляет искомый сетевой адрес в шестнадцатеричном виде: 5052100000000000000
3) Тип возвращаемого значения – VARCHAR(15).

Примеры

```
select cidrpfx (cast HEX('5052200B15') as byte(5));

| 80.82.32.0|

create or replace table tab2 (cidr byte(5));
insert into tab2 values(hex('8082320010'));
select cidrpfx( cidr ) from tab2;

| 128.130.0.0 |
```


© Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

Определение CIDR IP-адреса для совместимости с PostgreSQL (IP_HOST)

Функция

Определение CIDR IP-адреса для совместимости с PostgreSQL.

> **Примечание**
> Функция реализована для совместимости с PostgreSQL 7.3.

Спецификация

[1] <синтаксис>::= IP_HOST (<байтовое выражение>)

Синтаксические правила

`<Байтовое выражение>` должно иметь тип `BYTE(5)`.

Возвращаемое значение

1) Сетевой CIDR IP-адрес без префикса.
2) Тип возвращаемого значения – `VARCHAR(15)`.

Пример

```
select ip_host ( HEX('5052200B0a'));

| 80.82.32.11|
```

Проверка адреса по адресной маске (CIDRMATCH)

Функция

Проверка принадлежности адреса к указанной сети по адресной маске.

Адресная маска – битовая маска, используемая для выбора бит из IP-адреса с целью адресации подсети. Маска имеет размер 32 бита и выделяет сетевую часть IP-адреса и один или несколько бит локальной части адреса.

Спецификация

[1] <синтаксис>::=
CIDRMATCH (<байтовое выражение 1>, <байтовое выражение 2>, <маска>)
[2] <маска>::= <байтовый литерал>|<двоичный литерал>

Синтаксические правила

1) `<Байтовое выражение 1>` должно иметь тип `BYTE(5)`, `<байтовое выражение 2>` и `<маска>` должны иметь тип `BYTE(4)`.

Возвращаемое значение

1) Значение `TRUE`, если CIDR IP-адрес из `<байтового выражения 1>` идентичен по `<маске>` адресу `<байтового выражения 2>`, `FALSE` – в противном случае. Идентификация адресов выполняется путем побитового сравнения `<байтового`

выражения 1> и <байтового выражения 2>. Порядковые номера сравниваемых бит соответствуют порядковым номерам нулевых бит в <маске>.

2) Тип возвращаемого значения – BOOLEAN.

Примеры

```
select cidrmatch(cidrtoraw('80.82.32.1/32'),
      cast cidrtoraw('80.82.32.00/10')  as byte(4),
      cast hex('000000ff')  as byte(4));
```

или

```
select cidrmatch(cidrtoraw('80.82.32.1/32'),
      cast cidrtoraw('80.82.32.00/10')  as byte(4),
      cast cidrtoraw('0.0.0.255/8')  as byte(4));

|T|
```

```
create or replace table tab2 (cidr byte(5));
insert into tab2 values(hex('8082320010'));
select cidrmatch( cidr, hex('80500000'), hex('00ffffff') ) from
 tab2;

| T|
```

```
select cidrmatch( cidr, hex('80500000'), hex('ff000000') ) from
 tab2;

| F|
```

Определение количества IP-адресов (CIDRCOUNT)

Функция

Определение количества IP-адресов.

Спецификация

[1] <синтаксис>::= CIDRCOUNT (<байтовое выражение>)

Синтаксические правила

1) <Байтовое выражение> должно иметь тип BYTE(5).

Возвращаемое значение

1) Допустимое количество IP-адресов, которые могут быть указаны в CIDR IP-адресе с данным префиксом. Длина префикса 32 определяет единственный сетевой адрес, длина 1 задает максимально возможное количество IP-адресов.
2) Тип возвращаемого значения – INT.



Примечание
Если CIDR IP-адрес имеет префикс 1, то происходит переполнение типа данных INT, и в таком случае фиксируется код завершения «Переполнение целого числа».

Примеры

```
select cidrcount(cast hex('5052200b18') as byte(5));

|256|

select cidrcount(cast hex('5052200b15') as byte(5));

|2048|

select cidrcount(cidrtoraw('80.82.32.11/32'));

| 1|

select cidrcount(cidrtoraw('80.82.32.11/2'));

| 1073741824|
```

Выделение сетевого адреса со сбросом бит префикса (CIDRNET)

Функция

Выделение сетевого адреса со сбросом бит префикса.

Спецификация

[1] <синтаксис>::= CIDRNET (<байтовое выражение>)

Синтаксические правила

1) <Байтовое выражение> должно иметь тип BYTE(5).

Возвращаемое значение

1) Конечный сетевой IP-адрес узла, задаваемый CIDR IP-адресом. Биты адреса, отведенные под префикс, установлены в 0.
2) Тип возвращаемого значения – BYTE(4).

Примеры

```
select cidrnet(hex('5052200b0a'));

|50400000|

create or replace table tab2 (cidr byte(5));
insert into tab2 values(hex('8082320010'));
select ltrim(to_char(getbyte(cidrnet(cidr), 0), '999')) || '.' ||
       ltrim(to_char(getbyte(cidrnet(cidr),1), '999')) || '.' ||
       ltrim(to_char(getbyte(cidrnet(cidr),2), '999')) || '.' ||
       ltrim(to_char(getbyte(cidrnet(cidr),3), '999')) from tab2;

| 128.130.0.0 |
```

Выделение сетевого адреса (IP_NETWORK)

Функция

Выделение сетевого адреса.

Спецификация

[1] <синтаксис>::= IP_NETWORK (<байтовое выражение>)

Синтаксические правила

1) <Байтовое выражение> должно иметь тип BYTE(5).

Возвращаемое значение

1) Конечный сетевой IP-адрес узла, задаваемый CIDR IP-адресом. Биты адреса после префикса установлены в 0.
2) Тип возвращаемого значения – BYTE(5).

Примеры

```
select ip_network(hex('5052200b0a'));

|504000000A|
select rawtocidr(ip_network(cidrtoraw('80.82.32.11/10')));

| 80.64.0.0/10 |
```

Выделение сетевого адреса с установкой бит префикса (CIDRBCAST)

Функция

Выделение сетевого адреса с установкой бит префикса.

Спецификация

[1] <синтаксис>::= CIDRBCAST (<байтовое выражение>)

Синтаксические правила

1) <Байтовое выражение> должно иметь тип BYTE(5).

Возвращаемое значение

1) Конечный сетевой IP-адрес узла, задаваемый CIDR IP-адресом. Биты адреса, отведенные под префикс, установлены в 1.
2) Тип возвращаемого значения – BYTE(4).

Примеры

```
select cidrbcast(hex('5052200b0a'));

| 507FFFFF |
```


© Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
create or replace table tab2 (cidr byte(5));
insert into tab2 values(hex('8082320010'));
select ltrim(to_char(getbyte(cidrbcast(cidr), 0), '999')) || '.'
 ||
       ltrim(to_char(getbyte(cidrbcast(cidr),1), '999')) || '.' ||
       ltrim(to_char(getbyte(cidrbcast(cidr),2), '999')) || '.' ||
       ltrim(to_char(getbyte(cidrbcast(cidr),3), '999')) from
 tab2;

|128.130.255.255|
```

Выделение сетевого адреса с установкой бит префикса для совместимости с PostgreSQL (IP_BROADCAST)

Функция

Выделение сетевого адреса с установкой бит префикса.

Примечание
Функция реализована для совместимости с PostgreSQL 7.3.

Спецификация

[1] <синтаксис>::= IP_BROADCAST (<байтовое выражение>)

Синтаксические правила

1) <Байтовое выражение> должно иметь тип BYTE(5).

Возвращаемое значение

1) Конечный сетевой IP-адрес узла, задаваемый CIDR IP-адресом. Биты адреса, отведенные под префикс, установлены в 1.
2) Тип возвращаемого значения – BYTE(5).

Примеры

```
select ip_broadcast(hex('5052200b0a'));

| 507FFFFF0A |
```

```
create or replace table tab2 (cidr byte(5));
insert into tab2 values(hex('8082320010'));
select ltrim(to_char(getbyte(ip_broadcast(cidr), 0), '999')) ||
 '.' ||
       ltrim(to_char(getbyte(ip_broadcast(cidr),1), '999')) || '.'
 ||
       ltrim(to_char(getbyte(ip_broadcast(cidr),2), '999')) || '.'
 ||
       ltrim(to_char(getbyte(ip_broadcast(cidr),3), '999')) || '/'
 ||
```

```
       ltrim(to_char(getbyte(ip_broadcast(cidr),4), '999'))
  from tab2;

|128.130.255.255/16|
```

Выделение маски сети (CIDRMASK/IP_NETMASK)

Функция

Выделение маски сети.

Спецификация

[1] <синтаксис>::=
{CIDRMASK | IP_NETMASK} (<байтовое выражение>)

Примечание
Функция `IP_NETMASK` реализована для совместимости с PostgreSQL 7.3.

Синтаксические правила

1) `<Байтовое выражение>` должно иметь тип `BYTE(5)`.

Возвращаемое значение

1) Маска сети, задаваемая CIDR IP-адресом (все биты до префикса установлены в 1, биты префикса установлены в 0).
2) Тип возвращаемого значения – `BYTE(4)`.

Примеры

```
select cidrmask(cidrtoraw('80.82.32.11/10')),
 ip_netmask(cidrtoraw('80.82.32.11/10'));

|FFC00000|FFC00000|
```

```
create or replace table tab2 (cidr byte(5));
insert into tab2 values(hex('8082320010'));
select cidrmask(cidr) from tab2;

|FFFF0000 |
```

```
select cidrmatch( cidr, hex('80500000'), cidrmask(cidr)) from
 tab2;

|FALSE|
```

Выделение сокращенного IP-адреса (IP_ABBREV)

Функция

Выделение сокращенного IP-адреса.

© Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

Спецификация

[1] <синтаксис>::= IP_ABBREV (<байтовое выражение>)

Синтаксические правила

1) <Байтовое выражение> должно иметь тип BYTE(5).

Возвращаемое значение

1) Сокращенный (т.е. без правосторонних нулевых байт) CIDR IP-адрес.
2) Тип возвращаемого значения – VARCHAR(18).

Пример

```
select ip_abbrev(cidrtoraw('80.82.00.00/10'));

| 80.82/10 |
```

Функции для работы с MAC-адресами

MAC (Media Access Control) – адрес, используемый системой управления сетевым доступом. Представляет собой уникальное 48-битовое число, отображаемое обычно в виде 12-значного шестнадцатеричного числа. MAC-адрес позволяет однозначно идентифицировать устройство в локальной сети (т.е. это сетевой аппаратный адрес устройства, подключенного к сети).

Примеры внешнего представления MAC-адресов:

- '08002b:010203';
- '08002b-010203';
- '0800.2b01.0203';
- '08-00-2b-01-02-03';
- '08:00:2b:01:02:03'.

Аргументы функций могут быть заданы <SQL-параметром>, который должен содержать спецификацию типа данных параметра, например:

```
select mactoraw(? (char(17)),? (char(17)));
08/00/2b/01/02/03
XX/XX/XX/XX/XX/XX

| 08 00 2B 01 02 03|
```

Преобразование внешнего представления MAC-адреса во внутреннее (MACTORAW)

Функция

Преобразование внешнего представления MAC-адреса во внутреннее.

Спецификация

[1] <синтаксис>::= MACTORAW (<символьное выражение>[, <формат>])
[2] <формат>::= <символьное выражение>

Синтаксические правила

1) <Символьное выражение> должно иметь тип CHAR или VARCHAR.
2) <Формат> задает формат представленного в <символьном выражении> MAC-адреса. Количество символов MAC-адреса (без учета пробелов) и символов формата должно строго соответствовать друг другу.
3) Формат имеет следующий шаблон:

   [#][…]x[…]

   где:

   - # – задает набор и представление символов-разделителей элементов MAC-адреса;
   - x (латинская кодировка) – задает количество шестнадцатеричных цифр, представляющих элемент MAC-адреса. Независимо от способа представления полного MAC-адреса шаблон формата должен содержать 12 знаков «x».

   Если <формат> не задан, по умолчанию применяются следующие форматы MAC-адресов:

   - xxxxxx;xxxxxx
   - xxxxxx-xxxxxx
   - xxxx.xxxx.xxxx
   - xx-xx-xx-xx-xx-xx
   - xx:xx:xx:xx:xx:xx

   Примеры возможных форматов:

Внешнее представление MAC-адреса	Формат
08002b/010203	xxxxxx/xxxxxx
008.00.02b/01:02:03	0xx.xx.0xx/xx:xx:xx
#08-00-2b-01.02.03	#xx-xx-xx-xx.xx.xx
<08><00><2b><01><02><03>	<xx><xx><xx><xx><xx><xx>
08 | 00 | 2b	xx|xx|xx
08002b:010203	xxxxxx:xxxxxx

Возвращаемое значение

1) Внутреннее представление MAC-адреса.
2) Код завершения 1036 («Неправильный аргумент»), если представление MAC-адреса и его формат не соответствуют друг другу.
3) Тип возвращаемого значения – BYTE(6).

Примеры

```
select mactoraw('08-00-2b-01-02-03');

|08002B010203|
```


© Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
select mactoraw('08/00/2b/01/02/03',
'XX/XX/XX/XX/XX/XX');

|08002B010203|

select mactoraw('008 | 000 |02b | 001 | 002 | 003', 'xxx|xxx|xxx|
xxx|xxx|xxx');

|00800002B001|

select rawtomac(mactoraw('08-00-2b-01-02-03'));

|08:00:2B:01:02:03|

create or replace table tab1 ("MAC-формат" varchar(20));
insert into tab1 values ('xx.xx.xx.xx.xx.xx');
select mactoraw('08.00.2b.01.02.03', "MAC-формат") from tab1;

|08002B010203|
```

Преобразование внутреннего представления MAC-адреса во внешнее (RAWTOMAC)

Функция

Преобразование внутреннего представления MAC-адреса во внешнее.

Спецификация

[1] <синтаксис>::= RAWTOMAC (<байтовое выражение>[, <формат>])

Синтаксические правила

1) <Байтовое выражение> должно иметь тип BYTE(6).
2) <Формат> задает формат внешнего представления MAC-адреса (см. функцию MACTORAW), при этом учитывается регистр представления цифры в шаблоне: «X» – представление буквы шестнадцатеричной цифры в верхнем регистре, «x» – в нижнем.
3) В соответствии с позицией каждого символа «x» или «X» в шаблоне считывается 8 бит информации из <выражения>. Если количество символов «x» или «X» превышает 12, оставшиеся символы игнорируются.
4) Если <формат> не указан, MAC-адрес выводится в виде 'XX:XX:XX:XX:XX:XX'.

Возвращаемое значение

1) Внешнее представление MAC-адреса в соответствии с заданным форматом.
2) Тип возвращаемого значения – VARCHAR(n), где n – длина шаблона <формата> (для шаблона по умолчанию n равно 17).

Примеры

```
select rawtomac (hex('08002b010203'));
```

```
|08:00:2B:01:02:03|
```

```
select rawtomac (hex('fa0c2b0102aa'), 'XX.xx.XX.xx.XX.xx');
```

```
|FA.0c.2B.01.02.aa|
```

```
select rawtomac(mactoraw('08-00-2b-01-02-03'));
```

```
|08:00:2B:01:02:03|
```

```
select mactoraw('008/000/02b/001/002/003',
 '0xx/0xx/0xx/0xx/0xx/0xx');
```

```
|08:00:2B:01:02:03|
```

Обнуление последних трех байт MAC-адреса (MAC_TRUNC)

Функция

Обнуление последних трех байт MAC-адреса.

Спецификация

[1] <синтаксис>::= MAC_TRUNC (<байтовое выражение>)

Синтаксические правила

1) <Байтовое выражение> должно иметь тип BYTE(6).

Возвращаемое значение

1) Внутреннее представление MAC-адреса с нулевым значением трех последних байт.
2) Тип возвращаемого значения – BYTE(6).

Примеры

```
select rawtomac(mac_trunc(mactoraw('08:00:2b:01:02:03')));
```

```
| 08:00:2B:00:00:00 |
```

```
select mac_trunc(hex('08002b010203')),
 rawtomac(mac_trunc(hex('08002b010203')));
```

```
|08002B000000|08:00:2B:00:00:00|
```

Прочие функции

Проверка разрешения доступа (ACCESS)

Функция

Проверка разрешения доступа.

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

Спецификация

[1] <синтаксис>::=
ACCESS (<имя пользователя>, <идентификатор таблицы>)
[2] <имя пользователя>::= <символьный литерал>
[3] <идентификатор таблицы>::= <целочисленный литерал>

Синтаксические правила

Примечания

1. <Идентификатор таблицы> – внутренний системный номер таблицы в БД. Идентификаторы таблиц хранятся в системной таблице $$$SYSRL в столбце с именем $$$S11, имена таблиц – в столбце $$$S13. Таким образом, для получения <идентификатора таблицы> уникальной в БД таблицы AUTO можно использовать запрос:

```
select $$$S11 from $$$SYSRL where $$$S13='AUTO';
```

2. Если в БД имеются таблицы с одинаковыми именами (разные схемы), необходимо использовать имя схемы таблицы. Список схем хранится в системной таблице $$$USR, столбец $$$S34.

```
select $$$S11 from $$$SYSRL, $$$USR
 where $$$s31 > 0 and $$$s31=$$$s12
   and $$$S13='AUTO' and $$$s34='SYSTEM';
```

1) Аргументы могут быть заданы <SQL-параметром>, который должен содержать спецификацию типа данных параметра.

```
select access(? (CHAR(6)), ? (int));
SYSTEM
176

|   SIDUAXR   |
```

2) <Целочисленный литерал> должен иметь положительное значение, при прочих значениях результат функции будет NULL.

Возвращаемое значение

1) Строка длиной 7 символов типа CHAR, содержащая коды доступных операций ('SIDUAXR' – SELECT, INSERT, DELETE, UPDATE, ALTER TABLE, CREATE INDEX, REFERENCES соответственно). Если операция недоступна, в соответствующей позиции строки ответа будет стоять пробел.

2) Если один из параметров имеет NULL-значение, то результат будет NULL-значение.

Примеры

1)

```
username SYSTEM/MANAGER8
create or replace user TEST identified by '12345678';
grant dba to TEST;
```

```
username TEST/12345678
create or replace table tst_acs (i int);

username SYSTEM/MANAGER8
select $$$S11 from $$$SYSRL where $$$S13='TST_ACS';

|    412|

username TEST/12345678
select access('SYSTEM', 412);

|       |

grant update on TST_ACS to SYSTEM;
select access('SYSTEM', 412);

|  U    |

revoke update on TST_ACS from SYSTEM;
select access('SYSTEM',412);

|       |
```

2) Получить список прав доступа на таблицу «Банки» пользователя «Петрова»:

```
select ACCESS($$$S34,$$$S11) from $$$SYSRL,$$$USR
 where $$$S12=$$$S31
   and $$$S34='Петров' and $$$S13='Банки';
```

3) Выдать таблицы, для работы с которыми у текущего пользователя есть хотя-бы одна разрешенная операция:

```
SELECT $$$S34, $$$S13
  FROM $$$SYSRL, $$$USR
 WHERE $$$S12 =$$$S31
   AND $$$S32 =0 AND ACCESS(USER, $$$S11) <> ' ';
```

Получить имя БД (DBNAME)

Функция

Получить имя БД.

Спецификация

[1] <синтаксис>::= DBNAME()

Возвращаемое значение

1) Возвращает 18-символьное имя БД, по соединению с которой подан запрос с данной функцией.

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

Пример

```
select dbname();

|DEMO DATABASE |
```

Получить дату окончания лицензионного периода СУБД (EXPIRATION_DATE)

Функция

Получить дату окончания лицензионного периода СУБД.

Спецификация

[1] <синтаксис>::= EXPIRATION_DATE()

Возвращаемое значение

1) Возвращает значение типа `DATE`;
2) Если лицензия до определенной даты – возвращает эту дату, иначе возвращает `NULL`.

Пример

```
select expiration_date();

|21.10.2017:00:00:00|
```

Игнорировать ошибку преобразования (NULLIFERROR)

Функция

Игнорирует ошибку преобразования, возвращая нормальный код завершения и NULL-значение.

Спецификация

[1] <синтаксис>::= NULLIFERROR(<значимое выражение>)

Общие правила

1) Функция NULLIFERROR игнорирует различные ошибки преобразования типов (1030-1033, 1040-1042).



Примечание
Игнорирование ошибок преобразования типов (1030-1033, 1040-1042) поддерживается со сборки 6.0.17.92.

```
create table tonum (c char(20));
insert into tonum values ('200');
insert into tonum values ('text');

select to_number(c) from tonum;
```

```
|                   200|
INL : execute status : 1042
character string to float conversion error

select nulliferror(to_number(c)) from tonum;

|                   200|
|                      |

select nvl(nulliferror(to_number(c)),0) from tonum;

|                   200|
|                     0|
```

2) Функция игнорирует код завершения 1036 («Значение аргумента в недопустимом диапазоне») при выполнении функций HEXTORAW, LINTER_FILE_INFO, LINTER_PAGE_INFO, LINTER_FILE_SIZE, LINTER_FILE_DEVICE и код завершения 1038 («Неверный формат преобразования даты») при выполнении ядром СУБД функций преобразования TO_DATE, TO_CHAR.

```
create table tnie(c char(20));
insert into tnie values ('2000');
select case nulliferror(to_date(c,'dd.mm.yyyy'))
  when null then 'null' end from tnie;

|null|
```

3) Если значение типа «дата» задано в неправильном формате в виде литерала, то в этом случае ошибка выявляется уже на стадии трансляции запроса и запрос до ядра СУБД не доходит, поэтому функция nulliferror не используется.

```
В нижеприведенных примерах генерируется код завершения трансляции
 2032 и дело до выполнения запроса (и, соответственно, вызова
 функции nulliferror) не доходит:
insert into test (dat) values (nulliferror('09.15.2001'));
insert into test (dat) values (nulliferror('39.05.2001'));
```

Возвращаемое значение

1) Возвращается <значимое выражение>, унаследованное от функций TO_DATE, TO_CHAR, HEXTORAW в случае корректного преобразования, и NULL-значение – в противном случае.

Пример

Практическая польза от функции nulliferror наблюдается в случае выборки записей, некоторые из которых потенциально могут содержать неверные данные. В этом случае процесс выборки записей не прекращается после выборки некорректной записи.

```
create table test(c char(20));
insert into test values ('01.01.1999');
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
insert into test values ('2000');
insert into test values ('22.09.2000');
```

Код завершения 1038 «Неверный формат преобразования даты» игнорируется, выборка данных не прерывается:

```
select nulliferror(to_date(c, 'dd.mm.yyyy')) from test;

|01.01.1999:00:00:00.00|
|                      |
|22.09.2000:00:00:00.00|
```

Код завершения 1038 «Неверный формат преобразования даты» не игнорируется, выборка данных прекращается после первой некорректной записи.

```
select to_date(c,'dd.mm.yyyy') from test;

|01.01.1999:00:00:00.00|
INL : состояние выполнения : 1038
Неверный формат преобразования даты
```

Замена значения (DECODE)

Функция

Замена значения в соответствии с правилом декодирования.

Спецификация

[1] <синтаксис>::=
DECODE (<значимое выражение>, <вариант1>, < значение1>[, …][, <значение по умолчанию>])
[2] <вариант...>::= <литерал>
[3] <значение по умолчанию>::=
<литерал>
|{SYSDATE | NOW}
|{LOCALTIME | LOCALTIMESTAMP}
|CURRENT_DATE
|{CURRENT_TIME | CURRENT_TIMESTAMP}
| NULL
| USER

Общие правила

1) Функция создана для совместимости с СУБД ORACLE. Эквивалентна конструкции СУБД ЛИНТЕР:

```
CASE <значимое выражение> WHEN <вариант1> THEN <значение1>
[...WHEN <вариантN> THEN <значениеN>]
[ELSE <значение по умолчанию>] END
```

2) Если <значение по умолчанию> не задано, используется NULL-значение.

3) Аргументы могут быть заданы <SQL-параметром>, который должен содержать спецификацию типа данных параметра.

```
select distinct decode(make, ? (char(20)), ? (char(20)),
  ? (char(20)), ? (char(20)), 'Прочие') from auto limit 5;
FORD
Форд
MASERATI
Мазерати

|Мазерати            |
|Прочие              |
|Форд                |
```

Примеры

```
create table tab1(code int);
…
select * from tab1;

| 1    |
| NULL |
| 2    |
| 4    |
| 3    |
```

```
select decode( code, 1, 'Директор', 2, 'Маркетинг',
  3, 'Бухгалтерия', 4, 'Программисты', null, 'Неизвестно')
  from tab1;
| Директор     |
| Неизвестно   |
| Маркетинг    |
| Программисты |
| Бухгалтерия  |
```

Одновариантная замена NULL-значения реальным значением (NVL)

Функция

Одновариантная замена неопределенного значения (NULL-значения) реальным значением.

Спецификация

[1] <синтаксис>::= NVL(<выражение1>, <выражение2>)
[2] <выражение1>::= <значимое выражение>|<подзапрос>
[3] <выражение2>::= <значимое выражение>|<подзапрос>

Общие правила

1) Функция возвращает значение <выражения1> в случае, если <выражение1> не является NULL-значением, иначе возвращает значение <выражения2>.
2) Функция эквивалентна конструкции СУБД ЛИНТЕР:

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
CASE WHEN <выражение1> IS NOT NULL
THEN <выражение1> ELSE <выражение2> END;
```

3) Типы данных <выражения1>, <выражения2> должны быть идентичны или приводимыми. При приведении типов данных по умолчанию (без конструкции CAST…) второй аргумент приводится к типу данных первого аргумента.

4) Аргументы могут быть заданы <SQL-параметром>, который должен содержать спецификацию типа данных параметра.

```
select nvl(? (char(5)), null);
111

|111  |

select nvl(? (char(5)), '222');
NULL

|222  |
```

5) Оба <подзапроса> должны содержать одно выбираемое <значимое выражение> и возвращать скалярное значение. Типы данных возвращаемых значений должны быть идентичными или приводимыми друг к другу.

```
create or replace table tst2 (id int, ch char(5));
insert into tst2(id, ch) values (1,'aaa'),(2,'bbb'), (3,NULL);

create or replace table tst1 (id int, val char(5));
insert into tst1(id, val) values (1,'111'),(2,'bbb'),(3,'aaa');

create or replace table tst (id int, ch char(5));
insert into tst(id, ch) values (1,'aaa'),(2,'bbb'),(3,NULL),
 (4,'bbb');

select count(*) from tst
where ch=nvl((select ch from tst2 where id=2),
(select val from tst1  where id=2));

|          2|

select count(*) from tst
 where ch=nvl((select ch from tst2 where id=3),
              (select val from tst1  where id=3));

|          1|
```

Примеры

```
create table tab1 (i int);
…
select * from tab1;
```

```
| 1    |
| NULL |
| 3    |
| 4    |
| NULL |
```

```
select avg (i), avg(nvl(i,0)) from tab1;

|2.66666666 |1.6 |
```

```
select nvl(cast i as char, 'Неопределено') from tab1;

| 1             |
| Неопределено  |
| 3             |
| 4             |
| Неопределено  |
```

Двухвариантная замена NULL-значения реальным значением (NVL2)

Функция

Двухвариантная замена неопределенного значения (NULL-значения) реальным значением.

Спецификация

[1] <синтаксис>::= NVL2(<выражение1>, <выражение2>, < выражение3>)
[2] <выражение1>::= <значимое выражение>
[3] <выражение2>::= <значимое выражение>
[4] <выражение3>::= <значимое выражение>

Общие правила

1) Функция возвращает значение <выражения2> в случае, если <выражение1> не является NULL-значением, иначе возвращает значение <выражения3>.
2) Функция эквивалентна конструкции СУБД ЛИНТЕР:

```
CASE WHEN <выражение1> IS NOT NULL
THEN <выражение2> ELSE <выражение3> END;
```

3) Типы данных <выражения1>, <выражения2>, <выражения3> должны быть идентичны или приводимыми. При приведении типов данных по умолчанию (без конструкции CAST) второй и третий аргументы приводятся к типу данных первого аргумента.
4) Аргументы могут быть заданы <SQL-параметром>, который должен содержать спецификацию типа данных параметра.

```
select 'Скидка '+ nvl2(? (char(5)), '10%', 'не предоставляется');
500
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
|Скидка 10%                |

select 'Скидка '+ nvl2(? (char(5)), '10%', 'Не предоставляется');
NULL

|Скидка не предоставляется|
```

Пример

```
create or replace table tst (i int);
insert into tst values (1), (null), (4), (10);

Эти запросы эквивалентны:

select case when i is not null
then cast i*100 as char else 'Неопределено' end from tst;

select nvl2(i,cast i*100 as char, 'Неопределено') from tst;

|100         |
|Неопределено|
|400         |
|1000        |
```

Проверка на равенство двух значений (NULLIF)

Функция

Проверка на равенство двух значений.

Спецификация

[1] <синтаксис>::= NULLIF(<выражение1>, <выражение2>)

Общие правила

1) Функция создана для совместимости с СУБД ORACLE.
2) Функция возвращает NULL-значение, если значения её аргументов совпадают, и <выражение1> – в противном случае. Эквивалентна конструкции СУБД ЛИНТЕР:

```
CASE WHEN <выражение1> = <выражение2>
THEN NULL ELSE <выражение1> END
```

3) Типы данных <выражения1>, <выражения2> должны быть идентичны или приводимыми. При приведении типов данных по умолчанию (без конструкции CAST) второй аргумент приводится к типу данных первого аргумента.
4) Аргументы могут быть заданы <SQL-параметром>, который должен содержать спецификацию типа данных параметра.

```
select nullif(? (int)+ ? (bigint), ? (double));
45
57650000
-0.756
```

```
|            57650045|

select nullif(? (real), ? (real));
0.00000000000000000000000000000000000000000000001
0.00000000000000000000000000000000000000000000099

|             |
```

Пример

```
select nullif(5,5), nullif(5, 4.99999);

|NULL| 5|
```

Поиск первого реального значения (COALESCE)

Функция

Поиск в заданном списке значений первого реального (не NULL) значения.

Спецификация

[1] <синтаксис>::=
COALESCE(<выражение1>, <выражение2> ... <выражениеn>)
[2] <выражениеn>::= <значимое выражение>

Общие правила

1) Функция создана для совместимости с СУБД ORACLE.
2) Функция возвращает первое <выражениеi> из заданного списка, которое не является NULL-значением. Если список выражений содержит только NULL-значения, возвращается NULL-значение. Например:

Var1	Var2	Var3	COALESCE (var1, var2, var3)
10	20	30	10
10	NULL	30	10
10	20	NULL	10
NULL	20	30	20
NULL	NULL	30	30
NULL	NULL	NULL	NULL

3) Тип данных используемых выражений должен совпадать (при этом UNICODE и не UNICODE типы данных смешивать нельзя).
4) Аргументы могут быть заданы <SQL-параметром>, который должен содержать спецификацию типа данных параметра.

```
select COALESCE(? (int), ? (int), ? (int));
56
-78
NULL
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
|         56|
```

Пример

```
Проверить, содержат ли записи таблицы реальные значения.
create or replace table tst (i1 int, i2 int, i3 int);
insert into tst (i1,i2,i3) values(null,null,null);
insert into tst (i1,i2,i3) values(null,null,3);
insert into tst (i1,i2,i3) values(null,2,3);
insert into tst (i1,i2,i3) values(1,2,null);
insert into tst (i1,i2,i3) values(1,null,3);
insert into tst (i1,i2,i3) values(1,2,3);
select i1, i2, i3,
  decode(COALESCE(i1,i2,i3), null, 'нет', 'да') from tst;

|   1|   2|   3|да |
|   1|NULL|   3|да |
|   1|   2|NULL|да |
|NULL|   2|   3|да |
|NULL|NULL|   3|да |
|NULL|NULL|NULL|нет|
```

Замена NaN-значения правильным значением (NANVL)

Функция

Замена неверного вещественного значения (NaN-значения) правильным.

Спецификация

[1] <синтаксис>::= NANVL(<выражение1>, <выражение2>)

Общие правила

1) <Выражение1>, <выражение2> должны быть вещественными значениями (типа REAL, DOUBLE) или приводиться к нему.

Общие правила

1) Если <выражение1> имеет значение NaN или +/-INFINITY, функция возвращает значение <выражения2>, в противном случае – значение <выражения1>.
2) Данная функция эквивалентна конструкции СУБД ЛИНТЕР:

```
CASE WHEN <выражение1> IS NOT NAN
THEN <выражение 1> ELSE <выражение2>;
```

3) Аргументы могут быть заданы <SQL-параметром>, который должен содержать спецификацию типа данных параметра.

```
select nanvl(cast HEXTORAW(? (char(30))) as double, ? (real));
000000000000F8FF0000000000000000
0.097
```

```
|    0.0970000028610229|
```

Примеры

```
create or replace table test( r1 real, r2 real );
insert into test values(1.1,1.1);
insert into test;
insert into test
 select cast( HEXTORAW('0000C0FF00000000') ) as real, 2.2;
insert into test
 select 2.2,  cast( HEXTORAW('0000807F00000000') ) as real;
insert into test
 select cast( HEXTORAW('0000C0FF00000000') ) as real,
        cast( HEXTORAW('0000C0FF00000000') ) as real;
insert into test
 select cast( HEXTORAW('000080FF00000000') ) as real,
        cast( HEXTORAW('000080FF00000000') ) as real;
insert into test values(3.3,3.3);
insert into test(r2)
 select cast( HEXTORAW('0000807F00000000') ) as real;
insert into test(r1)
 select cast( HEXTORAW('0000807F00000000') ) as real;
insert into test(r2) values (5.5);
insert into test(r1) values (5.5);
select * from test order by r1;
```

```
|R1   |R2   |
|1.1  |1.1  |
|2.2  |inf  |
|3.3  |3.3  |
|5.5  |     |
|-nan |2.2  |
|-nan |-nan |
|-inf |-inf |
|inf  |     |
|     |     |
|     |inf  |
|     |5.5  |
```

```
select sum(r1), sum(nanvl(r1, 0)), count(r2)
 from test where r2 is not nan and r2 is not null;
```

```
| -nan | 4.39999997615814 | 4 |
```

Генерация уникального идентификатора (SYS_GUID)

Функция

Генерация глобального уникального идентификатора для БД.

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

Спецификация
[1] <синтаксис>::= SYS_GUID()

Общие правила

1) Функция генерирует уникальный 16-ти байтовый глобальный (т.е. не повторяющийся в БД) идентификатор, состоящий из номера процесса клиента в операционной системе, времени открытия канала клиента для работы с БД и текущего системного времени.
2) Уникальный идентификатор генерируется при каждом вызове функции.

   Например, при выполнении запроса

```
SELECT SYS_GUID() FROM AUTO;
```

   будет выдано столько уникальных идентификаторов, сколько записей в таблице AUTO.

Возвращаемое значение

1) Результат имеет тип BYTE(16).
2) При занесении в CHAR/VARCHAR конвертируется по правилам функции RAWTOHEX.

Пример

```
SELECT SYS_GUID() FROM AUTO limit 1;

| DF 37 E8 3D 98 FA 64 2F C7 6B E0 F4 AE 26 CD 6B|
```

Информация о версии СУБД (LINTER_VERSION)

Функция

Предоставление информации о работающей версии СУБД ЛИНТЕР.

Спецификация
[1] <синтаксис>::= LINTER_VERSION([<формат>])
[2] <формат>::= <числовой литерал>

Общие правила

1) <Формат> – неотрицательное целочисленное значение типа INTEGER, задающее номер формата представления информации о версии СУБД ЛИНТЕР. Если параметр не задан, по умолчанию используется 0.

   Эти вызовы функции эквивалентны:

```
LINTER_VERSION(), LINTER_VERSION(0)
```

2) Поддерживаемые форматы:
   - 0: выводится наименование СУБД, полный номер версии и операционная система;

```
select cast linter_version(0) as char (60);

|Linter SQL Bastion v. 6.0.20.5 for LINUX (Socket) 64-bit   |
```

- 1: выводится информация о copyright;

```
select cast linter_version(1) as char (60);

|Copyright (C) 1990-2025 Relex, Inc. All rights reserved.    |
```

- 2: выводится информация о номере версии;

```
select cast linter_version(2) as char (60);

|6.0.20.5                                                    |
```

- 3: выводится наименование СУБД;

```
select cast linter_version(3) as char (60);

|Linter SQL Bastion                                          |
```

- 4: выводится наименование СУБД и полный номер версии.

```
select cast linter_version(4) as char (60);

|Linter SQL Bastion v. 6.0.20.5                              |
```

- Аргумент может быть задан <SQL-параметром>, который должен содержать спецификацию типа данных параметра.

```
select linter_version(? (int));
0

|Linter SQL Bastion v. 6.0.20.5 for LINUX (Socket) 64-bit|
```

Возвращаемое значение

Результат имеет тип VARCHAR(256).

Информация о размере файла таблицы (LINTER_FILE_SIZE)

Функция

Предоставление информации о размере файла таблицы.

Спецификация

[1] <синтаксис>::= LINTER_FILE_SIZE(<идентификатор таблицы>, <тип файла>, <номер файла>)
[2] <идентификатор таблицы>::= <числовой литерал>
[3] <тип файла>::= <числовой литерал>
[4] <номер файла>::= <числовой литерал>

Общие правила

1) Параметр <идентификатор таблицы> задает идентификатор той таблицы, характеристики файлов которой требуется получить. Идентификаторы таблиц хранятся в системной таблице $$$SYSRL, столбец $$$S11.

2) Параметр <тип файла> задает тип файла таблицы:
   - 0 – индексный файл;
   - 1 – файл данных;
   - 2 – файл BLOB-данных.
3) Параметр <номер файла> определяет порядковый номер файла заданного типа (начиная с 1).
4) Аргумент может быть задан <SQL-параметром>, который должен содержать спецификацию типа данных параметра.

```
select linter_file_size(? (int), ? (int), ? (int));
532
1
1

|           7|
```

Возвращаемое значение

1) Размер указанного файла в страницах (по 4 Кбайта).
2) Код завершения СУБД – если файл с заданными атрибутами не найден.
3) Тип возвращаемого значения – INTEGER.
4) Если хотя бы один аргумент равен NULL-значению, результат NULL-значение.

Пример

```
select linter_file_size(8,1,1) as "Размер файла";
Размер файла

|87 |
```

Информация о местонахождении файлов таблицы (LINTER_FILE_DEVICE)

Функция

Получить информацию о местонахождении файлов таблицы.

Спецификация

[1] <синтаксис>::=
LINTER_FILE_DEVICE(<идентификатор таблицы>, <тип файла>, <номер файла>)

Общие правила

1) Параметр <идентификатор таблицы> задает идентификатор той таблицы, информацию о местонахождении файлов которой требуется получить. Идентификаторы таблиц хранятся в системной таблице $$$SYSRL, столбец $$$S11.
2) Параметр <тип файла> задает тип файла таблицы:
   - 0 – индексный файл;

- 1 – файл данных;
- 2 – файл BLOB-данных.

3) Параметр <номер файла> определяет порядковый номер файла заданного типа (начиная с 1).

4) Аргументы могут быть заданы <SQL-параметром>, который должен содержать спецификацию типа данных параметра.

```
select linter_file_device(? (int), ? (int), ? (int));
176
1
1

|SY00|
```

Возвращаемое значение

1) Имя переменной среды окружения (например, SY01), задающей местонахождение указанного файла таблицы.

2) Код завершения СУБД – если переменная среды окружения не определена.

3) Тип возвращаемого значения – CHAR(4).

4) Если хотя бы один аргумент равен NULL-значению, результат NULL-значение.

Пример

```
select linter_file_device(8,1,1) as "Местоположение";
Местоположение

|SY00 |
```

Информация о состоянии файлов таблицы (LINTER_FILE_INFO)

Функция

Предоставление информации о текущем состоянии файлов таблицы.

Спецификация

[1] <синтаксис>::= LINTER_FILE_INFO(<идентификатор таблицы>, <тип файла>, <номер файла>, <тип информации>)

[2] <тип информации>::= <символьный литерал>

Общие правила

1) Параметр <идентификатор таблицы> задает идентификатор той таблицы, информацию о состоянии файлов которой требуется получить. Идентификаторы таблиц хранятся в системной таблице $$$SYSRL, столбец $$$S11.

2) Параметр <тип файла> задает тип файла таблицы:

- 0 – индексный файл;

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

- 1 – файл данных;
- 2 – файл BLOB-данных.

3) Параметр <номер файла> определяет порядковый номер файла заданного типа (начиная с 1).

4) Параметр <тип информации> определяет тип запрашиваемой о файле информации. Тип данных параметра – CHAR, значения параметра являются регистронезависимыми.

5) Допустимые значения параметра приведены в пункте «Возвращаемое значение».

6) Аргументы могут быть заданы <SQL-параметром>, который должен содержать спецификацию типа данных параметра.

```
select linter_file_info(? (int), ? (int), ? (int), ? (char(15)));
532
1
1
bitmap_size

|          1|
```

Возвращаемое значение

1) Тип возвращаемого значения – INTEGER.

2) Возвращаемое значение зависит от параметра <тип информации>.

Значение параметра <тип информации>	Возвращаемое значение
size	Размер файла в страницах
bitmap_size	Размер битовой карты файла в страницах
conv_size	Размер файла-конвертера в страницах
free_page_count	Число страниц файла (кроме битовой карты и конвертера), помеченных как свободные
full_page_count	Число страниц файла (кроме битовой карты и конвертера), помеченных как занятые

3) Если хотя бы один аргумент равен NULL-значению, результат NULL-значение.

Пример

```
select cast ($$$s34 as char(18)) as "UserName",
 cast ($$$s13 as char(18)) as "TableName",
 linter_file_info($$$s11,0,1,'size') as "IndexFileSize",
 linter_file_info($$$s11,0,1,'bitmap_size') as "IndexBitmapSize",
 linter_file_info($$$s11,0,1,'conv_size') as "IndexConvSize",
 linter_file_info($$$s11,0,1,'full_page_count')
  as "IndexFullCount",
 linter_file_info($$$s11,0,1,'free_page_count')
  as "IndexFreeCount",
 linter_file_info($$$s11,1,1,'size') as "DataFileSize",
```

```
linter_file_info($$$s11,1,1,'bitmap_size')
 as "DataBitmapSize",
linter_file_info($$$s11,1,1,'full_page_count')
 as "DataFullCount",
linter_file_info($$$s11,1,1,'free_page_count')
 as "DataFreeCount",
case when getbyte($$$s14,102) > 0
 then linter_file_info($$$s11,2,1,'size')
  else NULL end as "BlobFileSize",
case when getbyte($$$s14,102) > 0
 then linter_file_info($$$s11,2,1,'bitmap_size')
  else NULL end as "BlobBitmapSize",
case when getbyte($$$s14,102) > 0
 then linter_file_info($$$s11,2,1,'full_page_count')
  else NULL end as "BlobFullCount",
case when getbyte($$$s14,102) > 0
 then linter_file_info($$$s11,2,1,'free_page_count')
  else NULL end as "BlobFreeCount"
from LINTER_SYSTEM_USER.$$$sysrl,
     LINTER_SYSTEM_USER.$$$usr
where $$$s31 = $$$s12 and
      $$$s11 > 0 and
      $$$s32 = 0 and
      getbyte($$$s14,6) = 0;
```

Данный запрос выдает информацию о первых файлах таблицы каждого типа, для получения информации о прочих файлах таблицы запрос необходимо корректировать (усложнить).

Информация о состоянии страницы файла (LINTER_PAGE_INFO)

Функция

Предоставление информации о текущем состоянии заданной страницы файла таблицы.

Спецификация

[1] <синтаксис>::= LINTER_PAGE_INFO(<идентификатор таблицы>, <тип файла>, <номер файла>, <номер страницы>, <тип информации>)

[2] <номер страницы>::= <числовой литерал>

Общие правила

1) Параметр <идентификатор таблицы> задает системный идентификатор таблицы, о состоянии страниц которой запрашивается информация. Системные идентификаторы таблиц хранятся в системной таблице $$$SYSRL, столбец $$$S11.

2) Параметр <тип файла> задает тип файла таблицы:

   - 0 – индексный файл;

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

- 1 – файл данных;
- 2 – файл BLOB-данных.

3) Параметр <номер файла> определяет порядковый номер файла заданного типа (начиная с 1).

4) Параметр <номер страницы> определяет порядковый номер страницы в заданном файле (начиная с 1).

5) Параметр <тип информации> определяет тип запрашиваемой о файле информации. Тип данных параметра – CHAR, значения параметра являются регистронезависимыми.

6) Допустимые значения параметра приведены в пункте «Возвращаемое значение».

7) Аргументы могут быть заданы <SQL-параметром>, который должен содержать спецификацию типа данных параметра.

```
select linter_page_info(? (int), ? (int), ? (int), ? (int),  ?
 (char(15)));
173
1
1
1
used_size

|       4096|
```

Возвращаемое значение

1) Тип возвращаемого значения – INTEGER.

2) Возвращаемое значение зависит от параметра <тип информации>.

3) Если хотя бы один аргумент равен NULL-значению, результат NULL-значение.

Значение параметра <тип информации>	Возвращаемое значение
used_size	Размер занятой части страницы в байтах (для страниц битовой карты и конвертера – размер всей страницы)
free_size	Размер свободной части страницы в байтах
status	Битовая маска состояния страницы (первый бит): 1 – страница занята, 0 – свободна

Примеры

1)

```
select linter_page_info ($$$s11,1,1,2,'free_size'),
       linter_page_info ($$$s11,1,1,2,'used_size')
  from $$$sysrl
 where $$$s13='AUTO';

| 80|4016|
```

2) Проверить, является ли заданная страница свободной:

```
select decode(linter_page_info ($$$s11,1,1,2,'status'), 1,'Нет',0,
 'Да')
from $$$sysrl where $$$s13='AUTO';

|Нет|
```

Информация о состоянии страницы файла (LINTER_PAGE_TIME)

Функция

Предоставление информации о дате и времени последнего обновления страницы файла таблицы.

Спецификация

[1] <синтаксис>::=
LINTER_PAGE_TIME(<идентификатор таблицы>, <тип файла>,
<номер файла>, <номер страницы>, <тип информации>, <код типа страницы>)

Общие правила

1) Параметр <идентификатор таблицы> задает системный идентификатор таблицы, о состоянии страниц которой запрашивается информация. Системные идентификаторы таблиц хранятся в системной таблице $$$SYSRL, столбец $$$S11.
2) Параметр <тип файла> задает тип файла таблицы:
   - 0 – индексный файл;
   - 1 – файл данных;
   - 2 – файл BLOB-данных.
3) Параметр <номер файла> определяет порядковый номер файла заданного типа (начиная с 1).
4) Параметр <номер страницы> определяет порядковый номер страницы в заданном файле (начиная с 1).
5) Параметр <тип информации> определяет тип запрашиваемой о файле информации. Тип данных параметра – CHAR, значения параметра являются регистронезависимыми.
6) Допустимые значения параметра приведены в пункте «Возвращаемое значение», пункта «Информация о состоянии страницы файла».
7) Параметр <код типа страницы> (1 - конвертер, 2 - битовая карта, 3 - данные, 4 - индекс, 5 - BLOB, 6 - сортировка, 7 - бит-вектора, 8 - VIEW, 9 - ограничения целостности, 10 - рабочий файл, 11 - описание файлов, 12 - CHECK). Если пятый параметр - NULL, то ядро пытается самостоятельно определить тип страницы в зависимости от типа файла.

Возвращаемое значение

1) Тип возвращаемого значения – DATE.
2) Возвращается дата и время последнего обновления страницы в формате DATE (по Гринвичу) или NULL, если нельзя выдать дату и время обновления страницы (например, для битовой карты - там его нет).

Информация о фразовом индексе (LIN_INDEX_INFO)

Функция

Предоставление информации о ресурсах, используемых при работе с фразовым индексом.

Спецификация

[1] <синтаксис>::=
LIN_INDEX_INFO(<спецификация таблицы>, <спецификация столбца>,
<тип информации>)
[2] <спецификация таблицы>::=
<имя таблицы>
|<символьный литерал>
|идентификатор таблицы
[3] <спецификация столбца>::=
<имя столбца>
|<символьный литерал>
|идентификатор столбца

Общие правила

1) <Идентификатор таблицы> – значение поля $$$S11 в системной таблице $$$SYSRL.
2) <Идентификатор столбца> – значение поля $$$S22 в системной таблице $$$ATTRI.
3) <Спецификация столбца> должна ссылаться на столбец, по которому создан фразовый индекс.
4) Параметр <тип информации> определяет тип запрашиваемой о фразовом индексе информации. Допустимые значения параметра приведены в пункте «Возвращаемое значение».
5) Функция предоставляет информацию о фактических размерах ресурсов, используемых при выполнении поисковых операций с фразовым индексом. Анализ возвращаемой функцией информации по всем столбцам с фразовым индексом всех таблиц БД позволяет оценить правильность задания значения ключей запуска pcontcache, pbvcache ядра СУБД. Например, рекомендуется задавать значение ключа pcontcache не меньше суммы всех вычисленных в БД значений file_size (для поиска может быть достаточно только w2d_packed), а ключа pbvcache – не меньше суммы всех вычисленных в БД значений s2w_unpacked.
6) Значения <символьный литерал> и <идентификатор столбца> аргументов могут быть заданы <SQL-параметром>, который должен содержать спецификацию типа данных параметра.

```
select lin_index_info(? (char(10)), ? (char(20)), 'file_size')
  from $$$SYSRL, $$$ATTRI
 where $$$S11 = $$$S21
   and $$$S13 = ? (char(10)) and $$$S23 = ?(char(20));
TST
COL_INDEX
TST
COL_INDEX
```

Возвращаемое значение

1) Тип возвращаемого значения – INTEGER (при отсутствии фразового индекса для столбца возвращается NULL-значение).
2) Возвращаемое значение зависит от параметра <тип информации>.

Значение параметра <тип информации>	Возвращаемое значение
s2w_unpacked	Размер (в страницах 4 Кбайта) распакованной структуры индекса буквосочетаний (бит-векторный кэш)
w2d_unpacked	Размер (в страницах 4 Кбайта) распакованной структуры индекса соответствия слов документу (не кэшируется в распакованном виде)
s2w_packed	Размер (в страницах 4 Кбайта) упакованной структуры индекса буквосочетаний (контейнерный кэш)
w2d_packed	Размер (в страницах 4 Кбайта) упакованной структуры индекса соответствия слов документу (контейнерный кэш)
file_size	Размер (в страницах 4 Кбайта) файла контейнера фразового индекса
file_id	Идентификатор индексного файла фразового индекса (char(8))
dict_size	Суммарный объём словаря фразового индекса в байтах
word_count	Количество слов в словаре фразового индекса
hast_size	Суммарный объём вспомогательной части фразового индекса (для быстрого удаления документов и для поиска с расстояниями между словами)

Примеры

```
Выполняется с помощью утилиты inl:
create or replace table tst(col_index blob default filter
 DOCRTF2TEXT);
insert into tst values(NULL);
blob insert column=1  file=sql.pdf
CREATE OR REPLACE PHRASE deferred INDEX col_index ON tst;
REBUILD PHRASE INDEX col_index ON tst;
1)
Два нижеследующих запроса идентичны:
select lin_index_info('TST', 'COL_INDEX', 's2w_unpacked'),
       lin_index_info('TST', 'COL_INDEX', 'w2d_unpacked'),
       lin_index_info('TST', 'COL_INDEX', 's2w_packed'),
       lin_index_info('TST', 'COL_INDEX', 'w2d_packed'),
       lin_index_info('TST', 'COL_INDEX', 'file_size'),
       lin_index_info('TST', 'COL_INDEX', 'file_id')
  from $$$SYSRL, $$$ATTRI
 where $$$S11 = $$$S21 and $$$S13 = 'TST' and $$$S23 =
 'COL_INDEX';
```

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
|       1543|       1256|         27|         10|         48|
 66310|

select lin_index_info($$$S11, $$$S22, 's2w_unpacked'),
       lin_index_info($$$S11, $$$S22, 'w2d_unpacked'),
       lin_index_info($$$S11, $$$S22, 's2w_packed'),
       lin_index_info($$$S11, $$$S22, 'w2d_packed'),
       lin_index_info($$$S11, $$$S22, 'file_size'),
       lin_index_info($$$S11, $$$S22, 'file_id')
  from $$$SYSRL, $$$ATTRI
 where $$$S11 = $$$S21
   and $$$S13 = 'TST'
   and $$$S23 = 'COL_INDEX';

|       1543|       1256|         27|         10|         48|
 66310|

2)
select sum(lin_index_info($$$S13, $$$S23, 'file_size'))
 pcontcache,
       sum(lin_index_info($$$S13, $$$S23, 's2w_unpacked'))
 pbvcache
  from $$$SYSRL, $$$ATTRI
 where $$$S11 = $$$S21;

3)
select * from
(
 select
  $$$S13 tab,
  $$$S23 col,
  lin_index_info($$$S13, $$$S23, 'file_id')      fileid,
  lin_index_info($$$S13, $$$S23, 'file_size')    pcontcache,
  lin_index_info($$$S13, $$$S23, 's2w_unpacked') pbvcache
 from $$$SYSRL, $$$ATTRI
 where $$$S11 = $$$S21
) where pcontcache is not null;
```

Информация о длине внутреннего представления выражения (VSIZE)

Функция

Предоставление информации о длине (в байтах) внутреннего представления выражения.

Спецификация

[1] <синтаксис>::= VSIZE(<значимое выражение>)

Общие правила

1) <Значимое выражение> может быть любым допустимым в SQL СУБД ЛИНТЕР типом данных (в том числе, EXTFILE, BLOB).
2) Для типов данных переменной длины (VARCHAR, VARBYTE, NCHAR VARYING) размер внутреннего представления возвращается с учетом двух байт, в которых хранится значение длины этих типов данных.
3) Аргумент может быть задан <SQL-параметром>, который должен содержать спецификацию типа данных параметра.

```
select vsize(? (char (30)));
12345

|         30|
```

Возвращаемое значение

1) Значение типа BIGINT, содержащее длину внутреннего представления аргумента функции в байтах.

Примеры

```
select vsize(1), vsize(cast 1 as bigint);

|    4|                8|
```

```
select make, length(make), vsize(make) from auto limit 1;

|FORD    |    4|          20|
```

```
create or replace table tst (vc varchar(10));
insert into tst (vc) values('a');
insert into tst (vc) values('aaa');
insert into tst (vc) values('aaaaaaaaaa');
select vc, length(vc), vsize(vc) from tst;

| a          | 1  | 3  |
| aaa        | 3  | 5  |
| aaaaaaaaaa | 10 | 12 |
```

Информация об ограничениях целостности (LINTER_DICT_INFO)

Функция

Предоставление выражения для генерируемого столбца или ограничения целостности CHECK для других типов столбцов.

Спецификация

[1] <синтаксис>::= LINTER_DICT_INFO(<тип информации>, <идентификатор таблицы>[, <номер столбца>])
[2] <тип информации>::= <числовой литерал>

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

[3] <идентификатор таблицы>::= <числовое выражение> |<числовой литерал>
[4] <номер столбца>::= <числовой литерал>

Общие правила

1) Параметр <идентификатор таблицы> задает системный идентификатор таблицы, об ограничениях целостности которой запрашивается информация. Системные идентификаторы таблиц хранятся в системной таблице $$$SYSRL, столбец $$$S11.
2) Параметр <тип информации> задает тип запрашиваемой информации:
   - 1 – ограничение целостности CHECK;
   - 2 – выражение для генерируемого столбца.
3) Параметр <номер столбца> определяет порядковый номер столбца заданной таблицы (начиная с 1).
4) Если <номер столбца> не задан или равен 0, возвращаются ограничения целостности для всей таблицы.
5) Аргументы могут быть заданы <SQL-параметром>, который должен содержать спецификацию типа данных параметра.

```
select trim(linter_dict_info (? (int), ? (int), ? (int)));
1
537
0

|CHECK("I" > "J") |
```

Возвращаемое значение

1) Символьная строка переменной длины VARCHAR(2048). Формат строки зависит от значения входных параметров:
   - <номер режима>=1, <номер столбца> больше 0:

```
"CHECK(<выражение1>)<пробел> CHECK(<выражение2>)..."
```

   - <номер режима>=1, <номер столбца> равен 0:

```
"CHECK(<выражение1>),CHECK(<выражение2>) ..."
```

   - <номер режима>=2, <номер столбца> больше 0:

```
"<выражение для генерируемого столбца>"
```

2) Пустая строка – если для столбца не задано условие CHECK, или он не является генерируемым столбцом, или он создан с атрибутом GENERATED AS IDENTITY.

Примеры

1) Получить CHECK-выражение для всей таблицы

```
CREATE or replace TABLE TST_CHECK (I INT, J INT, CHECK (I>J));

select trim(linter_dict_info (1,$$$s11, 0)) from $$$sysrl
 where $$$s13='TST_CHECK';
```

```
|CHECK("I" > "J") |
```

2) Получить CHECK-выражение для заданного столбца

```
CREATE or replace TABLE TST_CHECK (I INT CHECK (I>0) CHECK
 (I<10));
select trim(linter_dict_info (1,$$$s11, 1)) from $$$sysrl
 where $$$s13='TST_CHECK';

|CHECK("I" > 0) CHECK("I" < 10) |
```

Информация о состоянии событий (GET_EVENTS_STATE)

Функция

Предоставление информации о текущем состоянии заданных событий.

Спецификация

[1] <синтаксис>::=
GET_EVENTS_STATE (<имя события1>[, <имя события2> …])
[2] <имя события1>::= <символьный литерал>
[3] <имя события2>::= <символьный литерал>

Общие правила

1) Максимальное число имён событий, которое можно передать функции, равно 32 (по числу бит в возвращаемом значении).
2) <Символьный литерал> может быть UNICODE-литералом.
3) Аргумент может быть задан <SQL-параметром>, который должен содержать спецификацию типа данных параметра.

Возвращаемое значение

Значение целого типа INTEGER, представляющее собой битовую маску состояний указанных событий. Например, для двух событий:

- если наступило событие1, но не наступило событие2, будет возвращено значение 1;
- если наступило событие1 и наступило событие2, будет возвращено значение (1 | 2) = 3.

Пример

Запуск в inl файла event1.cmd приводит к созданию таблицы и двух событий на добавление записей в таблицы $$$ATTRI и $$$SYSRL.

После запуска файла event2.cmd должны получить следующие результаты:

1) сначала оба события не наступили, и оба вызова функции GET_EVENTS_STATE возвращают 0;
2) затем наступило событие ATTRI (добавили столбец), вызов GET_EVENTS_STATE('ATTRI','SYSRL') возвращает 1, а вызов GET_EVENTS_STATE('SYSRL','ATTRI') возвращает 2;
3) когда оба события наступили, оба вызова GET_EVENTS_STATE возвращают 3.

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

```
event1.cmd
inl.exe -t -q -u SYSTEM/MANAGER8 -f eve1.sql
start inl.exe -u SYSTEM/MANAGER8 -f eve2.sql
start inl.exe -u SYSTEM/MANAGER8 -f eve3.sql

event2.cmd
inl.exe -t -q -u SYSTEM/MANAGER8 -f eve0.sql
inl.exe -t -q -u SYSTEM/MANAGER8 -f eve4.sql
echo result is 1:
inl.exe -t -q -u SYSTEM/MANAGER8 -f eve0.sql
inl.exe -t -q -u SYSTEM/MANAGER8 -f eve1.sql
echo result is 3:
inl.exe -t -q -u SYSTEM/MANAGER8 -f eve0.sql

eve0.sql
select GET_EVENTS_STATE('ATTRI','SYSRL');
select GET_EVENTS_STATE('SYSRL','ATTRI');

eve1.sql
create or replace table test(i int);

eve2.sql
drop event ATTRI;
create event ATTRI as insert on $$$ATTRI;
wait event ATTRI;

eve3.sql
drop event SYSRL;
create event SYSRL as insert on $$$SYSRL;
wait event SYSRL;

eve4.sql
alter table test add column j int;
```

Фонетический код значения (SOUNDEX)

Функция

Получение фонетического кода значения.

Спецификация

[1] <синтаксис>::= SOUNDEX(<значение>)
[2] <значение>::= <символьное выражение>

Общие правила

1) Параметр <значение> должен быть символьным типом данных или приводиться к нему.

2) Аргумент может быть задан <SQL-параметром>, который должен содержать спецификацию типа данных параметра.

```
select soundex(? (char (10))), soundex( ? (char (10))),
       soundex(? (char (10))), soundex(? (char (10))),
       soundex(? (char (10)));
РЕЛЭКС
РЕЛЕКС
РЕЛКС
РЭЛКС
RELEX

|P736|P736|P736|P736|R420|
```

Возвращаемое значение

1) Символьная строка, представляющая фонетический код заданного <значения>.
2) Тип возвращаемого значения – `CHAR(4)`.

> **Примечание**
> Возвращаемое значение в дальнейшем может использоваться для подбора подходящей фонетической фразы.

Примеры

1) Фонетические коды M200 указывают на близкое фонетическое звучание слов.

```
select soundex('make'), soundex('mak'),
       soundex('makk'), soundex('maket');

|M200||M200||M200||M230|
```

2)

```
select soundex('cupe');

|C100 |

select bodytype from auto where soundex(bodytype) ='C100';

|COUPE |
|COUPE |
```

…

Определение близости фонетического звучания (DIFFERENCE)

Функция

Определение близости фонетического звучания.

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

Спецификация

[1] <синтаксис>::= DIFFERENCE(<значение1>, <значение2>)
[2] <значение1>::= <символьное выражение>
[3] <значение2>::= <символьное выражение>

Общие правила

1) Параметры `<значение1>` и `<значение2>` должны быть символьным типом данных или приводиться к нему.
2) Аргументы могут быть заданы `<SQL-параметром>`, который должен содержать спецификацию типа данных параметра.

```
select difference(? (char (10)), ? (char (10)));
РЕЛЭКС
РЕЛЕКС

|          0|
```

Возвращаемое значение

1) Разность фонетического звучания двух аргументов, вычисляемая на основании кодов фонетического значения аргументов (см. функцию SOUNDEX).
2) Тип возвращаемого значения – `INTEGER`:
   - `0` – фонетическое совпадение аргументов;
   - `1` – существенное различие.

Пример

```
select difference('make', 'makk'),
      difference('make', 'makt'),
      difference('make', 'abcd');

|0 |1 |1 |
```

Приложение 1
Ключевые слова СУБД ЛИНТЕР

acos
add
add_months
after
all
alter
always
and
any
append
as
asbinary
asc
ascic
ascii
asin
astext
asymmetric
at_begin
at_end
atan
atan2
aud_obj_name_len
audit
autocommit
autoinc
autoload
autoreset
autorowid
autosave
avg
backup
before
between
bigint
blob
blobfile
blobfiles
blobpct
blobs
boolean
both
box
browse
by
byte
bytehex
call
cancel
cascade
cascaded
case
cast
ceil
char
char_length
character
character_length
check
chr
cidrbcast
cidrcount
cidrlen
cidrmask
cidrmatch
cidrnet
cidrpfx
cidrtoraw
circle
clear
clob
close
closed
coalesce
column
commit
committed
concat
connect
contains
convert
corresponding
cos
cosh
count
countblob
create
cross
crosses
current
current_date
current_schema
current_time
current_timestamp
cursor
datafile
datafiles
date
datesplit
datetime
dayname
dba
dbname
dec
decimal
decode
default
deferred
deftext
delete
denied
dense_rank
desc
disable

© Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

disconnect
disjoint
distinct
divtime
double
drop
each
ef
else
enable
end
equals
error
escape
event_info_size
every
except
exclude
exclusive
execute
exists
extfile
extract
exp
external
false
fetch
filter
findblob
findrtf
finish
first_value
float
floor
for
foreign
from
from_days
full
fuzzy
geodata
geoisvalid
geomcollfromtext
geomcollfromwkb
geometry
geometrycollection
geometryfromtext
geometryfromwkb
geometryn
geometrytype
geomfromtext
geomfromwkb
get
get_events_state
getbits
getblob
getblobstr
getbyte
getbyteb
getlong
getraw
getstr
gettext
gettextpos
getword
getwordb
glength
global
grant
granted
greatest
group
having
hex
hextoraw
identified
immediate
in
increment
index
indexes
indexfile
indexfiles
indextime
initcap
initial
inner
insert
instead
instr
int
integer
interiorringn
internal
intersect
intersection
intersects
into
ip_abbrev
ip_broadcast
ip_host
ip_masklen
ip_netmask
ip_network
ip_set_masklen
ip_text
is
isempty
isolation
join
krb
lag
last_autoinc
last_day
last_rowid
last_value
ldap
lead
leading
least
left
lenblob
like

limit
lin_area
lin_asbinary
lin_astext
lin_boundary
lin_buffer
lin_centroid
lin_contains
lin_convexhull
lin_crosses
lin_difference
lin_dimension
lin_disjoint
lin_distance
lin_dwithin
lin_endpoint
lin_envelope
lin_equals
lin_exteriorring
lin_filter
lin_geoisvalid
lin_geomcollfromtext
lin_geomcollfromwkb
lin_geometryfromtext
lin_geometryfromwkb
lin_geometryn
lin_geometrytype
lin_geomfromtext
lin_geomfromwkb
lin_glength
lin_index_info
lin_interiorringn
lin_intersection
lin_intersects
lin_isclosed
lin_isempty
lin_isring
lin_issimple
lin_linefromtext
lin_linestringfromtext
lin_linestringfromwkb
lin_linfromwkb
lin_mlinefromtext
lin_mlinefromwkb
lin_mpointfromtext
lin_mpointfromwkb
lin_mpolyfromtext
lin_mpolyfromwkb
lin_multilinestringfromtext
lin_multilinestringfromwkb
lin_multipointfromtext
lin_multipointfromwkb
lin_multipolygonfromtext
lin_multipolygonfromwkb
lin_numgeometries
lin_numinteriorrings
lin_numpoints
lin_overlaps
lin_pointfromtext
lin_pointfromwkb
lin_pointn
lin_pointonsurface
lin_polyfromtext
lin_polyfromwkb
lin_polygonfromtext
lin_polygonfromwkb
lin_relate
lin_srid
lin_startpoint
lin_symdifference
lin_touches
lin_transform
lin_union
lin_within
lin_x
lin_y
line
linefromtext
linefromwkb
linestring
linestringfromtext
linestringfromwkb
linter_dict_info
linter_file_device
linter_file_info
linter_file_size
linter_name_length
linter_page_info
linter_page_time
linter_user_id
linter_version
listtag
ln
local
localtime
localtimestamp
lock
log
logoff
logon
long
lpad
lseg
ltrim
mac_trunc
mactoraw
match
materialized
max
maxisn
maxrow
maxrowid
memory
min
mlinefromtext
mlinefromwkb
mod
modify
module
monthname
mpointfromtext
mpointfromwkb
mpolyfromtext

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

mpolyfromwkb
multilinestring
multilinestringfromtext
multilinestringfromwkb
multime
multipoint
multipointfromtext
multipointfromwkb
multipolygon
multipolygonfromtext
multipolygonfromwkb
nan
nanvl
national
natural
nchar
new
new_table
next_day
no
nomaxvalue
nominvalue
none
not
now
nowait
null
nullif
nulliferror
nulls
number
numeric
numgeometries
numinteriorring
numinteriorrings
nvarchar
nvl
nvl2
octet_length
of
old
old_table
on
only
open
optimistic
optimization
option
or
order
outer
over
overlaps
overlay
parameter
partial
partially
partition
pctfill
pctfree
placing
point
pointfromtext
pointfromwkb
pointn
pointonsurface
polyfromtext
polyfromwkb
polygon
polygonfromtext
polygonfromwkb
preload
press
primary
prior
privileges
proc_info_size
proc_par_name_len
procedure
public
purge
rand
rank
raw
rawtocidr
rawtohex
rawtomac
read
real
rebuild
references
relate
remote
rename
repeat_string
repeatable
replace
replication
restart
restore
restrict
reverse
revoke
rewrite
right
right_substr
rollback
row_number
rownum
rpad
rtrim
save
savepoint
security
select
sensitive
sequence
serializable
session
sessionid
set
share
shutdown

siblings
similar
sin
since
sinh
smallint
some
sortpool
soundex
sqrt
start
startup
stddev
stop
substr
substring
sum
symdifference
symmetric
synchronize
synonym
sys_guid
sysdate
table
tablesample
tan
tanh
then
ties
timeint_from_days
timeint_to_days
to
to_char
to_date
to_days
to_gmtime
to_localtime
to_number
touches
trailing
transaction
translate
trigger
trigger_info_size
trim
true
trunc
truncate
uncommitted
union
unique
unix_timestamp
unknown
unlisted
unlock
until
update
user
using
values
varbyte
varchar
variance
varying
view
volumes
vsize
wait
when
where
windows_code
with
within
without
write
xml

Приложение 2
Контекстно зависимые ключевые слова СУБД ЛИНТЕР

Ключевое слово	Недопустимое использование
ABS	функция
ACCESS	в командах GRANT ACCESS и REVOKE ACCESS
ACTION	в команде CREATE TABLE
ADDRESS	в команде CREATE STATION
ALIAS	в команде CREATE ALIAS
ANSWERCACHE	в команде SET ANSWERCACHE
AREA	функция
ASYNC	в команде BACKUP DATABASE
AUTHORIZATION	в команде CREATE SCHEMA
BASE	в команде SET PRIORITY
BERNOULLI	во FROM-спецификации
BITMAP	в команде TEST TABLE
BLOCK	в команде EXECUTE BLOCK
BOUNDARY	функция
BRIEF	в команде SET LOG
BUFFER	функция
CACHE	в конструкции подсказки о кэшировании запросов
CALCULATE	в команде CREATE REPLICATION RULE
CENTROID	функция
CHANNEL	в команде AUDIT
CHECKING	в команде SET DATABASE GEODATA VALIDITY
COMMENT	в команде CREATE DEVICE
CONNECTION	в команде SET TRACE
CONSTRAINTS	в команде SET CONSTRAINTS
CONVEXHULL	функция
CORRECT	в командах корректировки индекса
CUBE	в GROUP BY-спецификации
CURRENT_USER	в команде EXECUTE BLOCK
CYCLE	в команде CREATE TABLE
DATA	в команде ALTER TABLE
DATABASE	в командах CREATE TRIGGER, SET DATABASE NAMES
DAY	элемент «даты-времени»
DAYS	в команде AUDIT
DEFINER	в команде EXECUTE BLOCK
DESCRIPTION	в команде ALTER CHARACTER
DEVICE	после начальных ключевых слов CREATE|DROP|ALTER, в командах GRANT ACCESS и REVOKE ACCESS

Ключевое слово	Недопустимое использование
DIFFERENCE	в команде CREATE REPLICATION RULE
DIMENSION	функция
DIRECTORY	в командах CREATE DEVICE и ALTER DEVICE
DISTANCE	функция
DUMP	ключевое слово только после слова LINTER
DWITHIN	функция
ENDPOINT	функция
ENVELOPE	функция
EUC	в команде CREATE CHARACTER
EVENT	в команде CREATE EVENT
EXPIRATION_DATE	функция
EXTERIORRING	функция
FILE	в команде AUDIT
FILENAME	функция
FILES	в команде REBUILD TABLE
FILESIZE	функция
FILETIME	функция
FIRST	в конструкции ORDER BY-спецификации
GENERATED	в команде CREATE TABLE
GROUPING	аналитическая функция
HOUR	элемент «даты-времени»
HTML	функция
IDENTITY	в команде CREATE TABLE
IF	управляющий оператор
IGNORE	в команде REBUILD TABLE
INTEGRITY	в команде TEST TABLE
ISCLOSED	функция
ISRING	функция
ISSIMPLE	функция
KEY	в команде ALTER TABLE
LAST	в конструкции ORDER BY-спецификации
LENGTH	функция
LEVEL	псевдостолбец иерархического запроса и в конструкции управления уровнем доступа пользователей
LIFE	в команде ALTER USER
LIMIT	после ключевых слов STATION, DAYS, RECORDS
LINTER	в команде AUDIT
LOGIN	в команде ALTER USER
LOWER	функция
MATCHED	в команде MERGE

Ключевое слово	Недопустимое использование
MAXVALUE	в команде CREATE/ALTER SEQUENCE
MERGE	команда
MESSAGE	в команде AUDIT
MINVALUE	в команде CREATE/ALTER SEQUENCE
MINUTE	элемент «даты-времени»
MODE	после ключевых слов EXCLUSIVE и SHARE
MONTH	элемент «даты-времени»
NAME	в команде SET DATABASE
NAMES	в команде SET DATABASE и SET NAMES
NODE	в команде CREATE NODE
NONEUC	в команде CREATE CHARACTER
NUMBERS	команде CREATE REPLICATION RULE
NUMPOINTS	функция
OPERATION	в команде AUDIT
OWNER	в командах GRANT и REVOKE
PAGE	в команде CORRECT INDEX
PASSWORD	в команде BACKUP DATABASE
PATH	в команде SET DATABASE PATH
PERCENT	в конструкции FETCH FIRST
PHRASE	в команде REBUILD PHRASE INDEX
PLAN	в команде CREATE CHARACTER
POSITION	функция
POWER	функция
PRECISION	после идентификатора типа DOUBLE
PRESERVE	в команде CREATE TABLE
PRIORITY	после начального ключевого слова SET
PRIVATE	в команде CREATE EVENT
PROTOCOL	в команде CREATE STATION
QUANT	в команде TEST TABLE
QUANTUM	в команде TEST TABLE
RANGE	в команде SET PRIORITY
RECORD	в команде ALTER DATABASE
RECORDS	в команде AUDIT
RELEASE	в команде COMMIT и ROLLBACK
RESOURCE	в команде GRANT
REUSE	TRUNCATE TABLE
ROLE	в командах COMMENT ON, GRANT, REVOKE, CREATE ROLE, DROP ROLE
ROLLUP	в конструкции GROUP BY-спецификации
ROOT	в команде CREATE/ALTER TABLE

Ключевое слово	Недопустимое использование
ROUND	функция
ROW	после ключевого слова EACH
ROWS	в команде CREATE TABLE
RULE	после ключевого слова REPLICATION
SCAN	в команде SET { SESSION | DATABASE } { QUANT | QUANTUM }
SCHEMA	в команде CREATE SCHEMA
SECOND	элемент «даты-времени»
SERVER	в командах CREATE и DROP SERVER, CREATE REPLICATION RULE
SETS	в конструкции GROUP BY-спецификация
SIGN	функция
SIMPLE	в конструкции предиката соответствия
SIZE	в команде ALTER DATABASE
SOURCE	в команде ALTER EVENT и ALTER PROCEDURE DROP
SRID	функция
STARTPOINT	функция
STATEMENT	после ключевого слова EACH или в команде AUDIT
STATION	после начальных ключевых слов CREATE|DROP|ALTER, в командах GRANT ACCESS, REVOKE ACCESS, ALTER USER
STORAGE	в команде TRUNCATE TABLE
SUCCESS	в команде AUDIT
SYNC	в команде CREATE REPLICATION RULE
SYSTEM	в команде CREATE USER
TABLES	в команде GENDB
TEMPORARY	в команде CREATE TABLE
TEST	в команде TEST TABLE
TEXT	в команде DROP TEXT
TIME	в команде CREATE EVENT
TIMEOUT	после ключевого слова WAIT
TRACE	в команде SET TRACE
TRANSFORM	функция
TRANSLATION	в команде CREATE/DROP TRANSLATION
TYPE	в команде ALTER TABLE
UNICODE	в команде CREATE PHRASE UNICODE INDEX
UNLIMITED	в команде ALTER USER
UPPER	функция
UTF8	в команде CREATE CHARACTER
VALIDITY	в команде SET DATABASE GEODATA VALIDITY
VALUE	в команде ALTER SEQUENCE

Ключевое слово	Недопустимое использование
VARIABLE	в команде CREATE VARIABLE
WEIGHT	в контексте команд CREATE REPLICATION RULE и ALTER REPLICATION RULE
WIDTH	в команде CREATE CHARACTER
WORK	в командах COMMIT и ROLLBACK
WORKSPACE	в команде SET WORKSPACE
X	функция
Y	функция
YEAR	элемент «даты-времени»

Указатель определяемых элементов

<BLOB тип>, 27
<FROM-спецификация>, 132
<GROUP BY-спецификация>, 138
<HAVING-спецификация>, 150
<ORDER BY-спецификация>, 152
<OVER-спецификация>, 155
<select-запрос выборки>, 180
<SQL-символ>, 18
<table-запрос выборки>, 185
<UNICODE тип переменной длины>, 27
<UNICODE тип фиксированной длины>, 27
<UNICODE тип>, 27
<UNICODE-литерал>, 18
<values-запрос выборки>, 186
<WHERE-спецификация>, 135
<WITHIN GROUP-спецификация>, 156
<автоматическое наращивание значений>, 251
<агрегатная функция для интервала>, 109
<агрегатная функция>, 86
<аналитическая функция>, 96
<атрибут столбца>, 251
<байтовая конкатенация>, 43
<байтовое выражение>, 43
<байтовый литерал>, 18
<байтовый тип переменной длины>, 27
<байтовый тип фиксированной длины>, 27
<байтовый тип>, 27
<блокирование доступа к БД>, 446
<блокирование таблицы>, 445
<буква>, 18
<включение режима проверки пароля при запуске ядра>, 450
<включение режима сертифицированной БД>, 450
<включение события>, 390
<включение/отключение режима очистки оперативной памяти>, 451
<включение/отключение режима суперпользователя для DBA>, 451
<внешнее соединение>, 158
<внешний файл>, 27
<восстановление таблицы «в памяти»>, 346
<восстановление таблицы>, 338
<временное отключение всех триггеров таблицы>, 336
<временный запрет действия внешнего ключа>, 336
<временный запрет действия первичного ключа>, 336
<временный запрет действия уникального ключа>, 336
<время>, 173
<выбор значения последовательности>, 372
<вычисляемое значимое>, 251
<генерация временной последовательности>, 375
<генерируемое значение>, 251
<группировка по заданным группам>, 138
<группировка со сведением данных>, 138
<дата-время выражение>, 43
<дата-время литерал>, 18
<дата-время тип>, 27
<двоичный литерал>, 18
<двустороннее соединение>, 84
<заглавная латинская буква>, 18
<заглавная русская буква>, 18
<загрузка таблицы>, 251
<задаваемое значение>, 251
<закончить пакетное добавление>, 440
<запрос выборки>, 173
<запрос добавления>, 410
<запрос корректировки>, 420
<запрос удаления>, 426
<значение из последовательности>, 251
<значение по умолчанию>, 251
<значение>, 43, 410
<значимое выражение>, 43
<идентификатор>, 25
<идентификация сгруппированных данных>, 115
<изменение атрибутов пользователя>, 232
<изменение правила репликации>, 477
<изменение состояния триггера>, 383
<изменение уровня мандатного контроля доступа столбца>, 324
<изменение уровня мандатного контроля доступа таблицы>, 312
<изменение функциональности таблицы>, 312
<имя SQL-параметра>, 26
<имя алиаса>, 26
<имя группы>, 26
<имя индекса>, 26
<имя кодировки>, 26
<имя курсора>, 26
<имя пользователя>, 26
<имя последовательности>, 26
<имя правила>, 26
<имя псевдонима>, 26
<имя роли>, 26
<имя синонима>, 26
<имя события>, 26
<имя столбца>, 42

<имя схемы>, 26
<имя таблицы>, 41
<имя точки сохранения>, 26
<имя трансляции>, 26
<имя триггера>, 26
<имя узла>, 26
<имя устройства>, 26
<имя фильтра>, 26
<имя хранимой процедуры>, 26
<интервальный предикат>, 55
<исходная таблица>, 433
<каскадные ограничения>, 251
<кванторная функция для интервала>, 113
<классификатор>, 86
<ключ сортировки>, 152
<кодировка системного словаря>, 402
<кодировка соединения по умолчанию>, 403
<комбинированный запрос>, 194
<комментарий>, 25
<компиляция триггера>, 381
<контролируемое значение столбца>, 251
<корректировка битовой карты>, 364
<корректировка индекса>, 361
<корректировка конвертера>, 363
<латинская буква>, 18
<лексема>, 25
<литерал>, 18
<логический литерал>, 18
<логический тип>, 27
<логическое выражение>, 51
<логическое условие>, 251
<максимальный размер памяти каналов>, 462
<мандатный уровень доступа к столбцу>, 251
<мандатный уровень доступа к таблице>, 251
<метод выборки>, 132
<модификация последовательности>, 370
<модификация представления>, 351
<модификация столбца>, 324
<модификация столбцов таблицы>, 324
<модификация таблицы «в памяти»>, 347
<модификация таблицы>, 312
<модификация триггеров>, 312
<модификация файловых параметров таблицы>, 312
<назначение роли>, 250
<начать архивирование>, 472
<начать пакетное добавление>, 438
<объединение запросов>, 197
<объединение значений столбца>, 121
<объявление сервера репликации>, 474
<ограничение NULL-значений>, 251
<ограничение времени работы всех неактивных сессий СУБД>, 462
<ограничение времени работы отдельной неактивной сессии СУБД>, 463
<ограничение выборки>, 210
<ограничение длины записи>, 461
<ограничение таблицы>, 251
<ограничение уникальности>, 251
<ожидание события>, 391
<описатель столбца>, 42
<описатель таблицы>, 41
<описатель файла>, 251
<оповещение о событии>, 391
<определение привилегий на таблицы/представления/синонимы>, 353
<определение столбца>, 251
<определение таблицы «в памяти»>, 344
<опрос события>, 392
<остановить архивирование>, 474
<отмена временного запрета действия внешнего ключа>, 336
<отмена временного запрета действия первичного ключа>, 336
<отмена временного запрета действия уникального ключа>, 336
<отмена временного отключения всех триггеров таблицы>, 336
<отмена изменений>, 449
<отмена привилегий на таблицы/представления/синонимы>, 355
<отмена роли>, 251
<отмена события>, 390
<отмены/разрешения NULL-значений>, 324
<первая запись>, 106
<первичная таблица>, 132
<переименование столбца>, 324
<переименование таблицы>, 312
<пересечение запросов>, 208
<побитовая операция>, 49
<подзапрос>, 209
<подсказка для отмены итераторной стратегии>, 226
<подсказка о кэшировании запроса>, 222
<подсказка о кэшировании результата запроса>, 222
<подсказка о типе сортировки>, 220
<подсказка об отмене кэширования select-запроса>, 222
<подсказка очередности>, 214
<подсказка раскрытия>, 219
<подсказка сортировки по индексу>, 219
<подставляемое значение>, 251
<позиционная корректировка>, 429
<позиционное удаление>, 431
<пользовательская кодировка>, 404
<порожденная таблица>, 132
<порядок сортировки>, 152
<последняя запись>, 108

<последовательное ранжирование записей>, 104
<последующие данные>, 118
<предварительная загрузка таблицы>, 341
<предикат внешнего соединения в стиле ORACLE>, 84
<предикат вхождения>, 58
<предикат выборки>, 79
<предикат неверного вещественного значения>, 78
<предикат неопределенного значения>, 77
<предикат подобия>, 63
<предикат различимости>, 70
<предикат совмещения>, 82
<предикат соответствия>, 72
<предикат сопоставления>, 65
<предикат сравнения>, 52
<предикат существования>, 81
<предикат уникальности>, 62
<предикат>, 52
<представление байта>, 18
<представление даты>, 18
<представление символа>, 18
<предшествующие данные>, 117
<приближенный числовой литерал>, 18
<приближенный числовой тип>, 27
<приоритет пользователя>, 464
<проверяемая запись>, 70, 72
<псевдоним столбца>, 42
<псевдоним таблицы>, 41
<разблокирование доступа к БД>, 446
<разблокирование таблицы>, 445
<разделитель>, 25
<размер рабочей области>, 460
<разность запросов>, 207
<ранжирование с исключением дубликатов>, 102
<ранжирование с учетом дубликатов>, 97
<регулярное выражение>, 65
<режим выполнения процедур>, 466
<режим доступа>, 445
<режим проверки ограничения ссылочной целостности>, 470
<русская буква>, 18
<сброс события>, 393
<сжатие таблицы>, 339
<символьная конкатенация>, 43
<символьное выражение>, 43
<символьный литерал>, 18
<синхронизация таблиц репликации>, 478
<сличаемая запись>, 70
<сличаемые записи>, 72
<слияние без сопоставления>, 432
<слияние данных>, 432
<слияние с сопоставлением>, 432
<соединение слева>, 84
<соединение справа>, 84
<соединяемая таблица>, 158
<создание алиаса кодировки>, 407
<создание базовой таблицы>, 251
<создание глобальной временной таблицы>, 251
<создание индекса>, 356, 356
<создание кодировки>, 400
<создание комментария>, 396
<создание копии базовой таблицы>, 251
<создание пользователя>, 227
<создание последовательности>, 366
<создание правила репликации>, 476
<создание правила трансляции>, 405
<создание представления>, 349
<создание роли>, 249
<создание синонима>, 394
<создание события>, 386
<создание схемы>, 246
<создание таблицы>, 251
<создание точки сохранения>, 447
<создание триггера>, 375
<создание/изменение описания кодировки>, 408
<создание/модификация глобальной переменной простого типа>, 384
<сортировка NULL-значений>, 152
<составной запрос>, 197
<сохранение изменений>, 448
<сохранение таблицы «в памяти»>, 345
<специальный символ>, 18
<спецификация группировки>, 138
<спецификация действий>, 251
<спецификация значения по условию>, 126
<спецификация значения>, 34
<спецификация параметров таблицы>, 251
<спецификация слияния>, 432
<спецификация столбца>, 42
<спецификация таблицы>, 41
<спецификация типа>, 124
<список элементов таблицы>, 251
<ссылочная целостность столбца>, 251
<ссылочная целостность>, 251, 312
<стандартный идентификатор>, 25
<строковое выражение>, 43
<строковый тип переменной длины>, 27
<строковый тип фиксированной длины>, 27
<строковый тип>, 27
<строчная латинская буква>, 18
<строчная русская буква>, 18
<табличная ссылка>, 132
<табличное выражение>, 131
<табличный параметр>, 251
<тестирование именованного индекса>, 339

<тестирование одностолбцового индекса>, 339
<тестирование таблицы>, 339
<тестирование элементов структуры>, 339
<тип агрегатной функции>, 86, 109
<тип выборки>, 210
<тип данных>, 27
<тип кванторной функции>, 113
<тип совпадения>, 72
<тип соединения>, 158
<тип сортировки>, 152
<точный числовой литерал>, 18
<точный числовой тип>, 27
<удаление алиаса кодировки>, 408
<удаление глобальной переменной>, 386
<удаление индекса>, 360
<удаление кодировки>, 406
<удаление описания кодировки>, 409
<удаление пользователя>, 232
<удаление последовательности>, 370
<удаление правила репликации>, 477
<удаление правила трансляции>, 407
<удаление представления>, 351
<удаление роли>, 250
<удаление сервера репликации>, 475
<удаление синонима>, 395
<удаление события>, 390
<удаление схемы>, 248
<удаление таблицы>, 311
<удаление текста триггера>, 383
<удаление триггера>, 383
<универсальный идентификатор>, 25
<управление выводом сообщений процедур>, 457
<управление квантом обработки запросов в БД>, 454
<управление количеством квантов>, 454
<управление логированием BLOB-данных>, 465
<управление максимальным квантом обработки запросов в сессии>, 453
<управление максимальным квантом обработки запросов пользователя>, 452
<управление протоколированием>, 457
<управление режимом ядра>, 460
<управление трассировкой>, 458
<уровни мандатного доступа к объектам БД>, 469
<уровни по умолчанию мандатного доступа>, 467
<усечение таблицы>, 342
<условие связи>, 187
<условие соединения>, 158
<условие>, 79
<установка последовательности значений>, 324
<установка схемы>, 248
<файловый параметр>, 251
<фильтр для фразового поиска>, 251
<целевая таблица>, 433
<целочисленный литерал>, 18
<цифра>, 18
<числовое выражение>, 43
<числовой литерал>, 18
<шестнадцатеричная цифра>, 18
<шестнадцатеричный байтовый литерал>, 18
<шестнадцатеричный числовой литерал>, 18

Указатель операторов

+EXPAND, 219
+INDEX, 219
+LAST, 217
+PREDORDER, 214
+SORT, 220

A
ALTER CHARACTER SET, 408
ALTER CHARACTER SET DROP, 409
ALTER DATABASE SET CHANNEL MEMORY LIMIT, 462
ALTER DATABASE SET RECORD SIZE LIMIT, 461
ALTER DATABASE SET SESSION TIME LIMIT, 462
ALTER EVENT DISABLE, 390
ALTER EVENT ENABLE, 390
ALTER REPLICATION RULE, 477
ALTER SEQUENCE, 370
ALTER TABLE, 312, 324
ALTER TABLE DISABLE, 336
ALTER TABLE ENABLE, 336
ALTER TABLE IN-MEMORY, 347
ALTER TRIGGER, 383
ALTER TRIGGER DROP TEXT, 383
ALTER USER, 232
ALTER USER LOCK, 446
ALTER USER UNLOCK, 446

B
BACKUP DATABASE, 472
BACKUP STOP, 474

C
CASE, 126
CAST, 124
CLEAR EVENT, 393
COMMENT ON, 396
COMMIT, 448
CONNECT BY, 187
CORRECT BITMAP, 364
CORRECT INDEX, 361
CORRECT INDEX ROWID, 363
CORRESPONDING BY, 197, 207, 208
CREATE ALIAS, 407
CREATE CHARACTER SET, 400
CREATE EVENT, 386
CREATE INDEX, 356
CREATE NODE, 474
CREATE REPLICATION RULE, 476
CREATE ROLE, 249
CREATE SCHEMA, 246
CREATE SEQUENCE, 366
CREATE SERVER, 474
CREATE SYNONYM, 394
CREATE TABLE, 251
CREATE TABLE IN-MEMORY, 344
CREATE TRANSLATION, 405
CREATE TRIGGER, 375
CREATE USER, 227
CREATE VARIABLE, 384
CREATE VIEW, 349
CURRVAL, 372

D
DELETE, 426
DELETE OF CURSOR, 431
DROP ALIAS, 408
DROP CHARACTER SET, 406
DROP EVENT, 390
DROP INDEX, 360
DROP NODE, 475
DROP REPLICATION RULE, 477
DROP ROLE, 250
DROP SCHEMA, 248
DROP SEQUENCE, 370
DROP SERVER, 475
DROP SYNONYM, 395
DROP TABLE, 311
DROP TRANSLATION, 407
DROP TRIGGER, 383
DROP USER, 232
DROP VARIABLE, 386
DROP VIEW, 351

E
END APPEND, 440
EXCEPT, 207

F
FETCH FIRST, 210
FOR BROWSE, 210
FOR UPDATE, 210
FROM, 132

G
GET EVENT, 392
GRANT, 353
GRANT ROLE, 250
GROUP BY, 138

H
HAVING, 150

 © Справочник по SQL. АО НПП «РЕЛЭКС», 1990-2026

I
INSERT, 410
INTERSECT, 208

J
JOIN, 158

L
LIMIT, 210
LOCK TABLE, 445

M
MERGE, 432

N
NEXTVAL, 372

O
ORDER BY, 152
ORDER SIBLINGS BY, 187
OVER, 155

P
PRESS TABLE, 339
PRIOR, 187

R
REBUILD TABLE, 338
REBUILD TRIGGER, 381
RESTORE TABLE, 346
REVOKE, 355
REVOKE ROLE, 251
ROLLBACK, 449

S
SAVE TABLE, 345
SELECT, 180
SELECT LEVEL, 375
SET CERTIFIED, 450
SET CONSTRAINTS, 470
SET DATABASE DEFAULT CHARACTER SET, 404
SET DATABASE NAMES, 402
SET DATABASE QUANT, 454
SET DATABASE QUANTUM FOR, 454
SET EVENT, 391
SET LOG, 457
SET MEMORY CLEAR, 451
SET NAMES, 403
SET PRIORITY, 464
SET PROCEDURE TRACE, 457
SET QUANT TIMEOUT, 452
SET SAVEPOINT, 447
SET SCHEMA, 248
SET SESSION BLOB LOG, 465
SET SESSION DEFAULT SECURITY, 467
SET SESSION PROCEDURE AS DEFINER, 466
SET SESSION QUANT TIMEOUT, 453
SET SESSION QUANTUM FOR, 456
SET SESSION SECURITY, 469
SET SESSION TIME LIMIT, 463
SET STARTUP AUTHORIZATION, 450
SET SUPERUSER, 451
SET TRACE, 458
SET TRUE COMMIT, 460
SET WORKSPACE, 460
START APPEND, 438
START WITH, 187
SYNCHRONIZE REPLICATION RULE, 478

T
TABLE, 185
TEST TABLE, 339
TEST TABLE PRELOAD, 341
TRUNCATE TABLE, 342

U
UNION, 197
UNLOCK TABLE, 445
UPDATE, 420
UPDATE OF CURSOR, 429

V
VALUES, 186

W
WAIT EVENT, 391
WHERE, 135
WITHIN GROUP, 156

Указатель функций

+, 483
||, 483

A
ABS, 528
ACCESS, 617
ACOS, 530
ADD_MONTHS, 588
ANY, 95, 113
ASCII, 522
ASIN, 530
ATAN, 530
ATAN2, 530
AVG, 87, 109

B
BITAND, 50

C
CEIL, 528
CHAR_LENGTH, 506
CHARACTER_LENGTH, 506
CHR, 522
CIDRBCAST, 610
CIDRCOUNT, 608
CIDRLEN, 605
CIDRMASK, 612
CIDRMATCH, 607
CIDRNET, 609
CIDRPFX, 606
CIDRTORAW, 603
COALESCE, 626
CONCAT, 483
CONVERT, 552
COS, 530
COSH, 531
COUNT, 90, 109
COUNTBLOB, 561

D
DATESPLIT, 579
DAYNAME, 578
DBNAME, 618
DECODE, 621
DEFAULT, 94
DEFTEXT, 523
DENSE_RANK, 102
DIFFERENCE, 645
DIVTIME, 583

E
EVERY, 95, 113
EXP, 531
EXPIRATION_DATE, 619
EXTRACT, 579

F
FINDBLOB, 563
FINDRTF, 564
FIRST_VALUE, 106
FLOOR, 529
FROM_DAYS, 585

G
GET_EVENTS_STATE, 642
GETBITS, 557
GETBLOB, 570
GETBLOBSTR, 572
GETBYTE, 554
GETBYTEB, 554
GETLONG, 556
GETRAW, 559
GETSTR, 558
GETTEXT, 569
GETTEXTPOS, 565
GETWORD, 555
GETWORDB, 555
GREATEST, 538
GROUPING, 115

H
HEX, 540
HEXTORAW, 540
HTML, 551

I
INITCAP, 484
INSERT, 500, 574
INSTR, 485
IP_ABBREV, 613
IP_BROADCAST, 611
IP_HOST, 607
IP_MASKLEN, 605
IP_NETMASK, 612
IP_NETWORK, 610
IP_SET_MASKLEN, 605
IP_TEXT, 604

L
LAG, 117
LAST_DAY, 586
LAST_VALUE, 108
LEAD, 118
LEAST, 539

LENBLOB, 573
LENGTH, 506
LIN_INDEX_INFO, 637
LINTER_DICT_INFO, 640
LINTER_FILE_DEVICE, 631
LINTER_FILE_INFO, 632
LINTER_FILE_SIZE, 630
LINTER_KEYWORDS, 525
LINTER_PAGE_INFO, 634
LINTER_PAGE_TIME, 636
LINTER_VERSION, 629
LISTAGG, 121
LN, 531
LOG, 531
LOWER, 508
LPAD, 510
LTRIM, 513

M
MAC_TRUNC, 616
MACTORAW, 613
MAX, 91, 109
MEDIAN, 88, 109
MIN, 91, 109
MOD, 529
MONTHNAME, 579
MONTHS_BETWEEN, 593
MULTIME, 581

N
NANVL, 627
NEXT_DAY, 586
NULLIF, 625
NULLIFERROR, 619
NUMTODSINTERVAL, 593
NVL, 622
NVL2, 624

O
OCTET_LENGTH, 507
OVERLAY, 500

P
POSITION, 485
POWER, 532

R
RAND, 536
RANK, 97
RAWTOCIDR, 604
RAWTOHEX, 541
RAWTOMAC, 615
REGEXP_LIKE, 487
REGEXP_SUBSTR, 495
REPEAT_STRING, 520
REPLACE, 503, 576
REVERSE, 521
RIGHT_SUBSTR, 519
ROUND, 532
ROW_NUMBER, 104
RPAD, 512
RTRIM, 514

S
SET, 574, 576
SIGN, 536
SIN, 530
SINH, 531
SOME, 95, 113
SOUNDEX, 643
SQRT, 536
STDDEV, 94, 109
SUBSTR, 516
SUBSTRING, 516
SUM, 92, 109
SYS_GUID, 629

T
TAN, 530
TANH, 531
TIMEINT_FROM_DAYS, 585
TIMEINT_TO_DAYS, 584
TO_CHAR, 542
TO_DATE, 549
TO_DAYS, 584
TO_DSINTERVAL, 594
TO_GMTIME, 599
TO_LOCALTIME, 595
TO_NUMBER, 548
TO_TIMESTAMP, 549
TRANSLATE, 504
TRIM, 515
TRUNC, 534

U
UNIX_TIMESTAMP, 577
UPPER, 509

V
VARIANCE, 93, 109
VSIZE, 639

X
XML, 550