СИСТЕМА
УПРАВЛЕНИЯ
БАЗАМИ
ДАННЫХ

ЛИНТЕР®

ЛИНТЕР БАСТИОН

ЛИНТЕР СТАНДАРТ

Архитектура СУБД

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 4
Назначение документа 4
Для кого предназначен документ 4
Дополнительные документы 4
Введение 5
Компоненты СУБД ЛИНТЕР 7
Ядро СУБД 7
Транслятор SQL-запросов 9
Транслятор процедурного языка 9
Процессор сортировки 9
Сетевые средства 10
Локальный доступ по умолчанию 11
Локальный доступ с указанием идентификатора ЛИНТЕР-сервера 11
Удаленный доступ 12
Правила взаимодействия клиентского приложения и ядра СУБД ЛИНТЕР 12
Конвертация данных 13
Структура БД 14
Физическая структура 14
Логическая структура 14
Субъекты БД 15
Пользователи 15
Объекты БД 16
Роли 16
Схемы 16
Базовые таблицы 16
Временные таблицы 17
Глобальные временные таблицы 17
Вспомогательные временные таблицы 17
Таблицы «в памяти» 17
Представления 18
Последовательности 20
Синонимы 20
Индексы 20
Простые индексы 21
Составные индексы 21
Функциональные индексы 22
Временные индексы 22
Внутренняя структура индексов 22
Фразовые индексы 23
Хранимые процедуры 24
Глобальные переменные хранимых процедур 26
Триггеры 26
Средства интернационализации 28
Алиасы 29
Кэшируемые SQL-запросы 29
Предварительная загрузка таблиц 31
Словарь данных 32
Главный словарь данных 32
Дополнительные словари данных 32
Файловая структура БД 35
Файлы данных 37
Файлы индексов 37
Файлы неструктурированных объектов 39

Внешние файлы ... 40
Рабочие файлы ... 41
Характеристики СУБД ЛИНТЕР ... 43
Простые типы данных ... 43
Геометрические типы данных ... 43
Количественные характеристики ... 44
Ограничение целостности данных ... 45
Ключи ... 46
Способы представления геометрических данных ... 46
Средства обработки геометрических данных ... 47
Преобразование (кодирование) символьных данных ... 48
Защищенная БД ... 49
Хеширование пользовательских паролей ... 49
Шифрование сетевого трафика ... 49
Методы аутентификации ... 50
Механизмы обработки данных ... 52
Каналы (соединения) ... 52
Доступ к данным ... 53
Последовательный доступ ... 54
Параллельный доступ ... 54
Механизм параллельного доступа ... 54
Квант обработки запроса ... 55
Механизмы защиты данных ... 56
Контроль целостности ... 57
Авторизация пользователей ... 57
Привилегии ... 59
Роли ... 59
Мандатная защита ... 60
Контроль доступа к БД с рабочих станций ... 61
Защита ввода-вывода на внешний носитель ... 64
Протоколирование доступа к БД ... 64
Удаление остаточной информации ... 66
Механизм транзакций ... 67
Многопользовательский режим ... 67
Пессимистичный режим ... 68
Контрольные точки ... 70
Подтверждение транзакции ... 70
Откат транзакции ... 71
Системный журнал ... 71
Средства поддержки реального времени ... 73
Асинхронная обработка ... 73
Приоритетная обработка ... 73
Аппарат событий ... 75
Предварительная трансляция запроса ... 75
Приостановка и отмена обработки запросов ... 76
Пакетная обработка данных ... 77
Мониторинг использования ресурсов ... 78
Горячее резервирование БД ... 79
Автоматическое переключение серверов ... 81
Удаленное управление СУБД ... 82
Сетевое администрирование СУБД в среде ОС Windows ... 82
Удаленное управление компонентами СУБД ... 82
Локальное управление компонентами СУБД ... 84
Программные интерфейсы ... 85
Обеспечение сохранности данных ... 87

Резервное сохранение и восстановление БД 87
Автономное архивирование БД 87
Оперативное архивирование 88
Меры безопасности при архивировании БД 89
Асинхронная репликация данных 90
Механизм асинхронной репликации 90
Модель асинхронной репликации 90
Функции компонентов системы репликации 92
Параметры эффективности СУБД 94
Распределение оперативной памяти 94
Влияние размеров очередей на производительность СУБД 95
Очередь таблиц 96
Очередь столбцов таблиц 96
Очередь файлов 97
Характеристики физической структуры БД 98
Характеристики файлов таблицы 98
Множество системных номеров 100
Упаковка пользовательских данных 101
Кэш SQL-транслятора 101
Совместимость продуктов 103

Предисловие

Назначение документа

В документе приведено описание физической и логической структуры хранения данных, механизмы работы СУБД ЛИНТЕР (обработка и защита данных, режимы обработки транзакций, средства реального времени, асинхронная репликация данных), взаимодействие СУБД с прикладными пользовательскими задачами.

Рассматриваются параметры запуска и конфигурирования СУБД и их влияние на эффективность функционирования СУБД.

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

Для кого предназначен документ

Документ предназначен для администраторов СУБД ЛИНТЕР и программистов, разрабатывающих приложения с использованием данной СУБД.

Дополнительные документы

- Системные таблицы и представления
- Создание и конфигурирование базы данных
- Тестирование базы данных
- Сетевые средства
- Запуск и останов СУБД ЛИНТЕР в среде ОС Windows
- Запуск и останов СУБД ЛИНТЕР в среде ОС Linux
- Справочник по SQL
- Рекомендации по настройке СУБД ЛИНТЕР

© Архитектура СУБД. АО НПП «РЕЛЭКС», 1990-2026

Введение

СУБД ЛИНТЕР – это система управления базами данных (БД), обеспечивающая поддержку реляционной модели данных компьютерных информационных систем различного назначения (в том числе систем реального времени и систем с повышенными требованиями к надёжности, безопасности и секретности данных).

Логическими сущностями БД являются ее субъекты и объекты.

В соответствии с реляционной моделью, информация в БД логически представлена в виде двумерных таблиц. Логическая структура представления данных обеспечивает высокую степень независимости клиентских приложений от физического представления данных.

Данные в таблицах физически хранятся построчно. В одну строку (запись) могут входить данные разных типов (например: целые и вещественные числа, строки символов/байтов различной длины).

СУБД ЛИНТЕР обеспечивает следующие возможности:

- управление схемой БД;
- манипулирование данными (удаление/изменение/добавление);
- управление ограничениями целостности данных;
- стандартные и расширенные средства языка баз данных SQL;
- квантование обработки SQL-запросов;
- поддержка больших (до 2 Гбайт) неструктурированных байтовых объектов (`BLOB`);
- поддержка внешних файлов (`EXTFILE`);
- работа с временными таблицами и таблицами в памяти (in-memory);
- работа с геометрическими типами данных;
- фразовый поиск информации в текстовых данных;
- импорт/экспорт данных из/в ASCII и DBF-файлов;
- блокирование доступа к таблицам или отдельным записям;
- различные режимы обработки транзакций;
- полное или выборочное сохранение БД в архив;
- работа с претранслированными SQL-запросами (в том числе с параметрами);
- создание, запуск и отладка хранимых процедур и триггеров;
- поддержка национальных языков, как на стороне сервера, так и на стороне клиента;
- работа с элементами обработки в режиме реального времени (приоритеты выполнения транзакций, асинхронное выполнение SQL-запросов, отслеживание процессов, проходящих в СУБД, приостановка и полная остановка транзакций);
- удаленное управление компонентами СУБД;
- горячее резервирование БД;
- гибкая и надежная система защиты информации;

- поддержка программных интерфейсов для доступа к БД (интерфейсы нижнего и верхнего уровней, встроенный SQL, JDBC, ODBC, .NET, PHP, Perl, Ruby, Qt, Python).

СУБД ЛИНТЕР может функционировать в среде следующих ОС:

- семейство Linux для 32-разрядных процессоров;
- семейство Linux для 64-разрядных процессоров;
- ЗОСРВ «Нейтрино»;
- семейство Windows.



Примечание
По отдельному запросу и согласованию может быть произведена сборка СУБД ЛИНТЕР для более чем 20 программных платформ.

Компоненты СУБД ЛИНТЕР

СУБД ЛИНТЕР – это комплекс программ, работающих под управлением операционной системы. Структура программного обеспечения СУБД ЛИНТЕР приведена на рисунке 1.



Рисунок 1. Структура программного обеспечения СУБД ЛИНТЕР

Часть программ являются собственно системой управления БД (это ядро СУБД), другие представляют собой утилиты и инструментальные средства для поддержки жизненного цикла БД (администрирование, архивирование и восстановление БД, тестирование структуры БД). Центральной частью СУБД являются средства ведения БД (обработки SQL-запросов).

СУБД ЛИНТЕР использует унифицированные средства доступа к данным из клиентских приложений. Их основой является интерфейс нижнего уровня (CALL-интерфейс).

Ядро СУБД

Ядро СУБД запускается на требуемой БД. Для одновременной работы с несколькими БД необходимо запускать требуемое количество экземпляров ядра СУБД. Изменение БД в процессе работы экземпляра ядра СУБД не допустимо.

Уровни конфигурации СУБД:

- настройки БД;
- настройки ядра СУБД;
- настройки, выполняемые с помощью SQL-команд.

Ядро СУБД обрабатывает запросы квазипараллельно, то есть «разделяет» процессор, время и другие ресурсы для обработки SQL-запросов от разных клиентских приложений. Одним из самых важных ресурсов, выделяемых при обработке SQL-запроса, является «время непрерывной работы» (см. пункт «Квант обработки запроса»).

Ядро СУБД гарантированно выполняет некоторую часть обработки одного SQL-запроса (например, часть процедуры просмотра данных), и только после этого переключается на обработку SQL-запроса другого клиентского приложения (или того же самого клиентского приложения, но по другому каналу). Обработка большинства поисковых SQL-запросов сводится к просмотру одной/нескольких таблиц и, в общем случае, их индексов. Именно эта часть обработки SQL-запроса может быть прервана. СУБД, просмотрев определенное число записей при обработке SQL-запроса, запоминает состояние текущей обработки в области данных, отведенной под соответствующий канал, и переключается на обработку SQL-запроса, поступившему по другому каналу. Для ускорения обработки запроса используются индексы – двоичные «деревья», построенные по значениям индексированного столбца. Атомарная работа с «деревом» индекса (поиск, удаление, добавление значения) непрерываема. Но в промежутках между атомарными индексными операциями или после непрерывного выполнения заданного числа таких операций (настраиваемый параметр ядра СУБД) СУБД может прервать обработку текущего запроса и переключиться на обработку запроса по другому каналу.

Другим важным разделяемым ресурсом СУБД является оперативная память, которая предоставляется ядру СУБД при ее загрузке. Память предназначена для хранения системной информации об обрабатываемых SQL-запросах. В ней находятся описатели состояния каналов, открытых файлов, данные из БД.

Процессы обработки SQL-запросов используют выделенную ядру СУБД оперативную память ОС совместно: необходимые для каждого процесса данные считываются с диска, вытесняя другую (иногда нужную для продолжения обработки неактивного в данный момент времени SQL-запроса) информацию. Логически ресурс «память» представляет собой набор областей, содержащих разделяемую информацию (*очереди (кэши)* объектов БД) или неразделяемую (список каналов).

Примеры очередей (кэшей) СУБД ЛИНТЕР:

- очередь описателей таблиц;
- очередь описателей столбцов;
- очередь дескрипторов файлов;
- очередь страниц (пул ядра СУБД);
- очередь результатов поисковых запросов;
- очередь кодировок;
- очередь алиасов кодировок.

Каждая очередь имеет индивидуальные характеристики, наиболее важными из которых являются размер элемента очереди и число элементов в очереди (длина очереди). Разработчик клиентского приложения должен учитывать степень динамичности очередей, так как она оказывает серьезное влияние на эффективность работы СУБД (и, соответственно, клиентского приложения).

Следующим важным ресурсом является внешняя память.

В процессе обработки SQL-запросов СУБД хранит некоторые промежуточные результаты обработки запроса (бит-векторы ответов, сортируемую информацию)

в своих рабочих файлах. В процессе функционирования СУБД придерживается следующей стратегии разделения дискового пространства:

- пространство файла сортировки используется монопольно процессом сортировки;
- пространство файла бит-векторов выделяется по требованию ядра СУБД для обработки SQL-запроса и закрепляется за каналом до получения по этому каналу нового SQL-запроса или обнаружении ошибки в обработке текущего запроса.

Зависимость эффективности работы СУБД от размеров очередей, а также методы повышения быстродействия ядра СУБД ЛИНТЕР подробно рассмотрены в разделе «Параметры эффективности СУБД».

Ядро СУБД для выполнения SQL-операторов может запускать следующие процессы:

- транслятор SQL-запросов;
- транслятор хранимых процедур;
- программа сортировки данных.

Так как эти процессы являются независимыми, то они могут выполняться параллельно.

Рассмотрим назначение каждого процесса подробно.

Транслятор SQL-запросов

Транслятор SQL-запросов работает в последовательном режиме, то есть следующий SQL-запрос к СУБД будет транслироваться только после завершения трансляции предыдущего.

В процессе трансляции все запросы проходят этап привязки к объектам БД: проверяется наличие в БД таблиц/столбцов, на которые есть ссылка в тексте SQL-запроса, совместимость декларированных и реальных типов данных и т.д. Для выполнения этих операций SQL-транслятор выполняет простейшие, не требующими трансляции, команды интерфейса нижнего уровня. Эти команды непрерываемы, не распараллеливаются с обработкой других команд, хотя и посылаются по параллельным каналам. Предпочтительно использование претранслированных SQL-запросов.

Транслятор процедурного языка

Транслятор процедурного языка предназначен для трансляции текстов триггеров, хранимых и временных процедур. Исходный текст триггеров и хранимых процедур хранится в БД. Он работает в последовательном режиме, то есть следующий объект трансляции будет обрабатываться только после завершения трансляции предыдущего. Однако трансляция очередного триггера (или хранимой процедуры) будет выполняться параллельно с обработкой ядром СУБД SQL-запросов или ранее оттранслированных триггеров (хранимых процедур).

Процессор сортировки

Процессор сортировки – тоже отдельная программа ядра СУБД ЛИНТЕР, поэтому сортировка результатов поисковых SQL-запросов выполняется параллельно с другими рабочими процессами: обработкой ядром СУБД претранслированного запроса и трансляцией следующего запроса. Однако собственно сортировка (так же, как

и трансляция) может выполняться только последовательно, то есть сортировка результатов очередного SQL-запроса начнется только после окончания сортировки текущего SQL-запроса. Для ускорения обработки сортируемых поисковых запросов СУБД ЛИНТЕР может параллельно запускать несколько процессоров сортировки. Однако если количество сортируемых поисковых запросов превысит допустимое количество процессоров сортировки, то очередной запрос, требующий сортировки, будет ожидать освобождения процессора сортировки и его рабочего файла. Это ожидание не повлияет на обработку других SQL-запросов, не требующих сортировки.

Примечание
СУБД ЛИНТЕР обеспечивает поддержку 256 процессов сортировки. Реальное количество зависит от ограничений, накладываемых операционной системой.

Сетевые средства

Сетевые средства СУБД ЛИНТЕР реализуют модель взаимодействия «клиент-сервер» и предназначены для организации доступа клиентских приложений к локальным или удаленным ЛИНТЕР-серверам.

Сетевые средства СУБД ЛИНТЕР представлены сетевыми драйверами (сервера и клиента) и файлом сетевой конфигурации. Сетевые средства работают со следующими протоколами:

- `TCPIP` – сетевой протокол Internet;
- `TCPIPS` – протокол TCP/IP в режиме защищенного SSL-соединения;
- `TLS` – протокол TCP/IP в режиме защищенного TLS-соединения;
- `LOCAL` – протокол доступа к локальному ЛИНТЕР-серверу;
- `ATCPIP` – протокол TCP/IP для использования в системе репликации (асинхронного тиражирования);
- `ATCPIPS` – протокол TCP/IP для использования в системе репликации (асинхронного тиражирования) в режиме защищенного TLS-соединения;
- `REZ` – протокол системы резервирования.

ЛИНТЕР-сервер – экземпляр ядра СУБД запущенный на требуемой БД.

Сетевой драйвер сервера предназначен для передачи запросов удаленных клиентов ядру СУБД.

Сетевой драйвер клиента предназначен для передачи запросов локальных клиентов к удаленным серверам СУБД через сетевой драйвер сервера или к локальным серверам СУБД.

Файл сетевой конфигурации – файл, в котором устанавливается соответствие между реальными сетевыми параметрами сервера (сетевой адрес и порт) и его символическим именем, используемым в клиентских приложениях для соединения с нужным ЛИНТЕР-сервером.

Возможны следующие варианты взаимодействия:

- локальное по умолчанию;
- локальное с указанием идентификатора ЛИНТЕР-сервера;

- удаленное.

Локальный доступ по умолчанию

Схема локального взаимодействия по умолчанию приведена на рисунке 2.

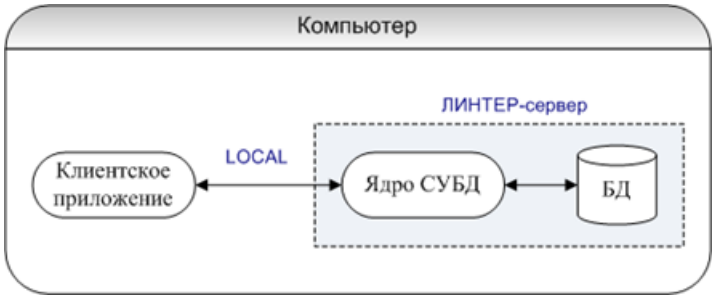


Рисунок 2. Конфигурация локального доступа по умолчанию

Особенности организации локального доступа по умолчанию:

- при запуске ядра СУБД не надо знать идентификатор ЛИНТЕР-сервера;
- не требуется запускать сетевые средства;
- возможен запуск лишь одного экземпляра ядра СУБД.

Локальный доступ с указанием идентификатора ЛИНТЕР-сервера

Схема локального взаимодействия с указанием идентификатора ЛИНТЕР-сервера приведена на рисунке 3.

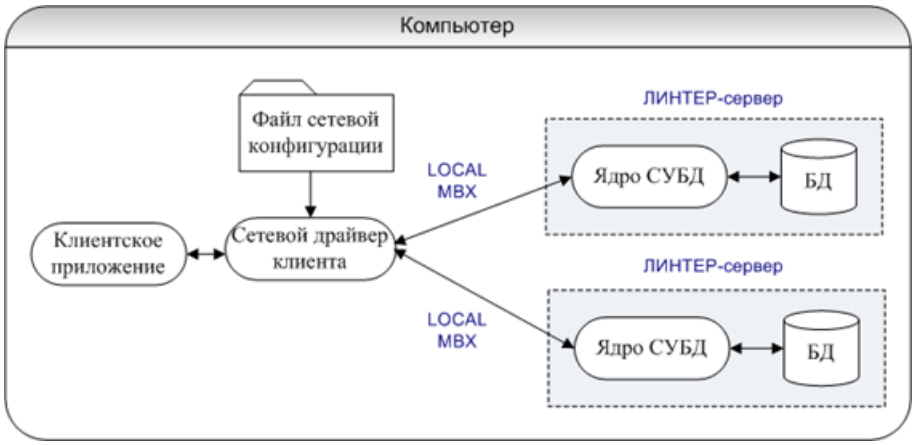


Рисунок 3. Конфигурация локального доступа с указанием идентификатора ЛИНТЕР-сервера

Особенности организации локального доступа с указанием идентификатора ЛИНТЕР-сервера:

- при запуске ядра СУБД необходимо указывать уникальный идентификатора ЛИНТЕР-сервера;
- необходимо сформировать файл сетевой конфигурации;
- требуется запуск сетевого драйвера клиента;

- возможен локальный запуск требуемого числа экземпляров ядра СУБД.

Удаленный доступ

Схема удаленного доступа приведена на рисунке 4.

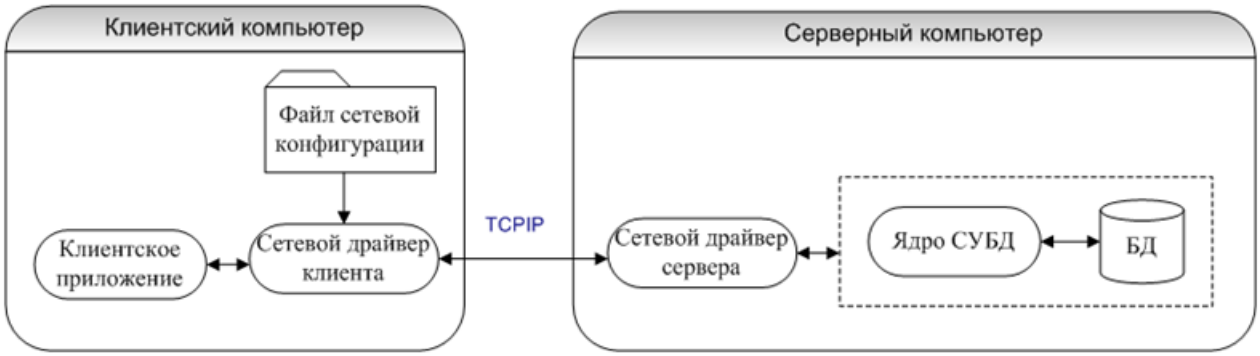


Рисунок 4. Конфигурация удаленного сетевого доступа

Для организации удаленного сетевого доступа необходимо:

1) настроить серверную часть на серверном компьютере:
   - запустить ядро СУБД;
   - запустить сетевой драйвер сервера (утилита `dbs_tcp`).
2) настроить клиентскую часть на клиентском компьютере:
   - зарегистрировать требуемые ЛИНТЕР-сервера в файле сетевой конфигурации `nodetab`. В этом файле устанавливается соответствие между реальным сетевым адресом ЛИНТЕР-сервера и его символическим именем (идентификатором), используемым в клиентских приложениях для соединения с требуемым ЛИНТЕР-сервером;
   - запустить сетевой драйвер клиента (утилита `dbc_tcp`).

Для автоматизации сетевого управления в среде ОС Windows рекомендуется использовать утилиту «Сетевой администратор СУБД ЛИНТЕР» (`linadm`).

Правила взаимодействия клиентского приложения и ядра СУБД ЛИНТЕР

Взаимодействие клиентского приложения и ядра СУБД ЛИНТЕР осуществляется по следующим правилам:

- сетевой драйвер клиента, запущенный на компьютере с клиентским приложением, принимает команды интерфейса нижнего уровня, посланные клиентским приложением удаленному ЛИНТЕР-серверу, и перенаправляет их по сети сетевому драйверу сервера или локальной БД (если используется локальный протокол обмена данными `local`);
- сетевой драйвер сервера принимает команду и (как любое клиентское приложение) отправляет ее ядру СУБД.

Соответствие имени удаленного ЛИНТЕР-сервера и его сетевого адреса задается в конфигурационном файле сетевого клиента (`nodetab`).

Оба драйвера работают в автоматическом режиме.

Таким образом, схема прохождения команды от клиентского приложения на узле-клиенте к ядру СУБД на узле-сервере выглядит следующим образом (выделенное квадратными скобками не всегда является обязательным):

```
Приложение -> Драйвер клиента -> [Драйвер сервера] -> ядро СУБД
```

Результаты обработки запроса (ответы) проходят обратный путь:

```
ядро СУБД -> [Драйвер сервера] -> Драйвер клиента -> Приложение
```

Если имя удаленного ЛИНТЕР-сервера на задано явно, то клиентское приложение «не видит» разницы между собственно ядром СУБД и сетевым драйвером клиента. Приложение подаёт запрос, не зная, кто его получит (ядро СУБД или сетевой драйвер клиента). Аналогичным образом принимаются и ответы – все равно от кого: от ядра СУБД или от драйвера сервера.

Такой механизм взаимодействия позволяет отлаживать клиентское приложение на локальном компьютере и впоследствии без изменения кода переносить его на удаленный компьютер.

Если клиентскому приложению необходимо использовать несколько ЛИНТЕР-серверов (например, для переноса данных из одной БД в другую), то ему придётся явно указывать логическое имя ЛИНТЕР-сервера, к которому относится посылаемая команда. Привязка логического имени ЛИНТЕР-сервера и его сетевого адреса осуществляется в конфигурационном файле (nodetab).

СУБД ЛИНТЕР позволяет кодировать (шифровать) передаваемые ею по сети данные. Кодирование/декодирование выполняется сетевыми драйверами клиента и сервера (шифрование возможно только в том случае, если установлен соответствующий протокол обмена данными).

Конвертация данных

Процесс перехода от различных реляционных БД к БД ЛИНТЕР (а также при переходе от одной версии СУБД ЛИНТЕР к другой) может быть упрощён с помощью набора предназначенных для этих целей инструментальных средств:

- утилит конвертации данных, работающих через ODBC и .NET интерфейсы;
- конвертера для загрузки таблицы БД ЛИНТЕР из текстового файла DBF-формата.

Структура БД

Структура БД ЛИНТЕР может быть представлена в виде физической и логической структур, которые в совокупности образуют собственно БД. Такое разделение позволяет управлять физической структурой данных, не изменяя доступ к логическим структурам данных.

Физическая структура

Физическая структура БД ЛИНТЕР реализуется файлами ОС, в которой функционирует СУБД ЛИНТЕР. Каждая БД ЛИНТЕР состоит из файлов следующих типов:

- файл данных таблиц;
- индексный файл таблиц;
- файл BLOB-данных таблицы;
- файл фразовых индексов таблицы;
- файл системного журнала;
- системные файлы (файл бит-векторов, файл сортировки, рабочий файл).

Также БД может содержать ссылки, хранимые в столбцах типа данных `EXTFILE` («внешний файл»), на файловые объекты файловой системы ОС. Внешние файлы не являются частью БД, поэтому при архивировании БД они не архивируются.

Логическая структура

В основе логической структуры БД ЛИНТЕР лежат понятия субъектов и объектов БД. Субъектом БД является любой зарегистрированный в ней пользователь. СУБД ЛИНТЕР может работать только с идентифицированным пользователем. Пользователь является владельцем тех объектов БД, которые он создал. Объекты БД и отношения между ними формируют логическую структуру БД. Создать объект БД, не принадлежащий ни одному пользователю, невозможно.

В соответствии с реляционной моделью данные в БД логически представлены в виде двумерных таблиц.

© Архитектура СУБД. АО НПП «РЕЛЭКС», 1990-2026

Субъекты БД

Субъектами БД являются 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

Роли

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).

Схемы

Схема – это именованная группа объектов БД, имеющая владельца.

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

Создавать объекты БД в схеме может только создатель схемы.

Для каждого пользователя БД всегда создается схема, имя которой совпадает с его именем. Дополнительно могут создаваться и другие схемы (например, схема «Справочники» содержит перечень доступных всем пользователям БД набор таблиц нормативно-справочной информации).

Базовые таблицы

Базовая таблица – это основной объект хранения данных в БД ЛИНТЕР.

Таблица состоит из *столбцов* и *строк* (или записей), на пересечении которых находятся *поля* строк (значения), соответствующие тому или иному конкретному столбцу.

Кроме собственно данных, с каждой таблицей связан набор *метаданных*, в котором хранится системная информация о структуре таблицы и ее атрибутах (схема таблицы, имя таблицы, информация о ее расположении на носителе данных, число столбцов, их имена, свойства и др.).

Базовые таблицы разделяются на системные и пользовательские таблицы.

Системные таблицы предназначены для хранения метаданных, например, обо всех таблицах БД, обо всех столбцах таблиц, о полномочиях пользователей БД. Они разделяются на две группы: обязательные системные таблицы и дополнительные. Обязательные системные таблицы создаются автоматически при генерации БД – без них СУБД ЛИНТЕР работать не сможет. Дополнительные системные таблицы предназначены для функциональной поддержки специфических возможностей СУБД ЛИНТЕР (национальные кодировки, полнотекстовый поиск

и т.п.). Необходимость создания дополнительных системных таблиц определяет администратор БД в зависимости от функциональных потребностей пользователей БД. Создание таких таблиц выполняется с помощью соответствующих SQL-скриптов.

Пользовательские таблицы создаются пользователем БД. Каждая пользовательская таблица имеет *владельца*, то есть пользователя БД, который её создал и схему, в которой она создана.

Временные таблицы

Временные таблицы используются для непостоянного (временного) хранения в БД обрабатываемых данных. Они подразделяются на глобальные и вспомогательные временные таблицы.

Глобальные временные таблицы

Глобальные временные таблицы имеют постоянно сохраняемую в БД схему определения (до тех пор, пока глобальная временная таблица, подобно базовой, не будет удалена с помощью SQL-оператора удаления таблицы).

Данные, находящиеся в глобальной временной таблице, не видны из другого пользовательского сеанса (если даже этот сеанс инициирован тем же самым пользователем). Запрещается удалять глобальную временную таблицу, модифицировать её схему определения, изменять или удалять связанные с ней триггеры, если она используется хотя бы в одном сеансе.

Для глобальной временной таблицы возможно создание первичных и уникальных ключей, обычных и составных индексов, а также внешних ключей, ссылающихся на другие таблицы. Внешние ссылки на глобальную временную таблицу создавать нельзя.

Вспомогательные временные таблицы

Вспомогательные временные таблицы создаются, по необходимости, самой СУБД с целью оптимизации выполнения SQL-запросов. Они используются для записи промежуточных результатов или в качестве рабочего пространства. Явное обращение пользователей к ним запрещено.

Данные во вспомогательных временных таблицах не сохраняются в БД. Они автоматически очищаются в конце сеанса работы пользователя с БД.

Таблицы «в памяти»

Таблица «в памяти» (in-memory таблица) – это таблица, данные и метаданные которой полностью хранятся в оперативной памяти ядра СУБД ЛИНТЕР в течение всего сеанса работы с этой таблицей и не синхронизируются автоматически с ее данными и метаданными, хранящимися в файлах на диске.

Использование таблиц «в памяти» позволяет улучшить быстродействие СУБД как при выполнении операций манипулирования данными (добавление, обновление, удаление, быстрая загрузка данных), так и отчасти при обработке SELECT-запросов. Средствами улучшения быстродействия СУБД при использовании таблиц «в памяти» являются:

- блокировка элементов системных очередей и страниц в пуле ядра СУБД, что уменьшает накладные расходы на поиск элементов в очередях и пуле и повторное считывание их с диска;

- отказ от журнализации большинства изменений таблиц «в памяти» при выполнении над ними операций манипулирования данными.

Это означает, что обычный транзакционный механизм обработки данных на таблицы «в памяти» не распространяется, то есть команды `COMMIT` и `ROLLBACK` для этих таблиц не применяются, вместо них для подтверждения внесенных изменений следует выполнять операцию `SAVE`, а для отмены операцию `RESTORE`. Однако, если при работе с таблицей «в памяти» происходит ошибка, делающая невозможным ее последующее сохранение, то выполняется неявная операция `RESTORE` и выдается соответствующее сообщение на консоль ядра СУБД и в файл протоколирования работы ядра СУБД `linter.out`.

Для таблиц «в памяти» существуют три стандартные процедуры, которые выполняет над ними СУБД ЛИНТЕР – активизация, сохранение и сброс, и два флага состояния – активизирована ли таблица в текущий момент и была ли она сохранена на диске.

Чтобы начать работу с таблицами «в памяти» необходимо не забыть выполнить ряд настроек:

1) до начала работы сконфигурировать БД с помощью утилиты `gendb` (до запуска ядра на БД):
   - задать размер очереди для таблиц «в памяти» (команда вида `SET IN-MEMORY TABLES 50;`);
   - задать количество столбцов у таблиц «в памяти» (команда вида `SET IN-MEMORY COLUMNS 250;`);
   - задать количество файлов (команда вида `SET IN-MEMORY FILES 50;`);
2) запуск ядра производить с ключом `/INMEMPOOL` (ключ вида `/INMEMPOOL=100000`, где значение задается в страницах по 4 Кбайт).

Для того чтобы выполнить любое обращение к таблице «в памяти», необходимо её сначала активизировать – загрузить с диска в оперативную память СУБД и заблокировать там. Если активизация по какой-то причине не удается, то дальнейшая работа с такой таблицей «в памяти» либо запрещается, либо выполняется как с обычной базовой таблицей.

Таблица «в памяти» может быть сохранена на диск. Если таблица «в памяти» была сохранена на диске, то у нее существуют файлы, и при ее активизации в пул страниц ядра СУБД считывается именно содержимое этих файлов. Сохранены должны быть обязательно все файлы таблицы, а не часть их. Если таблица «в памяти» не была сохранена на диске, то для СУБД ее файлы не существуют на диске – являются фиктивными, как файлы временных таблиц.

Для автоматической загрузки и сохранения данных таблиц «в памяти» можно воспользоваться опциями `AUTOLOAD` и `AUTOSAVE` для таблиц «в памяти».

Для таблиц «в памяти» есть ограничения функциональности, например, нельзя использовать ссылочную целостность и триггеры.

Представления

Представление (`VIEW`) – это виртуальная, то есть не существующая в БД таблица, которая ссылается на данные, фактически хранящиеся в одной или нескольких базовых таблицах. С точки зрения СУБД представлением является хранимый SELECT-запрос (с логикой соединения входящих в него базовых таблиц и фильтрации данных), из результирующей выборки которого можно выбирать данные, как если бы это была

базовая таблица. Исходными таблицами представления могут быть не только базовые таблицы, но и другие представления.

Представления могут быть обновляемыми (выборка данных выполняется из одного объекта БД) и необновляемыми (выборка данных выполняется одновременно из нескольких логически связанных объектов БД).

При работе с обновляемым представлением допустимы те же операции обработки данных, что и при работе с базовыми таблицами: формирование поисковых запросов, вставка, удаление, модификация данных.

При работе с необновляемым представлением допустимы только поисковые запросы.

Для повышения скорости обработки поисковых запросов может применяться механизм материализованных представлений.

Материализация означает, что при создании материализованного представления СУБД создает временную таблицу, заполняя её данными SELECT-запроса, хранящегося в теле определения представления.

Для поддержания актуальности данных материализованного представления СУБД автоматически отслеживает каждое изменение данных в исходных таблицах, задействованных в представлении, и при наличии изменений выставляет признак «представление, нуждающееся в обновлении зависимостей». В этом случае обновление материализованного представления будет выполнено при очередном обращении к нему.

SELECT-запросы к материализованным представлениям обрабатываются с той же скоростью, что и запросы к реальным таблицам БД.

Материализованные представления рекомендуется применять при использовании сложных SELECT-запросов (в частности, `SELECT` с `DISTINCT`, `UNION`, `GROUP BY` и `HAVING`) к таблицам (выборкам) большого объема в случае, если эти таблицы обновляются сравнительно редко.

Все операции, связанные с изменением данных в обычном (не материализованном) представлении, в действительности затрагивают исходные таблицы этого представления.

Применение представлений обусловлено следующими их свойствами:

- представления обеспечивают дополнительный уровень защиты информации, ограничивая доступ к предопределенным множествам строк и столбцов исходной таблицы. Например, представление можно составить таким образом, чтобы некоторые столбцы исходной базовой таблицы со специфической информацией не включались в определение представления;
- представления позволяют скрыть структуру БД. Например, единственное представление может служить для построения соединения, которое является отображением взаимосвязанных столбцов или строк из нескольких таблиц. Такое представление скрывает тот факт, что эти данные на самом деле принадлежат разным таблицам;
- представления помогают упростить составление SQL-запросов. Например, с помощью представления пользователь может выбирать информацию из нескольких таблиц, не зная, как осуществлять сложный коррелированный запрос;
- с помощью представления можно переименовать столбцы, не затрагивая самих таблиц, на которых базируется представление;

- с помощью материализованных представлений можно существенно повысить скорость выполнения поисковых запросов из взаимосвязанных таблиц большого объема со сложными условиями выборки.

Последовательности

Для автоматической генерации последовательных значений заданного типа, уникальных в пределах всей БД (а не одной таблицы), используются *последовательности*. Сгенерированные с помощью последовательности значения могут присваиваться полям добавляемой в таблицу строки. Использование последовательностей для генерации уникальных значений устраняет очередь, которая возникала бы в многопользовательской среде при генерации значений программными средствами.

Например, предположим, что двое пользователей БД одновременно вставляют записи о новых сотрудниках в таблицу "КАДРЫ". Благодаря использованию последовательности для генерации уникальных номеров сотрудников для столбца "ТАБЕЛЬНЫЙ НОМЕР" никто из них не будет ожидать завершения операции другого, чтобы ввести очередной свободный номер сотрудника. Последовательность автоматически генерирует правильное значение для каждого из пользователей.

Последовательность может быть общей или личной. Каждый пользователь может создать *личную последовательность*. Администраторы БД создают *общие последовательности*. Для работы с последовательностями в БД должна быть системная таблица $$$SEQ, для создания которой должен быть выполнен SQL-скрипт systab.sql, расположенный в подкаталоге \dict установочного каталога СУБД ЛИНТЕР.

Синонимы

Синоним – это дополнительное имя базовой таблицы или представления. Синоним – это не объект БД, он является прямой ссылкой на объект (то есть физически не создается еще одна таблица или представление).

Синонимы используются для следующих целей:

- маскировки действительного имени и схемы объекта;
- обеспечения общего доступа к объекту;
- упрощения создания SQL-запросов для пользователей БД (например, длинное имя таблицы в SQL-запросе заменяется ее коротким синонимом).

Синоним может быть общим или личным. Каждый пользователь БД может создать личный синоним. Администраторы БД создают общие синонимы.

Индексы

Индекс – это специальная надстройка над таблицей, предназначенная для повышения скорости обработки поисковых запросов, связанных с этой таблицей. Индекс (если он существует) всегда является частью таблицы.

Например, запрос

```
SELECT * FROM "КАДРЫ" WHERE "ТАБЕЛЬНЫЙ НОМЕР" = 101;
```

будет работать быстрее, если по столбцу "ТАБЕЛЬНЫЙ НОМЕР" построить индекс.

 © Архитектура СУБД. АО НПП «РЕЛЭКС», 1990-2026

При обработке поискового запроса СУБД ЛИНТЕР может использовать индексы для эффективного поиска данных. В некоторых случаях СУБД по собственной инициативе создает индексы для ускорения обработки запроса (пункт «Временные индексы»).

Индексы наиболее полезны для поиска значений в следующих случаях:

- по условиям =, >, <, >=, <=;
- попадание в диапазон (`BETWEEN`);
- попадание в список (`IN`);
- поиск на соответствие шаблону (`LIKE`) если первый символ шаблона не является метасимволом.

Как правило, индексы бесполезны для поиска по условию <> или `LIKE`, если первый символ шаблона является метасимволом.

Созданный индекс автоматически поддерживается и используется СУБД ЛИНТЕР.

Изменения в данных таблицы (такие, как добавление новых строк, обновление или удаление строк) автоматически отражаются во всех соответствующих индексах.

Примечание
Не производятся изменения в индексах на системные таблицы `$$$SYSRL` и `$$$ATTRI`, касающиеся вспомогательных временных таблиц и их столбцов.

Индексы логически и физически независимы от данных. Их можно удалять и создавать в любой момент, не оказывая влияния на данные в таблице или другие индексы. После удаления индекса все клиентские приложения будут функционировать по-прежнему, однако доступ к ранее индексированным данным может быть замедлен. Это правило не относится к автоматически создаваемым самой СУБД ЛИНТЕР индексам для поддержки первичных, уникальных и внешних ключей. Удаление таких индексов делает невозможным добавление и удаление записей таблицы.

Индексы могут быть созданы по одному или нескольким столбцам таблицы, а также по отдельным словам (или фразам) текстовых документов.

Для тех версий ОС, которые не поддерживают работу с файлами размером больше 2 Гбайт, индексы можно размещать в нескольких индексных файлах.

Простые индексы

Индекс, созданный по одному столбцу таблицы, является *простым* (одностолбцовым) индексом. Некоторые простые индексы строятся СУБД автоматически, например, если столбец определен как первичный ключ (Primary key) или внешний ключ (Foreign key). Другие индексы должны создаваться пользователем БД вручную с помощью соответствующих SQL-операторов.

Составные индексы

Индекс, созданный по нескольким столбцам таблицы, является *составным* (многостолбцовым) индексом.

Столбцы в составном индексе могут задаваться в произвольном порядке, и не обязаны быть соседними в таблице.

Составные индексы могут ускорить поиск данных, если условие WHERE в SELECT-запросе относится ко всем или к ведущим столбцам составного индекса, поэтому при выборе порядка столбцов при создании составного индекса необходимо проанализировать наиболее часто используемые поисковые запросы. Для эффективного использования составного индекса чаще используемые и наиболее селективные столбцы должны идти первыми.

Функциональные индексы

Индекс, создаваемый по набору результатов некоторой SQL-функции, называется *функциональным* индексом. То есть функциональный индекс строится не на базе набора значений некоего столбца/столбцов, а на базе набора значений функции от указанных столбцов. Как правило, функциональный индекс должен создаваться для значений функции, наиболее часто используемых в операциях сравнения. Например, при сравнении строковых данных без учета регистра символов часто используется функция UPPER. Создание функционального индекса с функцией UPPER улучшает эффективность таких сравнений.

Примечание
В текущей версии СУБД ЛИНТЕР поддерживается создание функциональных индексов только для функций UPPER и LOWER в одностолбцовом варианте.

Временные индексы

Временные индексы могут создаваться самой СУБД ЛИНТЕР при обработке многопеременных предикатов (то есть таких, в которых есть обращения к столбцам двух и более таблиц). Решение о создании временного индекса осуществляется на основе анализа поискового запроса.

Например, при обработке запроса вида

```
SELECT ... FROM A, B WHERE ... AND A.X=B.Y AND ...
```

для предиката A.X=B.Y проверяется:

- наличие индексов на столбцы A.X и B.Y;
- число записей в каждой из этих таблиц;
- число записей в каждой из этих таблиц, удовлетворяющих уже проверенным условиям.

В результате может быть принято решение построить временный индекс на множество значений столбца A.X (B.Y), соответствующих отобранным записям таблицы X (Y). Альтернативные решения – использовать существующий индекс (если он есть), либо какую-то таблицу сканировать полностью (обычно в случае ее малого размера).

Временный индекс является одностолбцовым. Он создаётся в рабочем файле (1.31) и автоматически удаляется по окончании обработки запроса, для выполнения которого он был построен.

Внутренняя структура индексов

СУБД ЛИНТЕР использует для индексов B*-деревья, сбалансированные таким образом, чтобы время доступа к любой записи было одинаковым. Верхние блоки (блоки ветвей)

B*-дерева индекса содержат данные индекса, указывающие на блоки индекса более низкого уровня. Блоки индекса низшего уровня (блоки листьев) содержат каждое индексированное значение данных и соответствующий `ROWID` (системный номер записи), используемый для нахождения записи; блоки листьев связаны в двусвязный список. Индексы по столбцам, содержащим символьные данные, базируются на двоичных значениях символов в наборе символов БД.

Для уникального индекса на каждое значение данных существует один `ROWID`. Для неуникального индекса `ROWID` включен в ключ. Неуникальные индексы отсортированы по ключу и по `ROWID`. Значения ключей, состоящие только из пустых значений, не индексируются. Для уникального индекса две строки не могут состоять из одних пустых значений.

Структура B*-дерева имеет следующие преимущества:

- все блоки листьев в дереве имеют одинаковую глубину, так что извлечение любой записи из любого места индекса требует приблизительно одинакового времени;
- B*-деревья индексов автоматически балансируются;
- все блоки в B*-дереве в среднем заполнены на 3/4;
- B*-деревья обеспечивают высокую скорость извлечения данных для широкого спектра запросов, включая поиск по точному совпадению и по диапазону значений;
- вставки, обновления и удаления эффективны и поддерживают порядок ключей для быстрого извлечения;
- производительность B*-деревьев не падает с ростом размера таблиц.

Фразовые индексы

Быстрый поиск текстовых документов в БД возможен только с помощью индексирования входящих в документ слов. Перед индексацией тексты должны быть преобразованы к стандартному представлению, исключающему элементы форматирования.

Поскольку в качестве документов могут использоваться документы разных форматов (TXT, DOC, RTF, PDF, HTML), то СУБД ЛИНТЕР использует набор специализированных *фильтров*. Фильтром в СУБД ЛИНТЕР является компонент ядра СУБД (динамическая библиотека – для внешних фильтров; модуль ядра – для встроенных фильтров), который извлекает из документа его текстовое содержимое (в виде потока слов) и свойства (автор, размер, дата создания). На вход фильтру подается исходный документ, на выходе возвращается «чистый» текст (без всяких элементов форматирования). Схема работы фильтра на примере документов WinWord приведена на рисунке 5. Нужные фильтры устанавливаются при настройке СУБД или непосредственно пользователем БД с помощью специальных SQL-операторов.

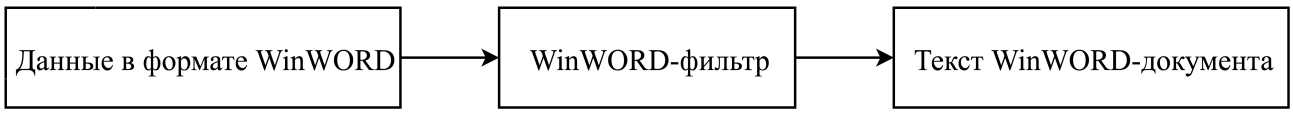


Рисунок 5. Схема фильтрации WinWord-документов в СУБД ЛИНТЕР

СУБД ЛИНТЕР имеет набор встроенных фильтров для некоторых наиболее распространенных форматов документов, а именно: ASCII (в том числе, в кодировке ANSI, KOI8-R), HTML/XML (в коде ASCII), UNICODE, HTML/XML (в коде UNICODE), RTF, PDF, XLS, XLSX, PPT, PPTX, DOC, DOCX, Open Office, PS.

Тем не менее, не все они обязательно должны использоваться. Это потенциально возможный набор. Чтобы фильтр стал доступным для применения, информация о нем должна быть помещена в БД специальным SQL-запросом. При необходимости пользователи БД могут разрабатывать собственные фильтры для своих специфических форматов данных и включать их в состав СУБД. Для поддержки полнотекстового поиска должны быть выполнены SQL-скрипты `search.sql` и `default.sql`.

Для данных типа `EXTFILE` (внешний файл) СУБД может автоматически подобрать нужный фильтр в зависимости от расширения файла.

Все поисковые системы производят поиск с помощью заранее построенного индекса, поэтому слова документов, участвующих в поиске, должны быть предварительно внесены в индекс. Для индексирования используются следующие методы:

- обычное индексирование: содержание поля заносится в индекс целиком. Из словосочетания «база данных» формируется одно индексное значение – «база данных». В этом случае для правильного поиска нужно вводить все слова в соответствующем порядке и с соответствующими разделителями. Предикаты `LIKE` и `SIMILAR` также позволяют использовать обычный индекс для поиска данных, соответствующих шаблонам;
- фразовое, или полнотекстовое индексирование: каждое слово, входящее в документ, индексируется отдельно, разделители не учитываются. Таким образом, из словосочетания «база данных» в индекс войдут два слова: «база» и «данных». Предикат фразового поиска `CONTAINS` требует обязательного существования соответствующего фразового индекса. При поиске с использованием этого предиката могут как задаваться, так и не задаваться порядок введенных слов, расстояние между словами и т.д.

При поиске данных (например, документов), содержащих указанные в условии поиска слова, СУБД ЛИНТЕР будет автоматически использовать построенный индекс.

Хранимые процедуры

Хранимой процедурой называется некоторый алгоритм, записанный на процедурном языке, сохраненный в БД и способный исполняться СУБД. Хранимые процедуры сочетают легкость и гибкость языка SQL с процедурными возможностями языка структурного программирования.

Хранимые процедуры можно использовать для реализации части логики приложения, выполнение которой перекладывается с клиентского приложения на сервер БД. Как правило, эта логика касается доступа к БД, что позволяет унифицировать определенные алгоритмы обработки данных, переведя их однажды на сторону сервера и контролируя на сервере.

Свойства хранимых процедур:

- хранимые процедуры, как и другие объекты БД, принадлежат владельцу (создателю), и должны иметь уникальное имя в схеме;
- владелец процедуры может предоставить право ее запуска другим пользователям (в том числе с правами владельца);
- способны обрабатывать атомарные данные всех типов данных, поддерживаемых СУБД ЛИНТЕР;
- способны обрабатывать выборки данных (курсоры), полученные как результат выполнения SELECT-запроса;

- могут принимать входные и выходные параметры атомарных типов данных;
- могут возвращать значение атомарного типа данных или выборку (курсор);
- обеспечивают возможность работы с глобальными переменными хранимых процедур;
- обеспечивают реализацию последовательных, ветвящихся и циклических алгоритмов;
- допускают использование в качестве элементов процедурного языка выражения различных типов и SQL-запросы к БД;
- обеспечивают обработку ошибочных ситуаций при помощи механизма исключений;
- явно вызываются на выполнение пользователем при помощи SQL-запроса или по имени внутри другой хранимой процедуры или триггера;
- могут быть созданы с поддержкой отладочной информации, что позволяет создателю процедуры отлаживать ее выполнение на сервере СУБД с использованием входящего в состав СУБД ЛИНТЕР отладчика триггеров и хранимых процедур.

В качестве входных параметров можно использовать любые выражения.

Список поддерживаемых хранимыми процедурами СУБД ЛИНТЕР типов данных приведен в таблице 1.

Таблица 1. Типы данных, поддерживаемые хранимыми процедурами

Тип данных	Значение
INT или INTEGER	4-х байтовое целое
SMALLINT	2-х байтовое целое
BIGINT	8-ми байтовое целое
REAL	4-х байтовое с плавающей точкой
DOUBLE	8-ми байтовое с плавающей точкой
NUMERIC (DECIMAL)	Число с фиксированной точкой
CHAR(размер)	Строки фиксированной длины (до 4000 символов), при работе с объектами БД дополняются пробелами до заданной длины
VARCHAR(размер)	Строки переменной длины (до 4000 символов), пробелами не дополняются
NCHAR(размер)	Строки фиксированной длины 2-х байтных UNICODE-символов (до 2000 символов), при работе с объектами БД дополняются пробелами до заданной длины
NVARCHAR(размер)	Строки переменной длины 2-х байтных UNICODE-символов (до 2000 символов), пробелами не дополняются
DATE	Дата и время
BOOLEAN	Логическое (TRUE/FALSE)
BYTE(размер)	Последовательность байт фиксированной длины (до 3919 байт), при работе с объектами БД дополняются двоичными нулями до заданной длины
VARBYTE(размер)	Последовательность байт переменной длины (до 3919 байт), двоичными нулями не дополняются
CURSOR	Представление результата поискового запроса

Тип данных	Значение
BLOB	Поддержка BLOB-данных выполняется с помощью встроенных функций

При создании хранимой процедуры ядро СУБД ЛИНТЕР вызывает специальный процесс TSP – транслятор хранимых процедур, который транслирует исходный текст процедуры в промежуточный байт-код, пригодный для эффективного исполнения.

Для поддержки хранимых процедур в БД должны быть системные таблицы `$$$PROC` и `$$$PRCD`, которые создаются при выполнении SQL-скрипта `systab.sql`, расположенного в подкаталоге `\dict` установочного каталога СУБД ЛИНТЕР.

Примеры создания процедур можно найти в каталоге `samples/sp` дистрибутива СУБД ЛИНТЕР.

Глобальные переменные хранимых процедур

Глобальная переменная хранимых процедур – это переменная СУБД ЛИНТЕР, которая доступна для использования всем хранимым процедурам СУБД ЛИНТЕР. Глобальная переменная может иметь любой допустимый тип данных СУБД ЛИНТЕР (кроме `EXTFILE`). Описание и значение глобальных переменных хранятся в системной таблице `$$$GLBVARS` БД ЛИНТЕР (создается скриптом `systab.sql`). Определение глобальной переменной и присвоение ей первоначального значения (по умолчанию) выполняется с помощью специального SQL-оператора. После этого она становится доступной всем хранимым процедурам СУБД ЛИНТЕР.

Процедурный язык СУБД ЛИНТЕР позволяет хранимым процедурам во время выполнения присваивать глобальным переменным значения, которые будут доступны всем хранимым процедурам, выполняемым в текущей сессии конкретного пользователя. После завершения сессии, измененные в ходе сессии значения глобальных переменных в БД ЛИНТЕР не сохраняются.

В процессе выполнения хранимых процедур значения глобальных переменных хранятся в локальной для каждой пользовательской сессии области глобальных переменных, то есть одна и та же глобальная переменная в каждой пользовательской сессии будет иметь свою копию.

Триггеры

Триггером в СУБД ЛИНТЕР называется хранимая процедура, которая вызывается на выполнение ядром СУБД автоматически при наступлении в БД события, на которое настроен триггер.

События, на которое может быть настроен триггер, делятся на события обработки данных и системные события. В качестве события на обработку данных для триггера может выступать модификация (`INSERT`, `DELETE` или `UPDATE`) некоторой таблицы. В качестве системного события может быть событие регистрации пользователя. Триггер может вызываться один раз на весь SQL-оператор или на каждую операцию, выполняемую с записью таблицы. Триггеры различаются по времени срабатывания: *до* выполнения операции с записью (`BEFORE`), *после* выполнения операции (`AFTER`) и *вместо* выполнения операции (`INSTEAD OF`).

Тело триггера представляет собой исходный текст на процедурном языке, то есть оно может включать декларации, исполняемые операторы и блок обработки исключений.

 © Архитектура СУБД. АО НПП «РЕЛЭКС», 1990-2026

В теле триггера разрешены любые операторы, в том числе, исполнение SQL-запросов, вызов хранимых процедур и т.д.

Триггеры типа INSTEAD OF применимы только к базовым таблицам и не применимы к представлениям.

В теле триггера, вызываемого на каждую строку (FOR EACH ROW) доступны специальные переменные с префиксом по умолчанию OLD и NEW, содержащие старое и новое значение строки. Обе эти переменные – структуры (то есть с точки зрения языка хранимых процедур – курсорные переменные), включающие столбцы тех же имен и типов, что и обрабатываемая триггером таблица.

В зависимости от вида триггера эти переменные определяются следующим образом:

- в триггере на INSERT доступна только переменная NEW; она содержит все значения, которые должны быть занесены в новую строку (если триггер BEFORE), или которые уже занесены (если триггер AFTER); при этом в триггере BEFORE допустимо присвоить полям переменной NEW новые значения, переопределив, таким образом, исходное поведение INSERT;
- в триггере на UPDATE переменная NEW содержит значение, которое должно содержаться в строке после обновления (если триггер BEFORE), или которое уже там содержится (если триггер AFTER); при этом в триггере UPDATE допустимо присвоить полям переменной NEW новые значения, переопределив, таким образом, исходное поведение UPDATE; переменная OLD всегда содержит все значения, которые были в строке до обновления;
- в триггере на DELETE доступна только переменная OLD; она содержит все значения удаляемой строки.

Кроме изменения значений переменной NEW, триггер BEFORE … FOR EACH ROW также может повлиять на ход выполнения SQL-запроса путем запрета операции. Для этого триггер должен вернуть логическое значение FALSE или завершиться с исключением. В такой ситуации конкретная строка, обработанная триггером, пропускается (не вставляется, не обновляется или не удаляется). Все остальные строки продолжают обрабатываться. Таким образом, в результате действия триггеров может оказаться, что часть строк, которые должны быть обработаны SQL-запросом, из этой обработки будут исключены. В этом случае количество реально обработанных строк можно получить с помощью соответствующих вызовов СУБД ЛИНТЕР.

СУБД ЛИНТЕР позволяет завершить и откатить весь SQL-оператор из триггера, вернув пользователю заданный код завершения.

С помощью триггеров можно снабдить СУБД такими возможностями, как альтернативный аудит, основанный на значениях данных, комплексный контроль безопасности и сложные правила целостности. Например, можно создать триггер, который будет позволять модифицировать некоторую таблицу лишь в заданный период времени.



Примечание

Хотя триггеры БД позволяют определять и вводить в действие правила целостности, они не эквивалентны этим правилам, а дополняют и/или расширяют их. Например, триггер, определенный для введения в действие правила целостности, не проверяет данные, уже загруженные в таблицу. Поэтому рекомендуется использовать триггеры вместо ограничений целостности только тогда, когда ограничение целостности не может быть реализовано подходящим правилом целостности.

Для поддержки триггеров должны существовать системные таблицы $$$TRIG и $$$PROC (создаются при выполнении SQL-скрипта systab.sql).

Средства интернационализации

СУБД ЛИНТЕР позволяет хранить символьную информацию в кодировке, задаваемой пользователем БД.

Это позволяет пользователю отправлять СУБД и получать от нее символьные данные в заданной кодировке (без преобразования), а также в других кодировках, для которых СУБД может автоматически выполнить (при необходимости) преобразование пользовательских данных в заданную кодировку.

По умолчанию в СУБД ЛИНТЕР включена поддержка следующих кодовых страниц:

- кириллические однобайтовые: CP866, KOI8-R, CP1251;
- европейские однобайтовые CP437, CP850, CP1252, и с ISO 8859-1 по ISO 8859-16;
- многобайтовые: CP932 (Japanese Shift-JIS), CP936 (Simplified Chinese GBK), CP949 (Korean), CP950 (Traditional Chinese Big5), EUC Japanese;
- UTF-8.

Кроме перечисленных кодовых страниц, в СУБД можно загружать любую необходимую пользователю кодовую страницу.

СУБД ЛИНТЕР для хранения информации о кодовых страницах создает в БД служебные таблицы $$$CHARSET и $$$TRANSL, входящие в системный словарь БД.

Таблица $$$CHARSET используется для хранения кодовых страниц. Она содержит информацию о коде страницы, о свойствах страницы, о «весах» символов кодовой страницы и т.п. Таблица трансляции ($$$TRANSL) хранит информацию о взаимном преобразовании однобайтовых кодовых страниц. Системные таблицы $$$CHARSET и $$$TRANSL создаются при выполнении SQL-скрипта cstables.sql.

СУБД ЛИНТЕР позволяет назначить кодировку:

- всей БД, то есть данные во всех пользовательских таблицах, для которых их кодировка и кодировка их столбцов не задана явно, будут храниться в указанной кодировке;
- системным таблицам, то есть данные во всех системных таблицах будут храниться в указанной кодировке;
- отдельной таблице БД, то есть данные всех столбцов конкретной таблицы будут храниться в одной заданной кодировке;
- столбцам таблицы, то есть данные в разных столбцах конкретной таблицы могут храниться в различных кодировках;
- каналу, по которому обмениваются данными СУБД ЛИНТЕР и клиентское приложение. Клиентское приложение будет получать запрашиваемые из БД данные в той кодировке, которая установлена для канала, по которому послан запрос. Если кодировка канала явно не задана, то данные будут представлены в кодировке по умолчанию, установленной в операционной системе (для Windows соответствующей текущему значению locale).

Установка требуемых кодировок выполняется с помощью соответствующих SQL-запросов.

Перекодировкой данных занимается ядро СУБД ЛИНТЕР. Для исключения дополнительной работы при перекодировке данных используется таблица трансляции.

 © Архитектура СУБД. АО НПП «РЕЛЭКС», 1990-2026

В случае отсутствия ее в СУБД она генерируется динамически – через таблицу UNICODE-символов.

Многобайтовые кодовые страницы хранятся в виде своих составляющих – набора однобайтовых кодовых страниц и при загрузке в СУБД «собираются» в единое целое с целью получения общей таблицы трансляции в/из UNICODE.

СУБД ЛИНТЕР может использовать разные кодовые страницы для системных словарей и пользовательских таблиц.

В только что созданной БД используется интегрированная кодовая страница 20127 (US-ASCII), имеющая имя «DEFAULT» (кодовая страница по умолчанию для системного словаря) и содержащая 127 символов. В символах этой кодовой страницы хранятся метаданные БД: названия таблиц, столбцов, триггеров, процедур и т.д. При необходимости может быть установлена другая однобайтовая кодовая страница по умолчанию для системного словаря.

Аналогично в БД может быть задана кодовая страница по умолчанию для хранения данных пользовательских таблиц. Если она явно не установлена, используется кодовая страница по умолчанию для системного словаря.

Для нормальной работы с кодовой страницей рекомендуется явно задать ее на клиентской части. Проще всего это можно сделать с помощью переменной окружения LINTER_CP.

Данные в столбцах в произвольной национальной кодовой странице могут храниться двумя способами:

1) в текстовых столбцах таблиц с заданной национальной кодовой страницей;
2) в UNICODE-столбцах таблиц.

В качестве длин текстовых полей указывается их размер в символах.

Поскольку хранение многобайтовых кодовых страниц требует существенного объема оперативной памяти, то внутри СУБД такие страницы хранятся в отдельной очереди, которая по умолчанию не инициализирована. Ее инициализация происходит только в результате занесения в таблицу $$$CHARSET первой многобайтовой кодовой страницы и последующего перезапуска СУБД ЛИНТЕР.

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 для 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 можно будет использовать вместо KOI8-R алиас «Кириллица».

Кэшируемые SQL-запросы

Для исключения повторной трансляции и повторного выполнения одних и тех же 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:

- кэш запросов: хранит ссылки на исходный текст кэшируемого SQL-запроса и его оттранслированную версию;
- кэш результатов выполнения запросов: ссылки на исходный текст кэшируемого SQL-запроса и результат его выполнения.

Тексты запросов и результаты их выполнения хранятся в рабочих файлах СУБД.

При поступлении на обработку очередного кэшируемого SQL-запроса ядро СУБД ищет текст этого запроса в кэше запросов. Это означает следующее:

- если приложение каждый раз посылает ядру СУБД один и тот же текст кэшируемого SQL-запроса, то он будет найден в кэше результатов выполнения запросов;
- если приложение посылает ядру СУБД один и тот же оттранслированный SQL-запрос, то результаты такого запроса в кэше результатов найдены не будут и данный запрос будет каждый раз выполняться заново.

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).

При изменении данных, влияющих на результат выполнения кэшированного запроса (обновление задействованных в запросе таблиц, удаление используемых синонимов, отмена прав доступа пользователя и т.п.), текущий результат выполнения запроса удаляется из кэша результатов. При следующей попытке получить результат выполнения кэшируемого запроса этот запрос заново обрабатывается ядром СУБД и его результат повторно заносится в кэш результатов.

СУБД ЛИНТЕР поддерживает кэширование только DML-запросов (`SELECT`, `INSERT`, `UPDATE`, `DELETE`).

Если задан режим кэширования текстов запросов, то по умолчанию все SELECT-запросы кэшируются.

Использование механизма кэширования может существенно увеличить производительность многих информационно-справочных систем и снизить нагрузку на СУБД. Например, в справочной системе аэропорта значительную часть запросов составляют запросы о расписании полетов в определенные города, длительности полета, цене билетов. В этом случае нет необходимости каждый раз транслировать и выполнять соответствующие запросы – можно сразу брать готовые ответы из кэша результатов, тем более, что обновление таких данных происходит не часто.

 © Архитектура СУБД. АО НПП «РЕЛЭКС», 1990-2026

Предварительная загрузка таблиц

СУБД ЛИНТЕР поддерживает механизм предварительной загрузки пользовательских таблиц.

Под предварительной загрузкой таблиц понимается процесс загрузки с диска в оперативную память компьютера страниц файлов данных таблицы и файлов индексов и, по желанию, страниц BLOB-файлов. Загрузка внешних файлов не поддерживается.

Предварительную загрузку наиболее эффективно применять на 64-битных аппаратных платформах, позволяющих предоставлять СУБД большие объемы оперативной памяти.

Предварительно выполненная загрузка таблиц позволяет существенно увеличить скорость обработки транзакций и выполнения поисковых запросов, использующих загруженные в оперативную память таблицы, за счет исключения операций ввода/вывода данных с диска.

Предварительная загрузка таблиц, необходимых для работы клиентских приложений, выполняется, как правило, после старта ядра СУБД и загрузке подлежат таблицы, содержащие наиболее важные и часто используемые данные.

Количество реально загружаемых в оперативную память страниц файлов таблицы зависит от выделенного ядру СУБД объема оперативной памяти (параметр /POOL команды запуска ядра СУБД ЛИНТЕР) и от размера загружаемых таблиц. Если выделенной ядру СУБД оперативной памяти недостаточно для загрузки нужных таблиц, то можно перезапустить ядро с увеличенным размером пула. В противном случае загруженные страницы будут вытесняться по мере необходимости страницами других таблиц БД.

Словарь данных

Каждая БД ЛИНТЕР имеет *словарь данных*, в котором хранится информация о логической и физической структуре БД (метаданные).

Таблицы словаря данных подробно описаны в документе «Системные таблицы и представления».

Главный словарь данных

Главный словарь данных СУБД ЛИНТЕР состоит из трех таблиц:

1) `$$$SYSRL` – таблица таблиц. Содержит информацию:
   - обо всех таблицах БД (как пользовательских, так и системных): имя таблицы, идентификатор владельца, число столбцов, размеры файлов и другая информация;
   - о представлениях;
   - о синонимах таблиц и представлений.
2) `$$$ATTRI` – таблица столбцов. Содержит информацию о столбцах всех таблиц БД: имя столбца, принадлежность к таблице, тип данных, информация об индексах (включая составные и именованные), ограничения целостности и другая информация;
3) `$$$USR` – таблица пользователей БД и их полномочий. Содержит информацию о зарегистрированных пользователях БД, созданных схемах, ролях, о назначении ролей пользователям БД и о назначении прав доступа пользователей и ролей к объектам БД.

Объекты главного словаря данных создаются автоматически при генерации БД. Без главного словаря СУБД ЛИНТЕР работать не сможет, так как на нем основывается регистрация, верификация и управление всей работой СУБД. Например, во время операций с БД ядро СУБД обращается к словарю данных для проверки того, что объекты БД существуют и что пользователи имеют к ним соответствующие права доступа.

При изменении структуры БД главный словарь данных обновляется автоматически.

Дополнительные словари данных

Кроме главного словаря данных, БД может содержать еще и дополнительные словари, которые добавляются в БД администратором БД исходя из функциональных требований клиентских приложений. Отсутствие объектов дополнительного словаря не критично для работоспособности СУБД и поэтому они не добавляются в БД при её первоначальном создании.

Создать объекты дополнительного словаря можно при помощи стандартных SQL-скриптов, входящих в дистрибутив СУБД ЛИНТЕР (расположены в подкаталоге `\dict` установочного каталога СУБД ЛИНТЕР).

SQL-скрипты могут быть выполнены утилитой `inl` или утилитой «`Рабочий стол СУБД ЛИНТЕР`».

Ниже приведен потенциально возможный состав объектов дополнительного словаря СУБД ЛИНТЕР.

Скрипт `systab.sql` создаёт следующие таблицы:

- `$$$GLBVARS` – словарь глобальных переменных хранимых процедур. Содержит информацию об именах, типах данных и текущих значениях глобальных переменных хранимых процедур. Если эта таблица не создана, то работа с глобальными переменными не поддерживается;
- `$$$PRCD` и `$$$PROC` – словари хранимых процедур. Содержат информацию об именах, типах параметров и о типе возвращаемого значения для каждой хранимой процедуры. Если эти таблицы не созданы, то хранимые процедуры в БД работать не будут;
- `$$$TRIG` – словарь триггеров. Содержит имена таблиц, для которых созданы триггеры, идентификатор владельца триггера, информацию о параметрах триггера. СУБД ЛИНТЕР поддерживает работу с триггерами, только если таблица `$$$TRIG` существует в БД;
- `$$$SEQ` – словарь последовательностей. Содержит информацию о создателе последовательности, начальном, максимальном и текущем значении последовательности, а также шаг последовательности. Без этой таблицы работа с последовательностями невозможна.

Для работы с расширенной подсистемой защиты данных (в частности, дискретный и мандатный контроль доступа к данным), которая включает в себя создание групп и уровней пользователей, контроль доступа с сетевых станций и ввода-вывода на внешние устройства, а также средств аудита в БД, необходимо создать специальные служебные таблицы.

Скрипты `security.sql`, `extsec.sql` и `enaacc.sql` создают следующие таблицы:

- `$$$LEVEL` – таблица уровней доступа. Уровни доступа вводятся для проверки на уровне ядра СУБД ЛИНТЕР прав на чтение/запись информации по мандатному принципу контроля доступа. Таблица содержит номер и имя групп доступа;
- `$$$GROUP` – таблица групп пользователей. Содержит информацию о пользователях, включенных в группу; о правах, назначенных непосредственно группе, и переданных ей от других групп;
- `$$$STATION` – содержит сведения о сетевых станциях (имя, протокол доступа, сетевой адрес, допустимый уровень доступа);
- `$$$RELATION` – содержит описание отношений между сетевыми станциями и пользователями БД;
- `$$$DEVICE` – таблица внешних устройств. Содержит описание (имя, путь, допустимый уровень доступа для чтения и для записи) всех внешних устройств, на которых располагаются файлы БД. Без этой таблицы также невозможно запустить внутреннее резервное сохранение БД средствами ядра СУБД ЛИНТЕР;
- `$$$AUDIT` – протокол работы с БД. Если включено протоколирование, то в эту таблицу будут занесены все действия клиентских приложений, связанные с идентификацией и аутентификацией, запросами на доступ к объектам защиты, созданием/уничтожением объектов защиты, с изменением прав доступа и т.п. То есть регистрации подлежат время и тип события, имя инициировавшего событие субъекта, сетевой адрес клиентской станции, статус завершения операции и т.д.

  Таблица `$$AUDIT` имеет довольно сложную структуру и для удобства просмотра протокола аудита имеется стандартное представление `AUDIT_EVENTS`.

Скрипты `distr.sql`, `arepl.sql` создают таблицы, необходимые для распределенной обработки данных и репликации:

- `SERVERS` – содержит имена удаленных серверов, на которые могут реплицироваться данные;
- `$$$REPL` – содержит описание правил репликации таблиц;
- `$$$EXTREPL` – содержит описание правил репликации столбцов.

Для хранения информации о кодовых страницах на физическом уровне необходимо создать системные словари для работы с кодировками.

Скрипты `cstables.sql`, `charsets.sql`, `mbcps.sql` и `descmb.sql` создают необходимые таблицы и заполняют их наиболее распространенными кодовыми страницами:

- `$$$CHARSET` – таблица кодовых страниц. Содержит информацию о коде страницы, имя и информацию о свойствах страницы, о «весе» символов кодовой страницы, о верхнем/нижнем регистре и о преобразовании в UNICODE. Эта таблица позволяет хранить как информацию об однобайтовых кодовых страницах, так и о многобайтовых;
- `$$$TRANSL` – таблица трансляций. Содержит информацию о трансляции однобайтовых кодовых страниц друг в друга: порядковый номер трансляции, номер кодовой страницы, из которой осуществляется трансляция, номер кодовой страницы, в которую осуществляется трансляция, имя трансляции, и собственно саму таблицу трансляции (информация о соответствии символов).

Для работы с СУБД через ODBC-драйвер необходимо исполнить SQL-скрипт `catalog.sql`. Он содержит таблицы ODBC-каталога. Без исполнения этого файла ODBC-приложения не смогут работать с БД. Перед исполнением `catalog.sql` необходимо выполнить `systab.sql` и `security.sql`, иначе не все возможности СУБД будут доступны через ODBC-драйвер.

Для поддержки .NET-интерфейса необходимо выполнить SQL-скрипты `catalog.sql` и `extsec.sql`.

Для работы подсистемы фразового поиска необходимо создать системные словари при помощи скриптов `search.sql` и `default.sql`.

Для поддержки оперативного резервирования данных необходимо создать системную таблицу при помощи скрипта `inkernel.sql`.

Для создания таблицы диагностических сообщений в соответствии с выбранным языком интерфейса СУБД ЛИНТЕР необходимо выполнить SQL-скрипт `cerrors.sql`. Тексты сообщений должны быть загружены из файла `errors.lod`.

 © Архитектура СУБД. АО НПП «РЕЛЭКС», 1990-2026

Файловая структура БД

На физическом уровне базовая таблица БД представлена в виде набора файлов операционной системы:

- файлы, которые содержат её данные (файлы данных);
- файлы со структурами, обеспечивающими ускоренный доступ к данным (файлы индексов);
- файлы, содержащие неструктурированную информацию большого объёма (BLOB-файлы);
- файлы фразовых индексов.

Два последних типа файлов создаются только в том случае, если таблица содержит столбцы соответствующего типа.

Именование перечисленных типов файлов в БД (кроме EXT-файлов, которые имеют пользовательские файловые имена) приведено на рисунке 6.

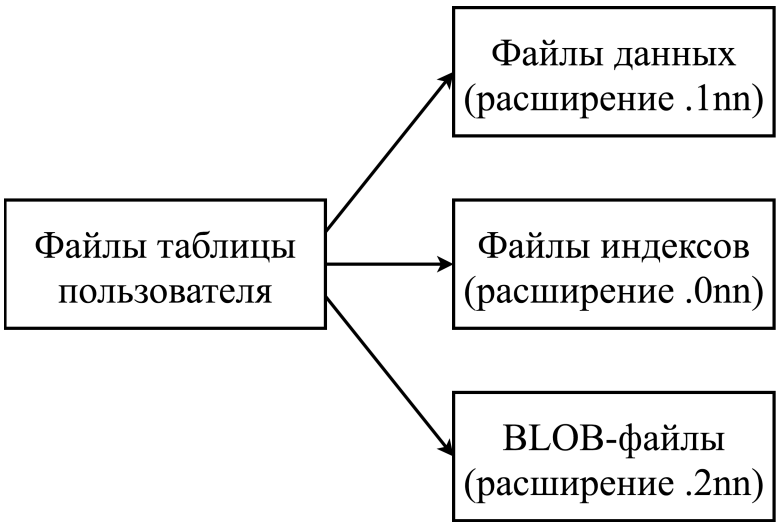


Рисунок 6. Физическая структура БД ЛИНТЕР

Максимальное количество создаваемых файлов каждого типа не более 63.

Имена файлов таблицы имеют следующий формат:

```
<системный номер таблицы>.<тип файла><номер файла>
```

где:

1) `<системный номер таблицы>` – системный номер таблицы в БД (значение столбца `$$$S11` данной таблицы в системной таблице `$$$SYSRL`);
2) `<тип файла>` – тип файла таблицы:
   - 0 – IDX-файл (файл индексов);
   - 1 – DATA-файл (файл данных);
   - 2 – BLOB-файл (файл BLOB-данных).

3) <номер файла> – порядковый номер файла данного типа.

Например, 116.01, 116.11.

Файлы таблицы могут располагаться на различных физических или логических дисках ОС. Они создаются/изменяются/удаляются ядром СУБД ЛИНТЕР при обработке SQL-запросов создания/модификации/удаления таблицы.

В запросах создания/модификации схемы таблицы можно указать количество и размеры файлов данных, индексов, BLOB-файлов, а также назначить физическое устройство, на котором следует создать тот или иной файл.

Выбор этих характеристик зависит от размеров таблицы, наличия свободного пространства на устройствах прямого доступа и количества этих устройств в конфигурации вычислительного комплекса.

Все файлы СУБД ЛИНТЕР (кроме файлов системного журнала) имеют страничную организацию.

Страница файла – это единица ввода/вывода для ядра СУБД ЛИНТЕР, а также единица захвата, освобождения и расширения файлового пространства. Кроме того, страница является единицей непрерывного расположения данных (запись любой таблицы не может быть расположена на двух и более страницах).

Длина страниц файлов БД ЛИНТЕР равна 4096 байт.

Примечание
СУБД ЛИНТЕР рассчитывает на атомарность записи страницы размером 4096 байт (СУБД ЛИНТЕР предполагает, что каждая такая страница при записи будет либо заменена полностью, либо останется в прежнем состоянии). В случае, если атомарность записи страницы не обеспечивается (если часть страницы на диске может быть записана содержимым из внутреннего пула страниц, а часть - остаться без изменений), корректная работа СУБД ЛИНТЕР не гарантируется.

Первые страницы каждого файла таблицы занимает битовая карта (BitMap) состояния страниц этого файла. Каждый бит битовой карты отражает:

- для IDX-файлов – факт занятости соответствующей страницы файла (занята или свободна);
- для DATA/BLOB-файлов – возможность использования страницы (наличие или / отсутствие свободного места в странице для добавления записи).

Например, первый индексный файл таблицы может иметь следующую структуру:

```
[B=1][C=2…32768][B=32769][C=32770…65536][B=65537][C=65538…65553]
[I=65554…98304][B=98305][I=98306…131072]...
```

где цифры соответствуют порядковым номерам страниц файла, а буквы обозначают:

- B – страницу битовой карты;
- C – страницу конвертера;
- I – страницу индекса.

Файлы данных

Страницы файла данных содержат упакованные записи таблицы.

Запись в таблице идентифицируется системным номером записи (ROWID). Место удаленной записи, а также ее системный номер повторно используются СУБД. Записи хранятся в неупорядоченном виде.

Правила упаковки данных

1) Упаковка не распространяется на BLOB-данные и внешние файлы (EXTFILE).
2) В символьных значениях усекаются последние пробелы, а при упаковке значений прочих типов данных отбрасываются последние нулевые байты. Например, целые числа со значениями от 0 до 255 будут занимать только один байт, целые числа от 256 до 32767 – 2 байта и т.д.
3) Если значение столбца нулевое (или состоит из одних пробелов), то в упакованной записи на соответствующем месте будет только первый байт или символ. Остальные байты (символы) значения в упакованной записи отсутствуют. При выборке таких значений из БД они преобразуются в первоначальный вид.
4) Для оптимизации работы ядра СУБД при вставке данных рекомендуется установить процент ожидаемого размера упакованной записи по сравнению с неупакованной (параметр PCTFILL, указывается при создании таблицы, по умолчанию равен 100%).

Правила размещения данных

1) При размещении данных рекомендуется установить процентный порог свободного места страниц файла данных, при котором страница будет помечена как занятая (параметр PCTFREE, указывается при создании таблицы, значение по умолчанию равно 0%). Параметр PCTFREE предназначен для предотвращения фрагментации в страницах данных.
2) Для размещения в таблице вновь поступившей записи определяется (через битовую карту состояния страниц файла данных) первая страница (с учетом параметра PCTFREE), в которую можно разместить упакованную запись (с учетом параметра PCTFILL), либо добавляется новая страница.
3) Страница помечается, как занятая до тех пор, пока не будет иметь более чем PCTFREE процентов свободного пространства. Таким образом, высокий процент заполнения страниц файла данных (параметр PCTFILL) минимизирует файловое пространство, занятое данными, ускоряя поиск данных, но замедляет добавление и модификацию данных. Низкий процент заполнения страниц файла данных приводит к неэффективному размещению данных, но может ускорить загрузку и модификацию данных.

Файлы индексов

Для каждой таблицы создается, по меньшей мере, один файл индексов – IDX-файл. Максимальное возможное количество файлов индексов не более 63. Каждый файл индексов содержит битовую карту состояния страниц файла (BitMap) и собственно индексные страницы. Дополнительно к этому первый индексный файл (с расширением .01) содержит адресный конвертер записей.

Бит карты BitMap IDX-файла указывает на присутствие/отсутствие в соответствующей странице информации (страница занята/свободна).

Индексные страницы содержат информацию о простых и составных индексах таблицы. Индексы предназначены для повышения скорости поисковых операций в БД.

Простой индекс по столбцу A строится из пар вида:

```
Значение <-> RowId
```

Составной индекс по столбцам строится из пар вида:

```
Значение1+Значение2+ … +ЗначениеN <-> RowId
```

Структура первого файла индексов приведена на рисунке 7.

При использовании в SQL-запросе условий поиска, наложенных на индексируемые столбцы, ядро СУБД автоматически подключает аппарат индексов.

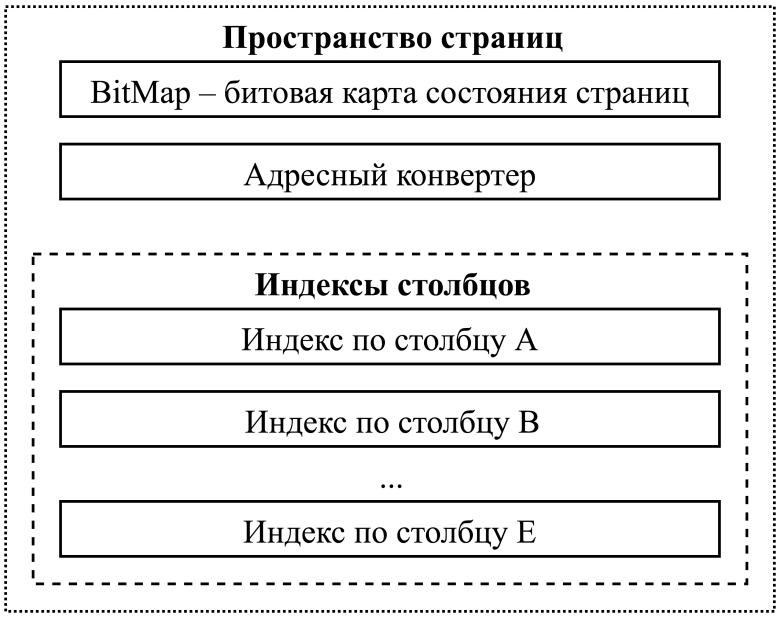


Рисунок 7. Структура первого файла индексов

Структура индексов представляет собой B*-дерево, каждый узел которого является страницей IDX-файла. При добавлении информации в индексированную таблицу дерево индексов разветвляется (занимая свободные страницы файла индексов), растет уровень дерева, то есть количество узлов, которое необходимо пройти по дереву для определения местоположения искомых данных (соответственно, операций чтения данных страниц файла индексов). При удалении данных из таблицы индексные структуры упрощаются, освобождая занятые страницы.

Индексы ссылаются на файл данных только косвенно, через адресный конвертер – специальную структуру данных, которая по номеру записи позволяет определить её местонахождение в файле данных. Каждой записи с номером N в адресном конвертере соответствует указатель на её местонахождение в виде пары чисел:

```
<номер файла данных><номер страницы в этом файле>
```

Следовательно, размер адресного конвертера пропорционален максимальному количеству `ROWID`, указанному при создании таблицы. На рисунке 8 приведена схема взаимодействия адресного конвертера и индексов.

 © Архитектура СУБД. АО НПП «РЕЛЭКС», 1990-2026

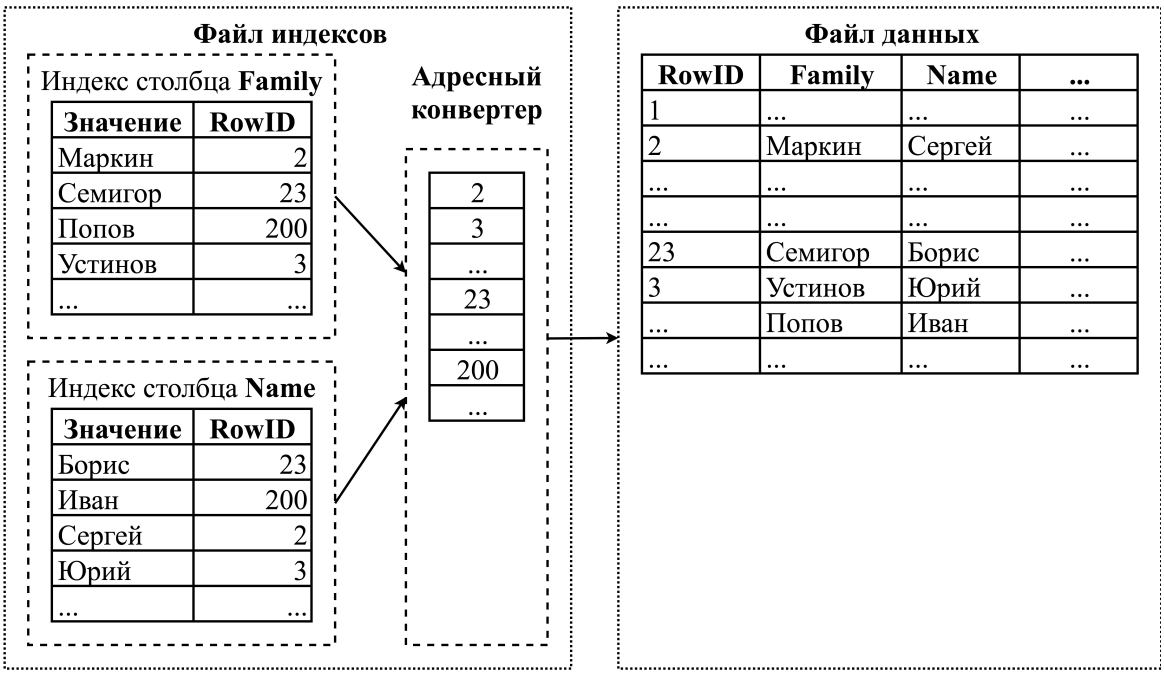


Рисунок 8. Схема взаимодействия файла индексов и адресного конвертера

Длина индекса в СУБД ЛИНТЕР не может превышать 1024 байта.

Файлы неструктурированных объектов

BLOB (Binary Large Object) – неструктурированный двоичный объект, предназначенный для хранения текстовой информации большого объема, графики, мультимедийных данных и т.п.

Значения типа BLOB располагаются в BLOB-файле таблицы, а в каждой записи файла данных на месте соответствующего BLOB-поля хранится байтовая строка, которая является ссылкой на страницу BLOB-файла, где хранится BLOB-значение (рисунок 9).

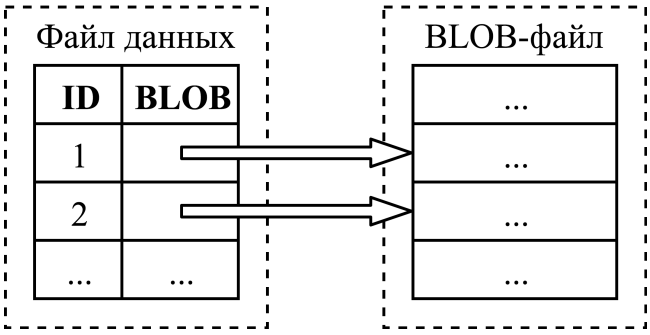


Рисунок 9. Структура ссылок на BLOB-объекты

Два небольших BLOB-значения могут соседствовать в одной странице, одно значение должно ограничиваться пределами одного BLOB-файла.

Примечание
Допускается создание нескольких BLOB-столбцов в таблице (их максимальное количество зависит от размеров записи таблицы). Для работы с BLOB-значениями существует два типа команд. Команды первого типа не рассчитаны на использование

нескольких BLOB-столбцов, и поэтому в них не указывается номер BLOB-столбца. Если использовать эти команды, то они всегда будут работать с первым BLOB-столбцом таблицы. В командах второго типа указывается номер BLOB-столбца. Нумерация столбцов начинается с 1, и при подсчете номера столбца учитываются столбцы всех типов.

При создании таблицы имеется возможность указать процент заполнения страниц BLOB-файла. Это число должно быть целым в пределах от 1 до 100. Если какая-либо страница заполнена свыше этого процента, то она будет использоваться только для расширения хранящихся в ней BLOB-значений, но не для добавления новых.

По умолчанию процент заполнения страниц BLOB-файла равен 50. Если заносимые в таблицу BLOB-значения в дальнейшем не расширяются, то рекомендуется увеличить процент заполнения.

С точки зрения СУБД характер информации, хранящейся в BLOB-столбце, значения не имеет, т.к. ядро СУБД выполняет с этими объектами только примитивные операции:

- выбрать (и передать) порцию данных клиентскому приложению;
- очистить (освободить) занятые BLOB-объектом страницы;
- удалить BLOB-данные (одновременно с удалением соответствующего поля в записи таблицы);
- добавить порцию данных в конец BLOB-объекта.

Выборка BLOB-данных и добавление их в BLOB-столбец выполняется порциями. Размер порции при этом не должен превышать 64768 байт.

Ядро СУБД интерпретирует BLOB-значение как последовательность байтов, однако для приложений, использующих такие значения, важно отличать графику от текста, аудио данные от анимации и т.д. Для этого пользователь может кодировать характер размещаемой в БД BLOB-информации. Для идентификации характера BLOB-информации используются коды. Значения кодов в СУБД не типизированы (пользователь по своему усмотрению выполняет идентификацию BLOB-данных). В СУБД ЛИНТЕР максимальное количество кодов равно 232, при этом значения меньше нуля зарезервированы разработчиком СУБД ЛИНТЕР для дальнейшего использования.

Внешние файлы

СУБД ЛИНТЕР поддерживает тип данных `EXTFILE` (внешний файл). Доступ к данным типа `EXTFILE`, размещенным в удаленных БД, осуществляется из клиентского приложения с помощью сетевых функций ОС или с использованием встроенной функции языка SQL.

По столбцу `EXTFILE` может быть построен фразовый индекс, позволяющий производить поиск информации по полному тексту или по всем текстовым полям файла.

Характер информации, находящейся во внешнем файле, не имеет значения. Ядро СУБД поддерживает с этим объектом следующие операции:

- получить имя файла, соответствующего типу данных `EXTFILE`;
- получить дату и время последнего изменения внешнего файла;
- получить размер внешнего файла;
- получить имя по умолчанию внешнего файла;

 © Архитектура СУБД. АО НПП «РЕЛЭКС», 1990-2026

- установить фильтр для столбца EXTFILE;
- получить номер фильтра, установленного для столбца с типом EXTFILE.

Рабочие файлы

В процессе работы СУБД ЛИНТЕР создает следующие рабочие файлы:

- 1.31 – файл бит-векторов. Используется ядром СУБД для хранения информации о записях, найденных при выполнении поискового запроса;
- 1.41 – рабочий файл для хранения найденных записей поискового запроса (используется также и для хранения временных индексов);
- 1.5* – файлы сортировки. * может изменяться от 1 до 255. Предназначены для выполнения сортировки результата поискового запроса, обработки SQL-запросов с опциями DISTINCT и GROUP BY, построения индексов и т.д. По умолчанию в БД создается один файл сортировки (1.51), но их количество может быть изменено утилитой gendb;
- 1.61 – специальный файл системного журнала, содержащий информацию о контрольных точках, горячем резервировании и т.д.;
- xxxxxxxx.61 (например: 00000001.61, 00000002.61, ...) – файлы системного журнала, предназначенные для ведения протокола о всех изменениях, сделанных в БД (сеансы чтения в журнале не регистрируются). При внезапном (аварийном) завершении работы СУБД (например, при отключении питания компьютера) некоторые транзакции могут быть не обработанными до конца. В этом случае при следующем запуске ядро СУБД определит по журналу наличие прерванных транзакций и аннулирует все сделанные ими изменения.

Файлы 1.31 и 1.41 имеют одинаковую структуру (рисунок 10).

страница битовой маски	32767 страниц данных	страница битовой маски	32767 страниц данных	...

Рисунок 10. Физическая структура файлов 1.31 и 1.41

Размеры рабочих файлов не являются фиксированными на протяжении всего сеанса работы ядра СУБД, а расширяются по мере необходимости. Устанавливаются только их предельные размеры (см. документ «Создание и конфигурирование базы данных», команды SET SYSWRK LIMIT и SET SYSWBV LIMIT).

Страницы данных рабочего файла 1.41 используются для хранения файловых рабочих областей СУБД ЛИНТЕР и их дескрипторов. Рабочие области предназначены для хранения результатов обработки данных различных пользователей, контента триггерных операций и хранимых процедур. Логическая структура страниц данных рабочего файла приведена на рисунке 11.

страница дескрипторов рабочих областей	рабочая область		рабочая область	рабочая область	...	страница дескрипторов рабочих областей	рабочая область	рабочая область	...

Рисунок 11. Логическая структура страниц данных рабочего файла

Страницы дескрипторов выделяются по мере необходимости (в начале работы СУБД ЛИНТЕР их нет вообще) и могут располагаться в разных местах рабочего файла, а номера этих страниц сохраняются в специально выделенном массиве в пуле ядра СУБД. Размер этого массива задается при создании БД (см. документ «Создание и конфигурирование базы данных», команда SET WORKAREA).

© Архитектура СУБД. АО НПП «РЕЛЭКС», 1990-2026

Характеристики СУБД ЛИНТЕР

Простые типы данных

Список поддерживаемых СУБД ЛИНТЕР простых типов данных приведен в таблице 2.

Таблица 2. Поддерживаемые типы данных

Тип данных	Обозначение	Длина (в байтах)
Строка символов фиксированной длины	CHAR	до 4000
Строка UNICODE-символов фиксированной длины	NCHAR	до 4000 (2000 символов)
Строка символов переменной длины	VARCHAR	до 4000
Строка UNICODE-символов переменной длины	NCHAR VARYING	до 4000 (2000 символов)
Строка байтов фиксированной длины	BYTE	до 4000
Строка байтов переменной длины	VARBYTE	до 4000
Короткое целое	SMALLINT	2
Целое	INTEGER	4
Длинное целое	BIGINT	8
Вещественное число (плавающая точка)	REAL	4
Вещественное число двойной точности (плавающая точка)	DOUBLE	8
Вещественное число (фиксированная точка)	NUMERIC	16
Логическое значение	BOOLEAN	1
Дата	DATE	16
BLOB-объект	BLOB	до 2 млрд.
Внешний файл	EXTFILE	до 2 млрд.
Геометрические типы данных	раздел «Геометрические типы данных»	

Геометрические типы данных

В СУБД ЛИНТЕР для поддержки геометрических типов данных реализовано подмножество среды SQL с геометрическими типами (SQL with Geometry Types), спецификация которой предложена консорциумом OpenGIS. Столбец таблицы, в котором хранятся геометрические данные, имеет геометрический тип.

Для хранения геометрических данных можно использовать тип данных `VARBYTE`/`GEOMETRY` с максимальным размером 4000 байт (применяется для хранения простых геометрических объектов), либо `BLOB` (применяется для хранения сложных геометрических объектов).

В СУБД ЛИНТЕР поддерживается возможность создания одностолбцовых индексов для BLOB-столбцов геометрических типов.

Геометрические типы данных позволяют генерировать, сохранять и анализировать географические данные, которые отражают элементы окружающего мира:

- географические объекты (например: гора, водоем, город);
- территории (например: область, задаваемая почтовым индексом);
- местоположения (например, перекресток, как специфическое место пересечения двух дорог).

СУБД ЛИНТЕР поддерживает следующие типы геометрических данных (таблица 3).

Таблица 3. Геометрические типы данных, поддерживаемые СУБД ЛИНТЕР

Тип данных	Описание
POINT	точка
LINESTRING[(n)]	ломаная линия
POLYGON[(n)]	многоугольник
MULTIPOINT[(n)]	набор точек
MULTILINESTRING[(n)]	набор ломаных линий
MULTIPOLYGON[(n)]	набор многоугольников
BOX	прямоугольник
LINE	простая (не ломаная) линия
CIRCLE	окружность
GEOMETRYCOLLECTION[(n)]	набор геометрических объектов

Количественные характеристики

Количественные характеристики БД ЛИНТЕР приведены в таблице 4.

Таблица 4. Количественные характеристики БД ЛИНТЕР

Параметр СУБД ЛИНТЕР	Значение
Максимальное число таблиц в SQL-запросе (на одном уровне)	32
Максимальное число таблиц в БД	65536
Максимальное число записей в таблице	до 2^{30} (~1 млрд.)
Количество записей, выбираемых одним запросом	до 2^{29} (~500 млн.)
Максимальное число столбцов в одной таблице	250
Максимальная длина записи (не считая BLOB-полей)	64 Кбайт[1)]
Максимальное количество ключей в таблице	250
Максимальная длина ключа (в байтах)	1024

[1)]Это не «чистый» размер записи (то есть сумма длин всех полей), а «грязный» – со всей вспомогательной информацией:

- дополнительно 2 байта на каждое поле переменной длины – `VARCHAR`, `NVARCHAR`, `VARBYTE`;
- дополнительные байты на каждый столбец для хранения информации о мандатном доступе;

 © Архитектура СУБД. АО НПП «РЕЛЭКС», 1990-2026

- дополнительный размер заголовка записи.

Вследствие этого реально максимальная длина записи несколько меньше 64 Кбайт. Насколько меньше – зависит от общего количества столбцов в таблице, количества столбцов с переменной длиной и т.д., но достаточно близко к указанному размеру.

Максимальная длина составного индекса равна 1024 байтам.

Ограничение целостности данных

Для БД важно гарантировать подчиненность данных, добавляемых в таблицы БД, некоторым организационным правилам, которые определяются администратором БД или разработчиком клиентского приложения. Если клиентское приложение пытается добавить или изменить запись, нарушающую установленные правила (правила целостности), то СУБД должна отвергнуть операцию и возвратить приложению соответствующий код завершения.

Для решения проблем, связанных с соблюдением правил целостности данных, СУБД ЛИНТЕР предлагает такие средства, как ограничения целостности и триггеры БД.

Ограничение целостности – это декларативный способ задания организационного правила для столбца таблицы. Речь идёт о логической целостности, например, значения столбца «Дата увольнения» не могут быть меньше значений столбца «Дата приема на работу». Ограничение целостности представляет собой утверждение о данных таблицы, которое должно быть всегда истинным:

- если для существующей в БД таблицы вводится ограничение целостности, а в таблице уже существуют данные, не удовлетворяющие этому ограничению, то такое ограничение нельзя ввести в действие;
- после того как ограничение целостности определено, все SQL-запросы, нарушающие это ограничение, игнорируются с выдачей соответствующего кода завершения.

Ограничения целостности определяются вместе с таблицей (или добавляются при модификации её схемы) и сохраняются как часть определения схемы таблицы централизованно в словаре данных. Если правило ограничения целостности изменяется, его требуется изменить лишь один раз – на уровне БД, а не индивидуально для каждого клиентского приложения.

СУБД ЛИНТЕР поддерживает следующие ограничения целостности:

- `NOT NULL` – запрещает внести в столбец таблицы неизвестные или неопределённые значения (обычно NULL-значения используются для обозначения именно неизвестных или неопределённых значений);
- `UNIQUE` (ограничение уникальности значений) – запрещает дублирование значений в столбце. СУБД должна автоматически отслеживать уникальность значений столбца или составного значения группы столбцов (составное значение – это конкатенации значений заданной группы столбцов). При нарушении уникальности приложение получит соответствующий код завершения, а все изменения в БД, сделанные данным SQL-запросом, будут отменены. UNIQUE-ограничение допускает NULL-значения;
- `PRIMARY KEY` (первичный ключ) – запрещает дубликаты и NULL-значения в столбце или составном значении группы столбцов;
- `FOREIGN KEY` (внешний ключ) – требует, чтобы каждое значение в столбце или составном значении группы столбцов в данной таблице совпадало со значением из столбца типа `UNIQUE` или `PRIMARY KEY` в указанной таблице. (Ограничения

целостности FOREIGN KEY также определяют действия «ссылочной целостности», которые определяют действия СУБД ЛИНТЕР с зависимыми данными при изменении тех данных, на которые они ссылаются);

- CHECK (декларативная целостность) – запрещает значения, которые не удовлетворяют логическому выражению ограничения. Это ограничение обычно накладывается на значения столбцов. Например, если корпоративными правилами не разрешено платить сотруднику менее 1000 руб. и более 10000 руб., именно такое ограничение на значения столбца можно ввести, чтобы исключить проверку допустимого диапазона вводимых значений непосредственно в коде клиентского приложения и возложить эту функцию на СУБД.

Другие, более сложные виды логической целостности, можно возложить на хранимые процедуры и триггеры.

Ключи

Ключ – это столбец или группа столбцов, участвующих в определении некоторых типов ограничений целостности. Ключи описывают отношения между различными таблицами и столбцами в реляционной БД.

Существуют следующие типы ключей:

- *первичный* – столбец или группа столбцов, включенных в ключ определения ограничения таблицы PRIMARY KEY. Значения первичного ключа уникально идентифицируют строки в таблице. Для таблицы может быть определен только один первичный ключ;
- *уникальный* – столбец или группа столбцов, включенных в ключ определения ограничения UNIQUE;
- *внешний* – столбец или группа столбцов, включенных в определения ограничения ссылочной целостности таблицы FOREIGN KEY.

Индивидуальные значения столбца, определенного как ключ, называются значениями ключа.

Термины «ключ» и «индекс» часто смешиваются, однако не следует путать их друг с другом. Индексы – это структуры данных, управление которыми выполняется с помощью пользовательских SQL-запросов и которые хранятся в индексных файлах таблиц БД. Индекс создается, чтобы обеспечить быстрый путь доступа к данным таблицы. Ключи – это логическое понятие.

Способы представления геометрических данных

Представление (описание) геометрических типов данных возможно двумя способами:

1) в текстовом виде – WKT-формат (Well-Known Text);
2) в двоичном виде – WKB-формат (Well-Known Binary).

Текстовое представление данных определяет формат в виде текстовой строки, содержащей:

1) имя типа объекта (Point, Linestring, Polygon, Multipoint, Multilinestring, Multipolygon, Geometrycollection, Box, Line, Circle);

 © Архитектура СУБД. АО НПП «РЕЛЭКС», 1990-2026

2) пары чисел как координаты точек;

3) скобки для группировки элементов.

Например, геометрический объект в виде двух многоугольников в текстовом представлении будет иметь вид:

```
MULTIPOLYGON(((10 10, 10 20, 20 20, 20 15, 10 10)),
  ((60 60, 70 7, 80 60, 60 60)))
```

Двоичное представление данных определяет формат в виде байтовой строки, представляющей структуру данных (в терминах языка программирования C/C++) соответствующего графического объекта.

Например, представление точки задается следующей структурой данных:

```
WKBPoint
    {
    byte byteOrder; /порядок байтов в структуре
    uint32 wkbType; /тип геометрического объекта
    Point point; /координаты объекта
    }
```

Соответственно, WKB-представлением точки с прямым порядком байтов и координатами `(1,1)` является последовательность из 21 байта:

```
000100000000000000000F03F000000000000F03F
```

где:

00 – `byteOrder`

01000000 – `wkbType`

000000000000F03F – `X`

000000000000F03F – `Y`

Средства обработки геометрических данных

Обработка геометрических типов данных выполняется с помощью стандартных SQL-запросов СУБД ЛИНТЕР с привлечением встроенных геометрических функций.

Набор функций позволяет выполнять следующие группы геометрических операций:

1) загрузку в таблицу геометрических объектов в текстовом и двоичном представлении, например:

   - загрузка в текстовом представлении:

```
INSERT INTO GEO_TEST(P)
  VALUES (POINTFROMTEXT('POINT (25, 67)', 45));
```

   - загрузка в двоичном представлении:

```
INSERT INTO POINT_TEST
VALUES
```

```
(GEOMFROMWKB(HEX('0101000000000000000000F03F000000000000F03F')));
```

2) выборку значений геометрических объектов, например:
   - подсчет количества точек с координатами (1,2):

```
SELECT COUNT(*) FROM GEO_TEST WHERE P='POINT(1,2)';
| 23| |
```

   - выборка всех графических объектов:

```
SELECT ASTEXT(P) FROM POINT_TEST;
|POINT (1 1) |
|POINT (0 1) |
|POINT (1 2) |
|POINT (1 1) |
```

3) получать общую информацию о геометрическом объекте (вид объекта, его размерность, систему координат), например, размерность ломаной линии:

```
SELECT DIMENSION(GEOMFROMTEXT('LINESTRING(1 1,2 2)'));
```

4) получать специфическую информацию о геометрическом объекте (например, координаты начала и конца ломаной линии, количество узлов линии, ее длину, замкнутость и т.п.):
   - координаты конечной точки ломаной линии:

```
SELECT ASTEXT(ENDPOINT(GEOMFROMTEXT('LINESTRING(1 1,2 2,3 3)')));
|POINT (3 3) |
```

5) проверку взаимного расположения геометрических объектов (пересечение, совпадение, вложенность и т.п.), например,
   - проверка пересечения двух многоугольников. Если пересечение имеет место, выдается результат пересечения:

```
SELECT ASTEXT(INTERSECTION(
  GEOMFROMTEXT('POLYGON
    ((0 0,0 5,5 5,5 0,0 0),(2 2,2 4,3 4,3 2,2 2))'),
  GEOMFROMTEXT('POLYGON
    ((1 1,1 6,6 6,6 1,1 1),(2 3,2 4,4 4,4 3,2 3))')));
|GEOMETRYCOLLECTION (POLYGON ((5 1,1 1,1 5,5 5,5 1),
  (2 3,2 4,3 4,4 4,4 3,3 3,3 2,2 2,2 3))) |
```

Преобразование (кодирование) символьных данных

Все записываемые в БД символьные данные подлежат преобразованию (кодированию) на основе единого для всех пользователей БД алгоритма преобразования.

Обратное преобразование извлекаемой из БД символьной информации осуществляется в оперативной памяти ядра СУБД непосредственно перед выполнением конкретных действий с данными.

Выборочное преобразование символьных данных (с помощью персональных пользовательских алгоритмов преобразования) отдельных таблиц, их записей и/или полей не поддерживается.

 © Архитектура СУБД. АО НПП «РЕЛЭКС», 1990-2026

Защищенная БД

Примечание
Механизм создания защищенной БД поддерживается только в СУБД ЛИНТЕР БАСТИОН.

Для защиты данных от физической кражи файлов БД или при передаче БД в электронном виде в СУБД ЛИНТЕР используются встроенные (GOST, AES256, DES) без применения специализированных сторонних программных средств алгоритмы кодирования страниц файлов данных.

Защита БД выполняется с помощью указываемого при создании БД пароля защиты:

```
SET PROTECTED DATABASE PASSWORD "<пароль защиты>";
```

Для доступа к защищенной БД необходимо указывать пароль защиты, например, задавать ключ запуска ядра СУБД или запуска утилиты тестирования БД testdb:

```
-pass <пароль>
```

или

```
-passfile=<файл>
```

Утилитам, которые выполняют свои функции путем обращения к ядру СУБД (например, dbstore/loarel), указывать пароль защиты БД не требуется.

Хеширование пользовательских паролей

В целях безопасности пароли пользователей БД подлежат хешированию, иначе если хранить пароль в БД в открытом виде, то его можно получить хакерскими методами.

Механизм хеширования паролей предусматривает преобразование по детерминированному алгоритму входного массива данных произвольной длины в выходную битовую строку фиксированной длины.

В СУБД ЛИНТЕР для хеширования паролей используется алгоритм SHA-224 семейства SHA-2.

Алгоритм SHA-224 предотвращает:

- восстановление пароля по его хешированному значению;
- создание одинаковых хеш-значений для различающихся паролей. Любые изменения в пароле приводят, с очень высокой вероятностью, к различным хеш-значениям.

Примечание
В СУБД ЛИНТЕР со старым алгоритмом хеширования паролей пользователи продолжают работать по-старому вплоть до первой смены пароля. После смены пароля в новой версии СУБД, старая версия СУБД принимать этот пароль уже не будет. В случае возврата к старой версии СУБД ЛИНТЕР потребуется сменить пароль вновь.

Шифрование сетевого трафика

Для защиты сетевого трафика при передаче данных по сети можно использовать сетевой защищенный протокол обмена данными (см. документ «Сетевые средства», пункты

«Настройка защищенного соединения со стороны сервера» и «Настройка защищенного соединения со стороны клиента»).

Методы аутентификации

СУБД ЛИНТЕР поддерживает следующие методы аутентификации пользователя БД:

1) `IDENTIFIED BY <пароль>`: пароль пользователя хранится в БД. Для доступа к БД необходимо вводить имя и пароль пользователя;
2) `IDENTIFIED BY SYSTEM`: для доступа к БД надо вводить имя и пароль, указанные при регистрации в ОС. СУБД проверяет только наличие в БД пользователя с указанным именем, а аутентификацию выполняет ОС. Успешная аутентификация означает разрешение на доступ к БД;
3) `IDENTIFIED BY PROTOCOL` (только в среде ОС Linux и только при локальном доступе): для доступа к БД вводить имя и пароль не требуется. Предполагается, что пользователь прошел аутентификацию средствами ОС. Идентификация и аутентификация выполняются по имени пользователя ОС, от которого работает клиентское приложение. Пользователь с данным именем должен быть создан в БД;
4) `IDENTIFIED BY LDAP`: парольная аутентификация по LDAP-протоколу через LDAP-сервер. Для успешной аутентификации требуется наличие пользователя в БД ЛИНТЕР и в БД LDAP-сервера, при этом имя пользователя БД ЛИНТЕР должно соответствовать одному из атрибутов пользователя в БД LDAP-сервера (для аутентификации без предварительного поиска данный атрибут должен входить в состав уникального имени пользователя в БД LDAP-сервера).

   В режиме без предварительного поиска сначала идет подключение к серверу `LDAP_SERVER` и попытка выполнить соединение с уникальным именем, сформированным как `LDAP_PREFIX<имя>LDAP_SUFFIX`, где `<имя>` и пароль вводятся пользователем. Успешное соединение означает успешную идентификацию и аутентификацию.

   В режиме с предварительным поиском сначала идет подключение к серверу `LDAP_SERVER` и попытка выполнить соединение со специальными именем `LDAP_SEARCHDN` и паролем `LDAP_SEARCHPW` (либо анонимно, если эти учетные данные не заданы), после этого производится поиск пользователя в БД LDAP-сервера в каталоге `LDAP_BASEDN` с фильтром `LDAP_FILTER`. Чтобы для режима LDAP-аутентификации с предварительным поиском не требовалось задавать `DN` и пароль пользователя с правами поиска, необходимо разрешение на анонимный поиск в LDAP-базе. Найденное в результате поиска уникальное имя (если единственное) используется для попытки соединения с введенным пользователем паролем. Успешное соединение означает успешную идентификацию и аутентификацию;
5) `IDENTIFIED BY KRB`: идентификация и аутентификация по Kerberos-протоколу (по тикету, полученному при предварительной Kerberos-аутентификации).

   Механизм идентификации и аутентификации по протоколу Kerberos:

   - при указании специального флага в команде OPEN интерфейса нижнего уровня или пустого имени пользователя в регистрационных данных ЛИНТЕР-серверу передается тикет Kerberos-сервера. Этот тикет используется ядром СУБД ЛИНТЕР для получения имени пользователя и его аутентификации с использованием механизмов Kerberos;
   - идентификация и аутентификация по Kerberos-протоколу возможны, если имя пользователя в БД Kerberos-сервера совпадает с именем пользователя БД ЛИНТЕР,

 © Архитектура СУБД. АО НПП «РЕЛЭКС», 1990-2026

и этому пользователю задан метод идентификации и аутентификации по Kerberos-протоколу. Слежение за соответствием имен пользователей в БД аутентификации Kerberos-сервера и БД ЛИНТЕР возлагается на администратора СУБД ЛИНТЕР;

- для использования идентификации и аутентификации по Kerberos-протоколу в информационной системе должен быть установлен и настроен Kerberos-сервер (удаленный или локальный).

Примечание
Идентификация и аутентификация по Kerberos-протоколу не поддерживается для ЗОСРВ «Нейтрино».

Механизмы обработки данных

Каналы (соединения)

Чтобы обратиться с запросом к СУБД, любое клиентское приложение должно предварительно открыть *канал* (установить соединение с ядром СУБД).

Канал СУБД ЛИНТЕР – это механизм, осуществляющий обмен данными между клиентским приложением и ядром СУБД. Управление каналами выполняется с помощью интерфейса, используемого клиентским приложением для доступа к БД (интерфейс нижнего уровня (CALL-интерфейс), прикладной интерфейс) (рисунок 12). Каналы имеют свой уникальный числовой номер. Канал открывается при установлении соединения с СУБД ЛИНТЕР (после успешной идентификации и аутентификации пользователя), неявно жестко связывается с конкретным клиентским процессом и используется в дальнейшем для выполнения любых действий этого приложения с БД.

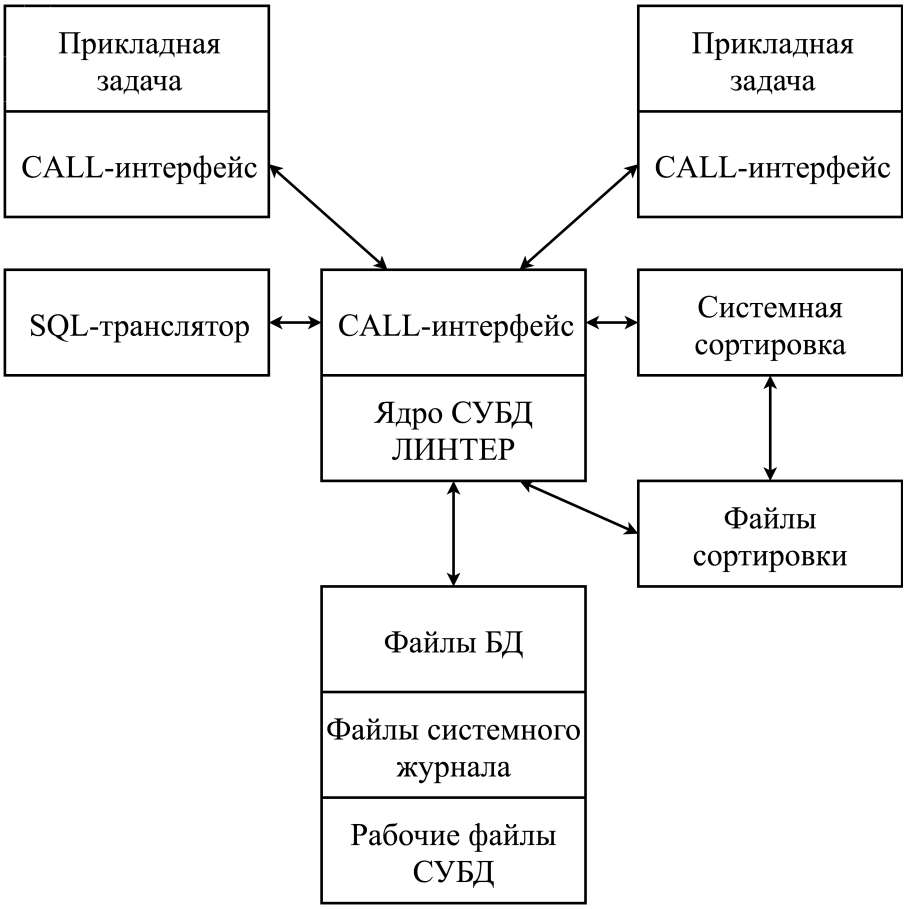


Рисунок 12. Схема доступа к БД

Архитектура СУБД ЛИНТЕР требует, чтобы клиентские приложения использовали разные (не пересекающиеся между ними) каналы. Параллельная (совместная) работа разных клиентских приложений по одному и тому же каналу невозможна. С каналом в ядре СУБД связаны различные характеристики, например:

- имя сетевого узла, на котором находится БД;
- идентификатор пользователя;
- режим обработки транзакций;

 © Архитектура СУБД. АО НПП «РЕЛЭКС», 1990-2026

- кодировка обрабатываемых данных;
- приоритет канала;
- диагностика последнего кода завершения;
- признак ожидания завершения асинхронной операции.

Очередь активных каналов в СУБД ЛИНТЕР обеспечивает квантование обработки запросов сразу по нескольким каналам: в каждый момент времени по одному из активных каналов нить ядра СУБД выполняет порцию (квант) текущей команды, затем выбирается следующий активный канал и т.д.

Канал может быть открыт как подчиненный от другого канала. В этом случае исходный канал играет роль соединения, а подчиненный – роль курсора.

Курсор – это канал, который связан с родительским каналом частью общих характеристик. С курсором соотносится область данных, описывающая состояние выполнения поискового SQL-запроса (например, результаты поискового запроса и последняя выданная запись). По одному и тому же курсору могут выполняться различные SQL-запросы, но курсор всегда будет содержать состояние последнего выполненного SQL-запроса. Приложение может открывать столько курсоров, сколько требуется (в рамках ограничений по памяти).

При открытии курсора устанавливается новый канал связи с СУБД ЛИНТЕР. Этому каналу идентификация и аутентификация уже не нужны, т.к. клиентское приложение уже выполнило соединение по главному каналу. Курсорный канал является дочерним (подчиненным) каналом по отношению к тому соединению, внутри которого он создается.

Соединение – средство объединения нескольких курсоров в одну транзакцию. Команда `COMMIT/ROLLBACK`, поданная по соединению, относится ко всем курсорам этого соединения. При закрытии соединения закрываются и все открытые в нем курсоры.

Как правило, приложение создает одно соединение (главный канал), и открывает внутри него несколько курсоров (подчиненных каналов), хотя может использовать и несколько независимых главных каналов. При открытии главного канала указывается, в частности, сервер СУБД ЛИНТЕР (логическое имя компьютера в локальной сети, на котором функционирует СУБД ЛИНТЕР и находится нужная БД). Открыв несколько главных каналов к разным серверам СУБД, приложение может одновременно работать с несколькими БД.

Открытие (закрытие) каналов, как и все другие операции доступа к БД, выполняются на уровне интерфейса нижнего уровня, используемого всеми программными интерфейсами СУБД ЛИНТЕР.

Механизм каналов обеспечивает параллельную обработку запросов, т.к. запросы, посланные по различным каналам, будут обрабатываться параллельно. По одному каналу запросы могут обрабатываться только последовательно.

Размер очереди каналов является параметром конфигурирования СУБД. Он определяет максимальное количество одновременных подключений к СУБД.

Доступ к данным

В СУБД ЛИНТЕР реализован следующий механизм доступа клиентского приложения к информации в БД:

1) клиентское приложение посылает СУБД через используемый интерфейс поисковый SQL-запрос;

2) СУБД обрабатывает полученный поисковый запрос. Если запрос корректный и данные найдены, она возвращает первую запись. Если запрос некорректный или полученная выборка данных пуста, возвращается соответствующий код завершения;
3) если результат поискового запроса содержит несколько записей, то клиентское приложение получает их поочередно с помощью простейших подкоманд типа «выдать следующую запись» (обычный режим), либо «выдать массив записей» (пакетный режим) выборки данных;
4) каждый такой сеанс связи с СУБД проводится по отдельному логическому каналу;
5) при организации логического канала ему выделяются требуемая оперативная память для хранения оттранслированного запроса и промежуточных результатов его обработки (например, номер последней выданной записи выборки данных).

Последовательный доступ

Если клиентскому приложению необходимо обработать результаты одного SQL-запроса, затем (возможно, исходя из результатов обработки) послать другой SQL-запрос, а потом по результатам двух предыдущих запросов добавить новую информацию в БД, то все эти три (или более) SQL-запроса будут выполняться последовательно, т.к. каждый последующий SQL-запрос зависит от результатов предыдущего. Для выполнения такой последовательности SQL-запросов можно не открывать три канала – достаточно одного. Последовательные SQL-запросы можно выполнять и по отдельным независимым каналам, однако такая процедура будет менее эффективной (с точки зрения оптимизации работы СУБД, а не клиентского приложения), так как в этом случае клиентское приложение займет большее количество каналов, используя ресурсы, которые, возможно, в это время требуется другим приложениям, что приведет к дополнительным задержкам, ожиданиям и пр.

При последовательном выполнении нескольких SQL-запросов с использованием одного канала каждый последующий SQL-запрос может посылать данные по тому же каналу, не дожидаясь получения всех данных предыдущего SQL-запроса (если полученной частично информации достаточно для формирования очередного SQL-запроса).

Параллельный доступ

Для параллельной обработки SQL-запросов требуется столько каналов, сколько SQL-запросов должно выполняться одновременно. Это условие необходимо соблюдать и в случае, когда в ответ на каждую запись выборки данных (результат выполнения поискового SQL-запроса) клиентское приложение должно сформировать и послать новый SQL-запрос. В такой ситуации нельзя допустить, чтобы обработка первого SQL-запроса была прервана одним из последующих SQL-запросов. Для реализации такого алгоритма нужно два канала: один – для обработки главного SQL-запроса, второй – для вспомогательного. Клиентское приложение должно циклически запрашивать очередную запись первого SQL-запроса и посылать по другому каналу новый сформированный SQL-запрос. В этом случае выполнение SQL-запросов, порождаемых по результатам обработки выбираемых записей, должно выполняться по отдельным каналам (параллельно).

Механизм параллельного доступа

Обработку SQL-запроса, в общем случае, выполняют три процесса: ядро СУБД, SQL-транслятор и процессор сортировки. Распараллеливание работы этих процессов возложено на операционную систему. Они работают независимо друг от друга, обмениваясь (асинхронно) сообщениями.

 © Архитектура СУБД. АО НПП «РЕЛЭКС», 1990-2026

Это первый уровень распараллеливания обработки SQL-запросов.

Например, ядро СУБД передает SQL-транслятору запрос на трансляцию и на фоне работы SQL-транслятора может обрабатывать другой SQL-запрос. Аналогично распараллеливаются процессы ядра СУБД и сортировки, SQL-транслятора и сортировки.

Второй уровень распараллеливания организует само ядро СУБД ЛИНТЕР. Ядро СУБД всегда принимает на обработку поступивший запрос. Если ресурсы ядра СУБД позволяют обработать запрос, то он ставится в очередь на обработку (иначе возвращается соответствующий код завершения), причем обработка этого запроса будет выполняться параллельно с обработкой других запросов, принятых на исполнение. Однако в отличие от ядра СУБД процессы SQL-транслятора, системных сортировок и транслятора хранимых процедур обрабатывают поступающие им от ядра СУБД запросы последовательно. Ядро исполняется строго в одной нити, и параллельность исполнения достигается за счет кооперативной многозадачности ядра.

СУБД ЛИНТЕР имеет настраиваемый параметр *квант обработки*, в течение которого непрерывно ведется работа над одним запросом. По завершении кванта обработки система диспетчеризации, проанализировав ситуацию, с учетом приоритетов и доступных ресурсов выбирает новый канал или текущий и планирует его работу на следующем кванте.

Квант обработки запроса

В реляционных СУБД в основе большинства операций манипулирования данными лежит поиск нужных записей (например, для удаления или передачи их клиентскому приложению). В СУБД ЛИНТЕР операция поиска может выполняться двумя способами:

1) просмотром записей таблиц;
2) поиском в индексных структурах.

Чтобы обеспечить возможность переключаться с одного активного канала на другой, необходимо ограничивать время обслуживания каждого канала. Возможны два способа ограничить время обслуживания канала: по длительности обслуживания и по количеству обрабатываемых записей. В первом случае ядро СУБД прерывается для переключения на другой канал после истечения кванта времени, во втором – после просмотра определенного числа записей.

То есть, в зависимости от того, какой режим квантования задан при конфигурировании СУБД, критерием прекращения работы с одним каналом и перехода к обслуживанию другого канала (в соответствии с его приоритетом) является либо длительность обслуживания канала, либо объем выполненной по каналу работы: количество просмотренных без прерывания записей или индексов.

Таким образом, существует возможность гибко задавать параметры квантования всей системы и реализовывать низкоприоритетные (фоновые) операции.

Механизмы защиты данных

При старте ядра СУБД проверяется целостность файлов БД путем сравнения эталонных контрольных сумм и фактических. При несовпадении - ядро СУБД будет остановлено.

Модель защиты данных в СУБД ЛИНТЕР основана на описании отношений между субъектами контроля и объектами доступа.

Субъектом контроля является пользователь БД, представленный в ней некоторым идентификатором с определенными характеристиками. Идентификатор используется для авторизации доступа к БД. Отдельно выделяется системный пользователь, или администратор безопасности (АБ). АБ – это пользователь, который является владельцем БД (всех её системных таблиц). Он не может быть удален из БД.

СУБД ЛИНТЕР может работать только с идентифицированным пользователем. Для идентификации пользователя используется уникальное символьное имя пользователя (не более 66 символов). Для того чтобы подтвердить подлинность идентификатора, пользователь обязан пройти процедуру аутентификации, то есть проверки правильности ввода его конфиденциального пароля.

Объектами доступа являются объекты БД (таблицы, представления, синонимы и др.).

Отношения между субъектами контроля и объектами доступа основаны на двух принципах:

1) дискреционный принцип – для каждой пары «субъект-объект» можно задать явное и недвусмысленное перечисление возможных действий субъекта по отношению к объекту. Например, для пользователя (субъекта) «Client» можно задать следующие действия по отношению к таблице «Sale» (объекту):
   - чтение данных (`SELECT`);
   - корректировку данных (`UPDATE`);
   - добавление данных (`INSERT`)

   и другие.

   Например, если действие `DELETE` не установлено, то пользователь "`Client`" не сможет удалять данные из таблицы "`Sale`".
2) мандатный принцип – контроль доступа осуществляется на основе сравнения меток безопасности объектов и субъектов (например, пользователь с меткой безопасности «Для служебного пользования» не сможет получить доступ к данным с меткой "`Секретно`", хотя доступ к таблице, в которой могут оказаться данные разного уровня секретности, ему не запрещен).

Примечание
Механизм мандатного доступа поддерживается только в СУБД ЛИНТЕР БАСТИОН.

Для исключения несанкционированного доступа (НСД) к информации в БД СУБД ЛИНТЕР применяет следующие методы:

- контроль целостности файлов БД;
- авторизация пользователей;
- дискреционная защита (привилегии, роли);

- мандатная защита;
- контроль доступа с удаленных компьютеров (рабочих станций);
- протоколирование работы пользователя;
- контроль хранения информации;
- удаление остаточной информации;
- аудит выполняемых операций.

Каждое функциональное расширение СУБД, связанное с хранением дополнительной информации, например, поддержка меток секретности влечёт за собой непременное замедление работы всей СУБД в большей или меньшей степени. Некоторые поисковые SQL-запросы при наличии комплекса средств защиты перестают оптимизироваться, увеличиваются размеры индексных структур, добавляются новые системные таблицы и т.п. Исходя из этого, следует использовать лишь минимально необходимый уровень защиты.

Поэтому не всегда оправданным подходом при проектировании системы безопасности БД является желание использовать одновременно все возможные средства комплекса средств защиты от НСД: все уровни, все группы, полный аудит (протоколирование всех неразрешённых действий), все виды расписаний работы с рабочими станциями, все виды работы с устройствами и т.п.

Контроль целостности

Контроль целостности БД выполняется с помощью следующих проверок:

1) контроль целостности системного словаря БД при запуске ядра СУБД;
2) контроль целостности хранимых процедур при запуске их на выполнение;
3) ежедневный контроль целостности БД в процессе функционирования СУБД;
4) контроль целостности БД в процессе ее восстановления из резервной копии.

При несовпадени контрольных сумм с ожидаемыми - ядро СУБД будет остановлено.

Авторизация пользователей

Пользователи – это субъекты БД. Каждая БД ЛИНТЕР имеет, как минимум, одного пользователя. Информация о пользователях БД хранится в системной таблице полномочий. Доступ пользователя к БД возможен только после прохождения процедуры авторизации доступа, в ходе которой пользователь должен предъявить свое регистрационное имя и пароль.

В СУБД ЛИНТЕР каждому пользователю, при его создании администратором БД, назначается одна из трех возможных категорий доступа к БД (или один из трех уровней доступа):

- Connect-категория. Предоставляет пользователю наименьшие права: доступ к БД с возможностью подавать SQL-запросы на манипулирование данными;
- 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;
- DBA-категория. Предоставляет пользователю уровень администратора БД с максимальными правами, включающими права Resource-категории и право создавать новых пользователей БД.

BACKUP-категория предоставляет пользователю права архивирования БД.

В зависимости от способа создания нового пользователя возможны следующие механизмы его идентификации и аутентификации:

1) `IDENTIFIED BY <пароль>`: устанавливает для пользователя режим идентификации и аутентификации по имени и паролю, хранящимся в БД ЛИНТЕР;
2) `IDENTIFIED BY SYSTEM`: устанавливает для пользователя режим встроенной аутентификации операционной системы. Это означает, что при доступе к БД надо указать имя и пароль, указываемый при регистрации в ОС. СУБД проверяет только наличие пользователя с указанным именем, а аутентификацию выполняет ОС;
3) `IDENTIFIED BY PROTOCOL`: задает идентификацию и аутентификацию пользователя при проверке доступа к БД по имени пользователя, зарегистрированного в ОС (при создании соединения с БД вводить имя пользователя и пароль не требуется). Данная идентификация доступна только локальным пользователям ЛИНТЕР-сервера. Пользователь-владелец БД не может идентифицироваться и аутентифицироваться таким способом. Запросы будут выполняться от имени пользователя, совпадающего с именем пользователя, запустившего клиентское приложение;

Примечание

Создание пользователя с помощью опции `IDENTIFIED BY PROTOCOL` поддерживается как в ОС Windows, так и в среде ОС Linux, но идентификация и аутентификация с помощью вышеуказанной конструкции поддерживается только в среде ОС Linux.

4) `IDENTIFIED BY LDAP`: устанавливает для пользователя режим парольной аутентификации через LDAP-сервер. Для успешной аутентификации требуется соответствие имени пользователя БД ЛИНТЕР одному из атрибутов пользователя в БД LDAP-сервера;
5) `IDENTIFIED BY KRB`: устанавливает для пользователя режим идентификации и аутентификации через Kerberos-сервер. Для входа под данным пользователем необходимо указывать пустое имя пользователя и/или задать специальный флаг в интерфейсе нижнего уровня.

Вне зависимости от способа создания пользователя его имя всегда хранится в БД ЛИНТЕР. Хеш пароля хранится в СУБД только в первом случае.

Аутентификация через LDAP-сервер выполняется следующим образом:

1) пользователь передает ЛИНТЕР-серверу имя и пароль обычным для интерфейса нижнего уровня (call-интерфейса) способом (через команду OPEN);
2) ЛИНТЕР-сервер проверяет наличие указанного пользователя в системной таблице `$$$USR` БД ЛИНТЕР;
3) если указанный пользователь найден в БД ЛИНТЕР, и он должен аутентифицироваться по LDAP-протоколу, то попытка соединения с LDAP-сервером зависит от установленного режима поиска уникального имени пользователя в БД LDAP-сервера;
   - если активен режим LDAP-аутентификации без предварительного поиска, то производится попытка соединения с LDAP-сервером с переданным ранее паролем и уникальным именем (`DN`), составленным согласно переменным окружения/ключам запуска СУБД ЛИНТЕР;
   - если активен режим LDAP-аутентификации с предварительным поиском, то производится попытка подключения к LDAP-серверу с использованием `DN` и

пароля для поиска (либо попытка анонимного подключения, если DN поиска не задано), после чего, в случае успеха, происходит поиск пользователя с помощью фильтра, составленного согласно переменным окружения/ключам запуска СУБД ЛИНТЕР; если найден только один пользователь, происходит попытка соединения с использованием найденного уникального имени (DN) и переданного ранее пароля;

4) в случае успешного соединения с LDAP-сервером пользователю предоставляется доступ к БД ЛИНТЕР-сервера.

Для использования LDAP аутентификации в информационной системе должен быть установлен и настроен LDAP-сервер (удаленный или локальный). Его местоположение задается с помощью специальной переменной окружения СУБД ЛИНТЕР (см. документ «Запуск и останов СУБД ЛИНТЕР в среде ОС Windows» и «Запуск и останов СУБД ЛИНТЕР в среде ОС Linux»).

Привилегии

Каждому пользователю предоставляются определённые права при работе с чужими объектами БД – *привилегии*. Предоставлять другим пользователям привилегии доступа к своим объектам БД (таблице, хранимой процедуре, триггеру и др.) может только владелец этого объекта. Привилегии устанавливают степень свободы данного пользователя при манипулировании чужими данными, например,

- SELECT – читать данные;
- INSERT – добавлять новые данные;
- DELETE – удалять данные;
- UPDATE – модифицировать данные;
- ALTER – изменять схему таблицы;
- INDEX – создавать/удалять/корректировать простой и составной индекс;
- REFERENCE – запрещать (разрешать) создавать ссылки на столбец (столбцы) таблицы;
- EXECUTE – выполнять хранимые процедуры;
- EXECUTE AS OWNER – выполнять чужие хранимые процедуры от имени их владельца;
- ALL – все привилегии для таблицы, то есть SELECT+INSERT+DELETE+UPDATE+ALTER+INDEX+REFERENCE.

Пользователь может получить привилегию двумя способами: явно или с помощью назначения ему роли.

Поскольку роли упрощают управление привилегиями, то привилегии обычно назначаются ролям, а не конкретным пользователям.

Роли

Некоторая совокупность прав дискреционного доступа может быть создана и назначена без привязки к конкретному пользователю. Такая именованная совокупность прав называется роль. После создания роли и присвоения ей прав она может быть назначена любому количеству пользователей, а каждому пользователю, в свою очередь, может быть назначено любое количество ролей.

Следующие свойства ролей облегчают управление привилегиями:

- упрощение назначений привилегий. Вместо того чтобы явно назначать одни и те же привилегии многим пользователям, администратор БД может присвоить эти привилегии определенной роли, а затем назначить эту роль конкретному пользователю БД или объединить пользователей с одинаковой ролью в группу и назначить роль этой группе;
- динамическое управление привилегиями. При необходимости изменения привилегии группе пользователей достаточно модифицировать привилегии лишь одной роли.

Мандатная защита

Мандатный принцип защиты информации основан на понятии *метка безопасности* или *метка доступа*.

Метка доступа в СУБД ЛИНТЕР состоит из трех составляющих: метка группы пользователя и два уровня доступа к этой группе.

Пользователи могут быть разбиты на непересекающиеся группы (до 250 групп). Администратор безопасности не должен менять свою группу, номер его группы всегда 0.

Включать пользователя в группу может только член этой группы или администратор безопасности (последний может включать пользователя в любую группу, не только свою).

Пользователи одной группы **без специального разрешения** не видят информацию, размещаемую в БД пользователями другой группы.

При внесении некоторым пользователем в БД информации вместе с данными в метке доступа сохраняется информация и о группе этого пользователя. Впоследствии доступ к этим данным смогут получить пользователи только этой группы и, возможно, нескольких других групп, если между группами установлено доверие.

В общем случае уровень доступа представляет собой идентификатор, соответствующий числовому значению в диапазоне от 1 до 10. Два уровня доступа отражают ограничения на действия, связанные с чтением и модификацией данных: уровень доступа на чтение `RAL` (Read Access Level) и уровень доступа на запись `WAL` (Write Access Level).

Метками доступа снабжается информация всех уровней – от таблицы до столбца и записи, включая значения полей записи, то есть уровни доступа являются характеристикой этих данных. Поэтому свой собственный уровень доступа может иметь даже отдельное значение конкретного атрибута конкретной строки таблицы. Это позволяет, в частности, защитить какие-то конкретные строки или даже отдельные ячейки таблицы, пометив их как секретные в таблице, которая в целом секретной не является.

Метки доступа являются неотъемлемой частью самих данных и физически хранятся вместе с данными.

Все субъекты доступа также снабжаются метками. При проверке доступа субъекта к конкретному объекту СУБД осуществляет дополнительную проверку и отвергает действие субъекта при отсутствии у него соответствующих прав.

Согласно правилам мандатного доступа, пользователь не увидит данных, секретность которых превышает его уровень доступа, и не сможет понизить секретность доступной

ему информации ниже назначенного ему уровня доверия, даже если он получит возможность видеть и/или модифицировать данные по дискреционному принципу.

Описание уровней доступа:

1) уровни доступа для пользователя (субъекта):
   - RAL-уровень доступа. Пользователь может читать информацию, RAL-уровень которой не выше его собственного уровня доступа;
   - WAL-уровень доверия на понижение уровня конфиденциальности. Пользователь не может вносить информацию с уровнем доступа (RAL-уровнем) более низким, чем данный WAL-уровень пользователя. То есть пользователь не может сделать доступную ему информацию менее секретной, чем указано в данном параметре.
2) для информации (данных) вводятся следующие уровни доступа:
   - RAL-уровень чтения. Информация может быть прочитана пользователем, RAL-уровень которого не ниже RAL-уровня информации (то есть только тем пользователем, который обладает достаточно высоким уровнем доступа);
   - WAL-уровень ценности или уровень доступа на запись (модификацию, удаление). Информация может быть модифицирована (удалена) пользователем, RAL-уровень которого не ниже WAL-уровня модифицируемой информации.
3) для таблиц и их столбцов уровни доступа имеют несколько иную интерпретацию:
   - RAL-уровень чтения. Указывает минимальный уровень доступа пользователя (RAL), который необходим для получения любых данных из таблицы (столбца), то есть RAL-уровень для таблиц и столбцов по смыслу идентичен RAL-уровню данных;
   - WAL-уровень записи. Ограничивает снизу уровень чтения данных (RAL), которые могут быть помещены в таблицу.

Пользователь не ограничивается в повышении уровня чтения данных. При желании он может заносить в БД информацию, RAL-уровень которой выше его собственного RAL-уровня. В частности, к таким пользователям можно отнести категорию информаторов, которые имеют самый низкий RAL-уровень. Они не могут читать никакой секретной информации, но могут размещать её в БД (для этого необходимо поднять их значение WAL-уровня).

При установленной мандатной защите уровни доступа субъекта и объекта БД контролируются СУБД ЛИНТЕР при попытке получения любого доступа субъекта к объекту.

Контроль доступа к БД с рабочих станций

Основополагающим понятием в процессе контроля доступа пользователя к БД с удаленного компьютера является понятие *сетевая станция*. Сетевой станцией для СУБД ЛИНТЕР является любая рабочая станция (компьютер), имеющая уникальный идентификатор – сетевой адрес.

СУБД ЛИНТЕР позволяет администратору безопасности регулировать доступ к БД с сетевых рабочих станций по следующим критериям:

- по графику (расписанию) работы пользователя; например, пользователю X доступ к БД разрешен по будням с 8.00 до 12.00 и с 13.00 до 17.00;
- по количеству одновременно активных логических соединений с СУБД;

- по списку разрешенных для доступа к БД станций; например, доступ к БД «Бухгалтерия» может быть запрещен с сетевых рабочих станций, установленных в службе технической поддержки;
- по уровням доступа;
- по списку разрешенных для доступа групп; например, доступ к БД «Бухгалтерия» с сетевых рабочих станций, установленных в финансово-экономическом отделе, разрешен только пользователям группы «Бухгалтеры» и запрещен пользователям группы «Экономисты».

СУБД ЛИНТЕР поддерживает несколько типов сетей, что требует различного подхода к интерпретации сетевых адресов.

В общем случае сетевой адрес состоит из адреса подсети (уникального в пределах всей сети) и адреса станции (уникального в пределах подсети). СУБД ЛИНТЕР позволяет управлять доступом, как на уровне конечной станции, так и на уровне подсети. В последнем случае ограничения, наложенные на всю подсеть, распространяются на все станции, расположенные в данной подсети.

Каждый сетевой адрес в СУБД ЛИНТЕР характеризуется следующими параметрами:

- типом сети (определяет внутреннюю структуру адреса);
- типом адреса (указывает на подсеть или конкретный узел);
- адресом в сети (собственно сетевой адрес; в зависимости от типа сети может включать/не включать адрес подсети);
- маской разрешенных групп пользователей (стандартная битовая маска, описывающая, разрешен ли доступ данной группы к станции);
- уровнями мандатного доступа (проверяется возможность пользователя выполнять соответствующие операции по отношению к данной станции);
- маской разрешенного времени доступа (с точностью до получаса описывается время, разрешенное для доступа со станции).

Управлять правами доступа с сетевых станций может только администратор безопасности СУБД.

При попытке установить соединение с некоторой сетевой станции СУБД выполняет следующие действия:

- проводит стандартную идентификацию и аутентификацию пользователя;
- проверяет наличие у пользователя Connect-категории или выше;
- получает у операционной системы тип сети и адрес сети – источника запроса на установку соединения;
- проверяет, ограничен ли доступ для данного пользователя (по расписанию работы);
- проверяет, существует ли для данного пользователя список разрешенных или запрещенных сетевых станций;
- если такой список существует, то проверяется совпадение меток доступа для пользователя и разрешенной станции (группа пользователя должна содержаться в маске групп пользователей у станции; RAL-уровень пользователя не должен быть выше RAL-уровня станции, WAL-уровень пользователя не должен быть ниже WAL-уровня станции);

 © Архитектура СУБД. АО НПП «РЕЛЭКС», 1990-2026

- если мандатный доступ разрешен, проверяется возможность данного пользователя работать с этой станции в текущий момент времени;
- если запрещенных комбинаций не обнаружено, то доступ разрешается.

Имя сетевой станции представляет собой идентификатор, который может использоваться администратором безопасности для разрешения/запрещения доступа пользователей с данного компьютера.

Характеристики сетевой станции включают:

- общее число неуспешных попыток доступа к БД со станции;
- текущее число неуспешных попыток доступа к БД со станции;
- флаги доступа (доступ запрещен, требуется проверка группы);
- уровни доступа с компьютера для мандатного доступа;
- маска разрешенных для доступа групп;
- маска временного доступа (битовое поле разрешенного времени работы со станции, с точностью до получаса на неделю);
- время последнего неудачного доступа к БД;
- время последнего успешного доступа к БД;
- дата и время, начиная с которого доступ к БД с данной станции разрешается;
- дата и время, до которого доступ к БД с данной станции разрешается;
- маска разрешенных для доступа к БД дней недели.

Уровни мандатного доступа для сетевой станции представляют собой два числа, аналогичные уровням доступа субъектов и объектов БД.

Маска разрешенных для доступа групп представляет собой битовую маску из 256 бит, 250 бит которой кодируют разрешение соответствующей группы на доступ к станции.

Дата и время, начиная с которого доступ к БД разрешается с данной станции, представляет собой одностороннее ограничение на дату начала разрешения доступа с данной станции.

Дата и время, начиная с которого доступ к БД запрещается с данной станции, представляет собой одностороннее ограничение на дату прекращения доступа с данной станции.

Маска разрешенных для доступа дней недели представляет собой битовую маску разрешенных для доступа дней недели.

К БД, в которой инициализирован мандатный доступ, доступ с сетевых станций по умолчанию запрещен. Перед использованием сетевой станции ее необходимо сделать видимой для СУБД ЛИНТЕР. Этого можно добиться двумя способами: создать станцию, то есть включить ее описание в список доступных СУБД станций или перевести СУБД ЛИНТЕР в режим беспрепятственной работы с неизвестными станциями (`UNLISTED STATIONS`), например,

```
grant access on unlisted station to all;
```

(разрешить доступ к этой БД со всех компьютеров).

Доступ физических лиц к консоли сервера СУБД ЛИНТЕР для изменения параметров работы со станциями должен строго ограничиваться административными мерами.

Защита ввода-вывода на внешний носитель

Примечание
Механизм защиты ввода-вывода на внешний носитель поддерживается только в СУБД ЛИНТЕР БАСТИОН.

Для размещения файлов таблиц и временных рабочих файлов СУБД ЛИНТЕР использует внешние устройства постоянного хранения информации.

Местоположение конкретного файла БД (файла таблицы или рабочего файла) внутри СУБД однозначно идентифицируется четырехсимвольным идентификатором – ЛИНТЕР-именем внешнего устройства. ЛИНТЕР-имя используется при создании новых базовых таблиц или изменении местоположения файлов уже существующих таблиц.

Защита внешних устройств необходима для того, чтобы пользователь, имеющий доступ к конфиденциальным данным, не мог поместить эти данные в ту таблицу, файлы которой находятся на некотором незащищенном устройстве (например, на флэш-диске). Администратор безопасности БД может ограничить набор физических каталогов, в которых будут храниться данные, и обеспечить недоступность этих каталогов другим пользователям средствами ОС.

Защита ввода-вывода на внешние устройства возможна только в том случае, если в СУБД инициированы расширенные средства защиты информации.

Описание защиты устройства содержит следующие данные:

- метка доступа – набор из двух значений: RAL-уровень и WAL-уровень внешнего устройства. Служит для принудительного контроля над созданием файлов (файлов таблиц и временных файлов) на описываемом устройстве. RAL-уровень устройства – это минимальный уровень доступа пользователя (RAL-уровень пользователя), необходимый для создания файлов таблиц на данном устройстве; WAL-уровень устройства – это максимальный WAL-уровень пользователя, необходимый для удаления таблицы;
- маска признаков доступа – выполняется/не выполняется проверка доступа к соответствующему устройству, запрещен/разрешен доступ к нему для СУБД. В последнем случае все SQL-запросы на создание новых объектов БД (получение информации из существующих объектов) будут отвергаться;
- маска разрешения доступа для групп пользователей – описывается доступ всех возможных групп;
- режим доступа к неизвестным устройствам (нужно ли запрашивать информацию о таких ЛИНТЕР-устройствах у ОС или отвергать работу с ними).

Протоколирование доступа к БД

Примечание
Механизм протоколирования доступа к БД поддерживается только в СУБД ЛИНТЕР БАСТИОН.

Расширенные средства защиты информации предполагают не только управление доступом пользователей к БД, но и фиксацию всех попыток обойти установленные ограничения (случайно или умышленно), регистрацию характера запрашиваемой из БД информации (с целью исключения ее утечки).

Указанные функции выполняет подсистема регистрации событий (аудита), которая позволяет регистрировать следующие события:

- включение механизмов идентификации и аутентификации;
- SQL-запросы на доступ к ресурсам БД;
- создание и удаление объектов БД;
- действия по изменению правил доступа;
- все попытки доступа к БД;
- действия администратора БД.

Все подлежащие регистрации действия записываются в системной таблице `$$$AUDIT`. Кроме регистрации событий, в эту таблицу может помещаться информация о внутренних кодах завершения СУБД, изменении состояния пользовательских событий, установленных в СУБД и т.д.

Запись информации в таблицу `$$$AUDIT` выполняется в циклическом режиме, то есть при превышении установленного размера новые записи размещаются с начала таблицы. Для исключения потери накопленной информации должно быть разработано специальное клиентское приложение, контролирующее степень заполнения таблицы `$$$AUDIT` и автоматически выгружающее данные из нее при угрозе переполнения.

Доступ к таблице `$$$AUDIT` имеют только пользователи DBA-категории и только на чтение.

В подсистеме протоколирования СУБД ЛИНТЕР предусмотрено три типа установок:

1) *глобальные*: для всех пользователей и объектов БД;
2) *персональные*: для конкретных пользователей или объектов БД;
3) *локальные*: для конкретных пользователей и объектов БД, то есть установки между конкретными пользователями и конкретными объектами.

Анализ установок проверяются СУБД в порядке их перечисления – глобальные, персональные и локальные. Таким образом, глобальные установки обладают наименьшим приоритетом, локальные – наибольшим.

Для протоколирования можно указать множество событий из разных групп:

- системные события (старт/останов/рестарт ядра СУБД, старт/останов подсистемы аудита, регистрация кодов завершения СУБД ЛИНТЕР);
- события, связанные с изменением схемы БД (создание/удаление объектов БД);
- события, связанные с подсистемой доступа (создание/изменение/удаление субъектов, привилегий, групп, уровней и т.д.);
- события, связанные с таблицами БД (выборка, изменение, добавление, удаление строк и т.д.);
- события, связанные с пользователями (соединение/отсоединение, открытие/закрытие курсоров, завершение транзакций, отказ в доступе и т.д.).

Удаление остаточной информации

Примечание
Механизм удаления остаточной информации поддерживается только в СУБД ЛИНТЕР БАСТИОН.

Для исключения несанкционированного доступа к данным рабочая оперативная и внешняя память после удаления данных очищается средствами СУБД с помощью записи маскирующей информации.

Это делается только в том случае, если инициализирована подсистема защиты информации от несанкционированного доступа.

© Архитектура СУБД. АО НПП «РЕЛЭКС», 1990-2026

Механизм транзакций

Механизм транзакций является частью СУБД ЛИНТЕР. Он позволяет обеспечить целостность БД, как в случае нештатных ситуаций, так и в случае многопользовательской работы. Для обеспечения надежной работы клиентских приложений необходимо понимание работы этого механизма и принципов его использования.

Под транзакцией понимается группа операций, выполняющихся как неделимое целое. В некоторых источниках транзакцией называют не группу операций, а эффект, который они производят. Транзакции являются составной частью любой информационной системы, к которой предъявляются требования надежности и являются ее основными строительными блоками. Эти требования обеспечиваются следующими свойствами транзакций:

- атомарность (неделимость). Гарантируется выполнение всей группы операций в случае удачного завершения транзакции и откат к состоянию до выполнения транзакции в случае сбоя или явного отката транзакции;
- целостность (непротиворечивость). Транзакция переводит БД из одного непротиворечивого состояние в другое. При этом подразумевается, что клиентское приложение является правильно написанным;
- изолированность. Для клиентского приложения создается иллюзия, что все остальные транзакции выполняются либо до, либо после его транзакции, что обеспечивает прозрачную работу в многопользовательском режиме;
- устойчивость. При удачном завершении транзакции гарантируется сохранение результатов ее работы даже в случае последующего сбоя в работе СУБД.

Для управления транзакциями в СУБД ЛИНТЕР служат специальные команды CALL-интерфейса и SQL-запросы. Любая транзакция должна заканчиваться выполнением оператора `COMMIT` или `ROLLBACK`, которые соответственно производят окончательные действия по выполнению транзакции, с сообщением в случае логической ошибки, или производят откат всех действий, которые транзакция произвела. В случае сбоя в работе СУБД (например, отключение питания) после перезапуска СУБД для всех не завершившихся транзакций команда `ROLLBACK` подается автоматически. Во время выполнения транзакции производимые ею в БД изменения, как правило, не должны быть доступны другим транзакциям (клиентским приложениям), т.к. это может привести к проблеме использования данных, которые не являются окончательными («грязные» данные).

Многопользовательский режим

При работе СУБД ЛИНТЕР в однопользовательском режиме не возникает проблем, характерных при взаимодействии двух и более клиентских приложений. В случае многопользовательского режима возникают специфические проблемы, которые СУБД должна предотвращать. Например, это могут быть логические ошибки клиентских приложений, приводящие к потере данных или некорректному построению отчетов.

Таких проблем немного. Если рассматривать весь набор операций с объектами БД, как операции чтения (`READ`) и записи (`WRITE`), то в случае двух пользователей различных комбинаций последовательного выполнения двух операций всего четыре:

1) READ – READ;
2) READ – WRITE;
3) WRITE – READ;

4) WRITE – WRITE.

В первом случае проблемной ситуации не возникает, так как последовательное чтение данных не приводит к их противоречивости. Проблемы возникают при выполнении операции записи, которая выполняется в результате SQL-запросов `INSERT`, `DELETE`, `UPDATE`.

Рассмотрим каждую из конфликтных ситуаций. Будем обозначать символами T1, T2 соответственно транзакции, выполняющиеся первым и вторым пользователем, а символом A – объект БД, который изменяется (например, запись) клиентским приложением или разными нитями (таблица 5).

Таблица 5. Возможные конфликтные ситуации

Конфликт	Описание
T1 WRITE A T2 READ A T1 WRITE A	Конфликт в том, что транзакция T2 прочитала содержимое объекта A на момент выполнения, когда он находится не в окончательном состоянии (до подачи `COMMIT`). Такой конфликт называется «грязным» чтением (dirty read). Эту ситуацию можно получить, выполняя построение отчета во время активной модификации БД сложными расчетами.
T1 READ A T2 WRITE A T1 WRITE A	Конфликт состоит в том, что транзакция T2 потеряла свои изменения до момента своего завершения. Такой конфликт называется «потерянным изменением» (lost update). Примером создания такой ситуации является попытка одновременной работы по изменению банковского счета, изменению количества товара на складе и т.д.
T1 READ A T2 WRITE A T1 READ A	Конфликт в том, что транзакция T1, выполняя повторно операцию чтения данных до завершения транзакции, получает новые (измененные) значения. Такой конфликт называется «неповторяемое чтение» (unrepeatable read). Проявление конфликта возможно при построении сложных отчетов, многократно обращающихся к БД, при одновременной ее модификации.

В СУБД ЛИНТЕР поддерживается пессимистичный (`PESSIMISTIC`) режим работы с транзакциями. При этом необходимо понимать, что желание избежать всех конфликтов в СУБД требует больших вычислительных расходов.

Примечание
Все DDL-операторы (операторы определения данных, например, создание таблицы) автоматически приводят к выполнению оператора `COMMIT` и не могут быть отменены СУБД.

Пессимистичный режим

В пессимистичном режиме предполагается, что к записям, которые будет использовать текущая транзакция, будут пытаться осуществить доступ другие транзакции. Поэтому перед любым изменением записей в БД (операторы `UPDATE`, `DELETE`) на объект накладывается блокировка, которая называется эксклюзивной (`EXCLUSIVE`). После успешной установки такой блокировки происходит изменение данных. В случае если объект до этого был заблокирован другим клиентским приложением (транзакцией), текущая транзакция переходит в состояние ожидания снятия блокировки. Выполнение ее будет временно приостановлено. Снятие блокировки осуществляется при завершении выполнения транзакции с помощью команды `COMMIT`

или ROLLBACK. Установка такой блокировки позволяет предотвратить конфликт «потерянных изменений». Транзакция не переходит в состояние ожидания, если выставляет блокировку повторно на уже заблокированную для себя запись.

Ожидание блокировки не производится, если SQL-запрос выполняется с опцией NOWAIT, в этом случае клиентскому приложению возвращается соответствующий код завершения («Требуемая запись заблокирована»).

SQL-запрос UPDATE в СУБД ЛИНТЕР всегда выставляет «длинную» по времени блокировку на запись. Это означает снятие такой блокировки только в конце транзакции.

Конфликт «грязного чтения» автоматически предотвращается в СУБД за счет установки «коротких» блокировок на чтение (SHARED) при выполнении SELECT-запроса. Это означает проверку каждой записи, попавшей в выборку данных, на присутствие у нее блокировки на запись (EXCLUSIVE). В случае если хотя бы одна запись в выборке данных заблокирована, выполнение SELECT-запроса приостанавливается до снятия блокировки другой транзакцией.

Для предотвращения конфликта «неповторяемого чтения» необходимо наличие «длинных» блокировок на чтение для тех записей, к которым был получен доступ.

Существует еще один конфликт, который присущ механизму блокировок. Этот конфликт носит название «фантомной записи». Механизм установки блокировки на одиночную запись здесь не является решением. Заблокировать запись, которой нет на момент выполнения транзакции, невозможно. Решение состоит в предварительном блокировании всей таблицы с помощью SQL-запроса LOCK TABLE … IN SHARED| EXCLUSIVE MODE. В таблице 6 приведены конфликты, требуемые блокировки и используемые для этого SQL-запросы.

Таблица 6. Правила установки блокировок для избежания конфликтов

Конфликт	Необходимый протокол блокирования	Используемые SQL-запросы	Производительность СУБД и возможность совместного доступа
Потерянные изменения	Длинные блокировки на запись	update … select …	Высокая
Грязное чтение	Длинные блокировки на запись, короткие блокировки на чтение	update … select …	Высокая
Неповторяемое чтение	Длинные блокировки на запись и на чтение	update … select … for browse| update или lock table in shared mode	Средняя
Фантомные записи	Длинные блокировки на запись и на чтение, предотвращение вставки данных	update … select … for browse| update или lock table in shared| exclusive mode	Низкая

Пессимистичный режим обработки транзакций необходимо применять в тех случаях, когда требуется изменять записи, которые с высокой вероятностью могут также изменяться другими параллельно работающими транзакциями.

В СУБД ЛИНТЕР для упрощения работы в пессимистичном режиме существует модификация этого режима – режим AUTOCOMMIT. В этом случае оператор COMMIT автоматически выполняется после каждого DML-оператора (запроса манипулирования данными).

Примечание
Большое количество блокировок уменьшает производительность СУБД, т.к. необходимы вычислительные ресурсы для их поддержки. Поэтому в случае, если в одной таблице одновременно блокировано большое число записей, СУБД автоматически пытается блокировать сразу всю таблицу.

Контрольные точки

Для более гибкого управления длинными транзакциями в СУБД ЛИНТЕР реализован механизм контрольных точек. Управление контрольными точками транзакций возложено на клиентское приложение. Оно может:

- создавать необходимое число контрольных точек;
- выполнять откат (ROLLBACK) транзакции к любой установленной контрольной точке;
- выполнять фиксацию изменений (COMMIT) транзакции до любой установленной контрольной точки.

Механизм контрольных точек позволяет клиентскому приложению возвращаться в транзакциях на «шаг назад» и осуществлять затем дальнейшие действия. Типичным примером использования контрольных точек являются задачи резервирования билетов, когда туристический маршрут не является прямым, а проходит через несколько пунктов. Резервирование билетов в каждый пункт маршрута осуществляется после установления контрольной точки для предыдущего отрезка пути. В случае если клиент не может воспользоваться предложенным вариантом передвижения по тем либо иным соображениям, клиентское приложение может вернуться в предыдущую контрольную точку и продолжить резервирование билетов по другому маршруту. В этом случае последовательность SQL-запросов будет выглядеть примерно следующим образом:

```
T1 SET SAVEPOINT A1
T1 SELECT …
T1 UPDATE …
T1 SET SAVEPOINT A2
T1 SELECT …
T1 UPDATE …
T1 ROLLBACK TO SAVEPOINT A2
T1 …
T1 COMMIT
```

Подтверждение транзакции

После выполнения команды COMMIT текущая транзакция завершается и начинается новая; блокировки, установленные транзакцией, сбрасываются. Изменения, проводимые клиентским приложением в БД, становятся доступными всем пользователям этой БД.

Если клиентское приложение имеет главный и подчиненные каналы и команда COMMIT подается по главному каналу, то COMMIT автоматически выполняется и для всех незавершенных транзакций в подчиненных каналах.

 © Архитектура СУБД. АО НПП «РЕЛЭКС», 1990-2026

Если приложение имеет главный и подчиненные каналы, все или часть которых функционирует в асинхронном режиме, то подача COMMIT по главному каналу должна выполняться после завершения всех асинхронных процессов, в противном случае может оказаться так, что не полностью завершенные изменения в асинхронных каналах будут зафиксированы в БД.

Если перед нормальным закрытием канала в нем имеется незавершенная транзакция, то по умолчанию она завершается командой COMMIT.

Если в транзакции были установлены контрольные точки, и фиксация изменений в них не производилась, то по команде COMMIT фиксируются все изменения от начала транзакции.

Если в транзакции были установлены контрольные точки, и в некоторых из них выполнялась фиксация изменений (то есть устанавливалось новое начало текущей транзакции), то по команде COMMIT фиксируются все изменения от нового начала транзакции.

Откат транзакции

После выполнения команды ROLLBACK текущая транзакция завершается и начинается новая; блокировки, установленные транзакцией (и клиентским приложением внутри транзакции), сбрасываются.

Если приложение имеет главный и подчиненные каналы и команда ROLLBACK подается по главному каналу, то ROLLBACK автоматически выполняется и для всех незавершенных транзакций в подчиненных каналах.

Если перед аварийным закрытием канала в нем имеется незавершенная транзакция, то по умолчанию она завершается командой ROLLBACK.

Если в транзакции установлены контрольные точки, но фиксация изменений в них не производилась, то по команде ROLLBACK выполняется откат всей транзакции.

Если в транзакции были установлены контрольные точки, и в некоторых из них выполнялась фиксация изменений (то есть устанавливалось новое начало текущей транзакции), то по команде ROLLBACK выполняется откат от нового начала транзакции до ее конца.

Системный журнал

Информация о всех изменениях БД, производимых транзакцией, фиксируется в специальном наборе файлов – системном журнале. Запись информации в системный журнал всегда предшествует выполнению изменения БД. Системный журнал является основой для обеспечения надежной работы СУБД, поэтому желательно располагать системный журнал и собственно БД на различных внешних носителях.

В момент активизации транзакции в системный журнал заносится отметка о начале выполнения транзакции в БД. При успешном завершении ставится соответствующая отметка. В случае сбоя СУБД и последующего ее рестарта или подачи команды ROLLBACK клиентским приложением информация из системного журнала используется для отката транзакции.

Если в СУБД одновременно обрабатывается большое количество транзакций, то необходимо увеличить размер файлов системного журнала. Системный журнал

может располагаться в одном или нескольких файлах операционной системы. Запись информации в системный журнал выполняется циклически, то есть при достижении конца файла новые записи размещаются на месте самых старых записей. При переполнении файлов системного журнала или нехватке дискового пространства поведение СУБД регулируется опциями, заданными на момент ее запуска.

Для обеспечения надежности БД необходимо периодически архивировать файлы самой БД и ее системного журнала. В случае потери данных, например, из-за аппаратного сбоя, возможно восстановление БД из ее архива (если он имеется) с использованием файлов системного журнала, в которых содержатся внесенные после архивации БД изменения.

© Архитектура СУБД. АО НПП «РЕЛЭКС», 1990-2026

Средства поддержки реального времени

В СУБД ЛИНТЕР есть свойства, позволяющие отнести ее к системам реального времени.

Механизмы для поддержки реального времени можно разделить на две группы:

1) универсальные – основаны на SQL-запросах и могут быть использованы в любых клиентских приложениях, имеющих интерфейс с СУБД ЛИНТЕР (например, аппарат событий);
2) специальные – поддерживаются только интерфейсами нижнего и прикладного уровней (например, асинхронная обработка). Эти механизмы доступны в клиентских приложениях, разработанных с использованием указанных интерфейсов.

Асинхронная обработка

В СУБД ЛИНТЕР реализована полнофункциональная асинхронность обработки SQL-запросов с определением процедуры обработки завершения асинхронной операции.

После отправки асинхронного SQL-запроса к СУБД управление сразу передается клиентскому приложению, которое может продолжать свою работу.

После выполнения ядром СУБД асинхронного SQL-запроса (а это, в зависимости от сложности запроса, может быть длительный процесс) работа клиентского приложения прерывается, и управление передается пользовательской процедуре завершения обработки асинхронного запроса (если она была задана) и после ее завершения вновь передается клиентскому приложению. Внутри пользовательской процедуры завершения асинхронной обработки можно также использовать асинхронные SQL-запросы, однако синхронные обращения к СУБД не разрешены.

Приоритетная обработка

Обработка SQL-запросов выполняется в соответствии с установленными для них приоритетами.

В СУБД ЛИНТЕР каждый SQL-запрос исполняется с определённым приоритетом, заимствованным из того канала (соединения или курсора), по которому он подан.

Приоритет – целое число в интервале от 0 до 256, влияющее на скорость обработки запроса в многопользовательской СУБД (то есть среди других одновременно выполняющихся SQL-запросов).

Чем выше (больше) приоритет запроса, тем больше времени и других ресурсов СУБД будет отдавать его обработке. При обработке равноприоритетных SQL-запросов, исполняемых одновременно, СУБД не отдаёт предпочтений ни одному из этих запросов, уделяя равные промежутки времени каждому из них.

Общий диапазон значений приоритета разбивается на группы:

- группа планирования с форой (под термином «фора» понимается режим вытесняющей многозадачности). Приоритеты от 0 до 99;
- группа циклического планирования. Приоритеты от 100 до 199;
- группа Real-time (или системных) запросов. Приоритеты от 200 до 249;

- зарезервированные значения приоритетов от 250 до 256.

Примечание
Не допускается выставлять зарезервированные системой приоритеты.

На каждом очередном кванте ядро СУБД осуществляет выбор запроса (канала) для обработки. При этом действуют следующие общие правила:

- для обработки всегда выбирается запрос с наивысшим (наибольшим) приоритетом;
- при обработке запроса в более приоритетной группе запросы во всех низкоприоритетных группах не обрабатываются.

То есть конкуренция запросов проходит только для запросов внутри группы.

В каждой из групп применяется собственная стратегия планирования выполнения множества обрабатываемых запросов.

В группе планирования с форой для обработки выбирается самый приоритетный запрос этой группы. После каждого кванта обработки приоритет этого запроса (если он отличен от нуля) снижается на единицу, так что приоритет в данном случае – это некая фора в ряду прочих запросов этой группы (рисунок 13).

При планировании с форой вновь поступивший запрос может непрерывно отработать несколько квантов, пока его приоритет не станет меньше приоритетов существующих запросов, после этого СУБД приступает к равномерному распределению ресурсов.

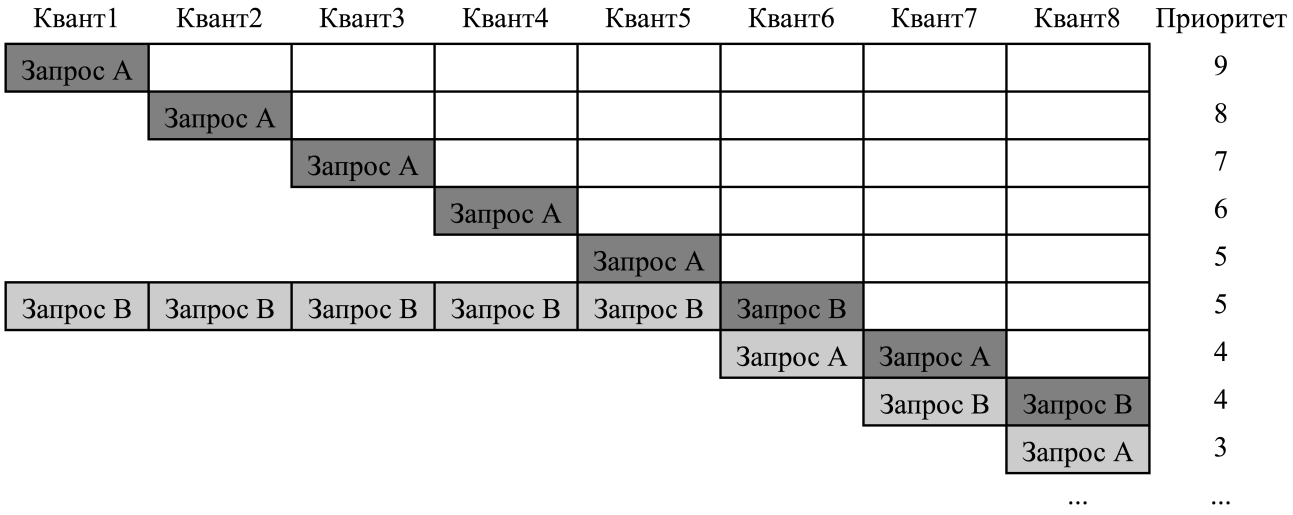


Рисунок 13. Группа планирования с форой

В группе циклического планирования циклически квантуется обработка запросов, имеющих наивысший приоритет (рисунок 14). Более низкоприоритетные запросы этой группы не будут обрабатываться до тех пор, пока не закончится выполнение всех запросов с более высоким приоритетом.

© Архитектура СУБД. АО НПП «РЕЛЭКС», 1990-2026

Квант1	Квант2	Квант3	Квант4	Квант5	Квант6	Квант7	Квант8	Приоритет
Запрос A	Запрос A	Запрос A	Запрос A	Запрос A	Запрос A	Запрос A	Запрос A	150
Запрос B	Запрос B	Запрос B	Запрос B	Запрос B	Запрос B	Запрос B	Запрос B	150
Запрос C	Запрос C	Запрос C	Запрос C	Запрос C	Запрос C	Запрос C	Запрос C	150
Запрос D	Запрос D	Запрос D	Запрос D	Запрос D	Запрос D	Запрос D	Запрос D	149

Рисунок 14. Группа циклического планирования

В группе Real-time (или системных) запросов всегда (на всех квантах) выполняется один самый приоритетный запрос. Все прочие запросы этой и прочих групп будут ждать его завершения.

Кроме того, каждый пользователь БД получает определённый приоритет – приоритет пользователя. Приоритет пользователя определяет приоритет по умолчанию всех SQL-запросов, поданных от его имени.

СУБД ЛИНТЕР позволяет изменять приоритеты обработки запросов в зависимости от ситуации.

Аппарат событий

Событие – механизм СУБД ЛИНТЕР, позволяющий отслеживать текущее состояние объектов реального мира, информация о которых хранится и обновляется в БД, в режиме реального времени.

Аппарат событий требуется, например, в ситуации, когда несколько клиентских приложений (операторов) управляют одним общим объектом их предметной области. В этом случае каждое клиентское приложение должно немедленно получать уведомление о том, что другие клиентские приложения изменили параметры поведения управляемого объекта. Получив уведомление (сигнал) об изменениях, клиентское приложение может запросить новые данные об объекте или предпринять другие действия, связанные с управлением объекта.

Например, некоторая задача прикладной системы SQL-запросом устанавливает событие `A` (допустим, модификация данных). Другие задачи могут потребовать, чтобы их оповестили о возникновении события `A`. Когда событие `A` реально произойдет, запросившие его задачи будут прерваны и управление будет передано определенным в этих задачах процедурам обработки уведомления о наступлении события. По концу обработки события (например, после того, как получены обновленные параметры управляемого объекта) выполнение задачи возобновится с того места, где она была прервана.

Аппарат событий СУБД ЛИНТЕР позволяет приложению устанавливать особые ситуации и обеспечивать реакцию на их возникновение.

Предварительная трансляция запроса

SQL-запрос можно один раз оттранслировать, а затем многократно выполнять каждый раз с новым константным содержанием параметров. Исключение этапа трансляции запроса существенно повышает скорость обработки ядром СУБД простых запросов. Допускается сочетать выполнение оттранслированного запроса и асинхронный режим его выполнения, что очень важно в системах управления технологическими процессами (например, при сборе информации с датчиков с последующим занесением ее в БД).

Приостановка и отмена обработки запросов

При работе в реальном масштабе времени клиентское приложение зачастую находится в очень жёстких временных рамках, поэтому в некоторых случаях большое значение имеет время реакции на событие, нежели реализация SQL-запроса в полном объёме.

Если SQL-запрос не выполнен за отведённое ему время (это может быть связано с большой нагрузкой в данный момент на сервер СУБД), то клиентское приложение вправе предпринять некоторые нештатные действия. Например, прервать выполнение данного SQL-запроса и «запомнить» его с тем, чтобы повторить позднее (когда нагрузка на сервер снизится до приемлемого уровня), или «приостановить» обработку запроса, разгрузив тем самым ресурсы СУБД. Через какое-то время (выбранное случайным образом или в зависимости от каких-то событий) можно будет продолжить выполнение приостановленного запроса.

Управление обработкой запроса клиентским приложением возможно только с помощью интерфейса нижнего уровня.

© Архитектура СУБД. АО НПП «РЕЛЭКС», 1990-2026

Пакетная обработка данных

Пакетная обработка данных эффективна в тех клиентских приложениях, в которых выполняется большое количество однотипных запросов на сбор и обработку информации.

Несколько однотипных запросов на добавление могут быть выполнены за одно обращение к СУБД или, напротив, за одно обращение можно получить сразу порцию записей поискового SQL-запроса. Это особенно важно при работе в сетевом режиме, т.к. уменьшает сетевой трафик, ускоряя тем самым работу всего комплекса задач, работающих в сети.

Мониторинг использования ресурсов

С помощью интерфейса нижнего уровня клиентское приложение может не только описывать и изменять структуру объектов БД, манипулировать данными (находить, добавлять, модифицировать, удалять), но и получать сведения о процессах, проходящих в ядре СУБД. Эта информация может пригодиться для правильной настройки ядра СУБД или для управления потоком заданий (запросов от различных клиентских приложений).

Производительность СУБД зависит от многих факторов – от используемой сервером СУБД операционной системы, от выделенных системных ресурсов, от параметров настройки ядра СУБД, от структуры SQL-запросов клиентских приложений к БД (например, преимущественное использования простых запросов, претранслированных, хранимых процедур, триггеров) и др. Все эти факторы оказывают разное влияние на производительность СУБД.

Единственным способом узнать, насколько эффективно функционирует СУБД, является отслеживание выполняемых ею операций, то есть мониторинг состояния и производительности СУБД в целом. Мониторинг может выполняться с помощью специальных команд интерфейса нижнего уровня.

Средства мониторинга позволяют собрать статистику о том, какие запросы посылает клиентское приложение и как СУБД их обрабатывает. Это особенно важно для многокомпонентных прикладных систем, работающих в многозадачной среде, когда запросы одного из компонентов системы могут задерживать обработку запросов других компонентов: при этом общая производительность прикладной системы снижается. Собрав и изучив статистику, администратор СУБД может разработать управляющую программу, которая в зависимости от собранной информации о поведении СУБД решает, какие из запросов следует приостановить (или даже вообще снять с обработки) в угоду более производительной обработке других, более приоритетных, запросов.

Для целей мониторинга имеется следующий набор команд интерфейса нижнего уровня:

- дать элемент очереди таблиц;
- дать элемент очереди столбцов;
- дать элемент очереди файлов;
- дать элемент очереди каналов.

Каждая из этих команд предоставляет информацию об основных очередях ядра СУБД. Свопинг этих очередей оказывает большое влияние на скорость обработки потока запросов.

Горячее резервирование БД

Для обеспечения сохранности данных в СУБД ЛИНТЕР включена подсистема горячего резервирования. Она выполняет полное дублирование процессов ведения БД главного сервера на резервном сервере. Горячее резервирование означает, что выход из строя одного из дублирующих серверов не будет замечен прикладными процессами, имеющими выполняющиеся запросы к БД главного сервера, и не повлияет на результаты последующих запросов. В подсистему заложены возможности «поднятия упавшего сервера» и восстановления процесса горячего резервирования после устранения нештатной ситуации.

Главный сервер – компьютер, на котором в данный момент работает ядро СУБД ЛИНТЕР под управлением системы резервирования и находится активная БД.

Резервный сервер – компьютер, работающий в паре с главным сервером, на котором в данный момент работает система резервирования и ведется копия БД главного сервера.

Серверы системы резервирования – компьютеры, предназначенные для выполнения функций главного/резервного серверов.

В основу горячего резервирования БД положены следующие принципы:

1) использование двух и более серверов в комплексе, каждый из которых может иметь одну и более линии связи;
2) на одном из серверов работает СУБД ЛИНТЕР. СУБД на этом сервере отрабатывает пользовательские запросы и хранит данные в своей БД. Этот сервер в дальнейшем будем называть главным сервером;
3) остальные сервера ведут копию БД главного сервера. Они непрерывно получают данные об изменениях БД с главного сервера по протоколу TCP/IP. В минимальной конфигурации должен быть хотя бы один резервный сервер;
4) управление запуском и остановом компонентов подсистемы резервирования берет на себя специализированная программа – «сервер резервирования»;
5) сервер резервирования при аварийном завершении компонентов подсистемы резервирования анализирует их коды завершения и, если возможно, осуществляет их автоматический перезапуск;
6) сервер резервирования, работающий на одном из компьютеров, обменивается информацией с остальными серверами резервирования и следит за их состоянием;
7) при отказе (или принудительном останове) главного сервера один из резервных серверов запускает СУБД ЛИНТЕР для работы с копией БД, полученной с главного сервера. С этого момента данный сервер становится главным сервером;
8) запросы клиентских приложений к СУБД ЛИНТЕР автоматически переадресуются главному серверу, даже при смене функции серверов.

Особенности функционирования системы резервирования:

- главный и резервный серверы не привязаны жестко к своим компьютерам;
- в процессе работы может происходить смена ролей серверов подсистемы резервирования по команде оператора или в случае проблем с главным сервером или линией связи;
- для того чтобы СУБД ЛИНТЕР полностью выполняла свои функции, достаточно работы главного сервера, однако в этом случае надежность комплекса снижается;

- резервный сервер может быть запущен в любое время после запуска главного. Его подключение «не видно» пользователям СУБД ЛИНТЕР;
- компьютер может выполнять функцию резервного сервера только тогда, когда работает главный сервер;
- при отказе главного сервера его функции принимает резервный сервер (становится главным). При этом все запросы к СУБД ЛИНТЕР, выполняющиеся в данный момент, возвращаются клиентским приложениям с соответствующим кодом завершения (информирует о том, что произошла смена главного сервера и запрос надо повторить);
- БД на резервном компьютере содержит полную копию рабочей БД на момент запуска подсистемы резервирования и накопленные изменения в БД главного сервера на текущий момент;
- поскольку архив рабочей БД постоянно возрастает в объеме (особенно при сильно обновляемой БД) за счет файлов системного журнала, то на резервном компьютере также работает ядро СУБД ЛИНТЕР в специальном режиме, которое переносит изменения из системного журнала в таблицы БД. Однако в момент получения таблиц БД также существует вероятность отказа главного сервера. Поэтому перед началом получения копии БД с главного сервера БД резервного сервера сохраняется в архивном каталоге. Таким образом, в подсистеме резервирования всегда (кроме начального момента) на резервном компьютере имеются две БД;
- подключение резервного компьютера и выравнивание БД выполняется незаметно для пользователя.

Автоматическое переключение серверов

В некоторых случаях для повышения надежности информационной системы можно обойтись без использования подсистемы горячего резервирования. Как правило, это системы с низкой динамикой изменения данных (например, информационно-поисковые системы), в которых изменение данных выполняется пакетом, что позволяет иметь одновременно необходимое количество дубликатов БД. Для повышения отказоустойчивости подобных систем можно применять вместо подсистемы горячего резервирования механизм автоматического переключения серверов БД с помощью протокола обмена LASSP (Linter Automated Standby Server Protocol).

При использовании протокола LASSP осуществляется попытка соединения со всеми ЛИНТЕР-серверами, перечисленными в строке файла сетевой конфигурации `nodetab` для этого протокола. При успешном соединении с узлом весь поток данных направляется через это соединение. При разрыве соединения, через которое осуществляется обмен данными, выполняется попытка автоматического переключения на другой ЛИНТЕР-сервер, принадлежащий данному LASSP. Если попытки соединения ко всем узлам оказались неудачны, приложению возвращается соответствующий код завершения.

Управление работой протокола LASSP выполняется с помощью сетевого драйвера клиента.

Удаленное управление СУБД

Удаленное управление СУБД ЛИНТЕР возможно с помощью следующих программ:

- сетевой администратор СУБД ЛИНТЕР (работает только в среде ОС Windows);
- удаленное управление компонентами СУБД.

Сетевое администрирование СУБД в среде ОС Windows

В сетевой операционной системе MS Windows NT все основные компоненты СУБД ЛИНТЕР и репликационного сервера запускаются и функционируют как службы (сервисы) MS Windows.

Для успешного удаленного администрирования СУБД ЛИНТЕР (сервисами) необходимо знать их текущее состояние и иметь возможность изменять параметры их функционирования.

Утилита `linadm` «Сетевой администратор СУБД ЛИНТЕР» предназначена для администрирования локальными и/или удаленными ЛИНТЕР и репликационными серверами, функционирующими в однородной сетевой среде семейства ОС Windows NT.

Основные возможности утилиты:

- запуск/останов ядра СУБД ЛИНТЕР;
- запуск/останов сетевых драйверов;
- создание/удаление БД;
- изменение параметров существующих БД;
- управление репликацией;
- изменение параметров сервисов СУБД ЛИНТЕР;
- работа с локальной и/или удаленными БД;
- тестирование физической целостности локальных БД.

Удаленное управление компонентами СУБД

Удаленное управление компонентами СУБД ЛИНТЕР реализовано на основе протокола SNMP (Simple Network Management Protocol – простой протокол управления сетью). Основная концепция протокола – вся необходимая для управления компонентом информация хранится у агента, который управляет работой компонентов, в так называемой «Базе данных управляющей информации» MIB (MIB – Management Information Base). MIB является набором переменных, характеризующих состояние контролируемого компонента.

Помимо стандартных переменных, поддерживаемых протоколом SNMP, в MIB можно включать дополнительные параметры, специфичные для данного компонента. Для того чтобы проконтролировать работу компонента, необходимо получить доступ к его MIB, и проанализировать значения некоторых переменных.

SNMP работает на основе протокола UDP и для общения с сетью использует порт с номером 161 (для посылки уведомлений – порт с номером 162). Использование UDP в

качестве основы SNMP означает, что данные передаются без установления соединения. Это дает возможность существенно уменьшить требования к сетевой инфраструктуре и накладные расходы на передачу данных.

SNMP поддерживает механизм авторизации с помощью имени пользователя и его пароля.

Функционально в состав системы удаленного управления компонентами СУБД ЛИНТЕР входят:

- агент системы управления;
- утилиты администратора удаленного управления.

Агент принимает SNMP-пакеты и выполняет соответствующие им действия, то есть посылает значение запрашиваемой переменной, устанавливает значение переменных, выполняет периодическое обновление информации MIB, выполняет в ответ на установку соответствующих переменных некоторые операции.

Работа агента управляется менеджером – программой, работающей на компьютере, с которого выполняется удаленное управление компонентами. Функции менеджера удаленного управления компонентами выполняет набор утилит. Агент выступает посредником между внутренними структурами управляемого компонента и менеджером.

Обычно взаимодействие происходит по инициативе менеджера и выглядит следующим образом:

- менеджер отправляет запрос агенту;
- агент обрабатывает запрос, собирает требуемые данные и отправляет их назад менеджеру;
- менеджер получает запрошенные данные и обрабатывает их в соответствии с предусмотренным алгоритмом.

В некоторых случаях агент может самостоятельно инициировать обмен данными. Обычно у агента должен быть список важных событий, о наступлении которых он обязан оповестить менеджера. Менеджер по своему усмотрению выполняет какие-либо действия в ответ на оповещение. Например, такими событиями могут быть аварийное завершение работы наблюдаемого компонента, аварийная перезагрузка, вызванная потерей питания или любая другая критическая ситуация. Процедура оповещения в терминах протокола SNMP называется отправкой ловушки (SNMP Trap). В сообщении уведомления агент посылает данные, специфичные для обнаруженного события, если они есть в MIB. Уведомление посылается одному или нескольким получателям, список которых хранится в MIB.

Локальное управление компонентами СУБД

Для локального управления компонентами СУБД ЛИНТЕР (утилита создания БД, ядро СУБД и другие компоненты) в среде ОС Linux, ЗОСРВ Нейтрино используется библиотека `linctrl`. Средства библиотеки позволяют создавать полностью автоматические (без участия человека) приложения БД для использования в автономных аппаратных комплексах реального времени или мобильных устройствах.

Клиентское приложение должно быть написано на языке программирования C/C++.

В частности, функции библиотеки позволяют выполнять из клиентского приложения следующие операции:

1) создание БД;
2) создание системных словарей БД;
3) запуск ядра СУБД;
4) управление активным ядром СУБД;
5) проверку текущего состояния ядра СУБД;
6) останов ядра СУБД.

© Архитектура СУБД. АО НПП «РЕЛЭКС», 1990-2026

Программные интерфейсы

Программные интерфейсы предназначены для доступа к информации БД из клиентских приложений, написанных на различных языках программирования или разработанных с помощью различных инструментальных средств.

Список программных интерфейсов СУБД ЛИНТЕР и их краткие характеристики приведены в таблице 7.

Таблица 7. Программные интерфейсы СУБД ЛИНТЕР

Обозначение интерфейса	Описание
call	Интерфейс нижнего уровня. Является базовым интерфейсом СУБД ЛИНТЕР. Предназначен для использования в программах на языке программирования C/C++. Предоставляет клиентскому приложению максимальные возможности для динамического управления обрабатываемыми данными (асинхронная обработка, изменение приоритетов запросов, мониторинг, управление выполнением запросов и т.п.). Позволяет разрабатывать высококачественные приложения системного уровня
LinAPI	Интерфейс верхнего уровня (прикладной интерфейс). Разработан с учетом спецификаций X/OPEN на основе интерфейса нижнего уровня. Предназначен для использования в программах на языке программирования C/C++ или Pascal. Предоставляет клиентскому приложению широкие возможности для управления обрабатываемыми данными (асинхронная обработка, работа с претранслированными запросами, управление соединениями и курсорами и т.п.). Позволяет разрабатывать высококачественные приложения системного и прикладного уровня
PCI	Встроенный SQL (имеет режим совместимости со спецификацией PRO*C фирмы ORACLE). Разработан на основе интерфейса верхнего уровня. Предназначен для использования в программах на языке программирования C/C++. Предоставляет клиентскому приложению широкие возможности для манипулирования обрабатываемыми данными непосредственно в виде SQL-операторов (в том числе и претранслированных). Позволяет разрабатывать прикладные приложения достаточно высокого уровня
ODBC	ODBC (3.x)-драйвер (включая 3.8). Разработан на основе спецификаций стандарта Microsoft ODBC 3.0. Предназначен для разработки приложений, основанных на технологии ODBC и использующих язык SQL в качестве стандарта языка обработки данных для доступа к реляционным БД. Позволяет осуществлять максимальную переносимость приложения с одной СУБД на другую без учета их специфики
JDBC	Драйвер JDBC (3, 4, 4.1, 4.2). JDK (1.4, 1.5, 1.6, 1.7, 1.8).
PHP DBX Pear::db PDO	PHP-интерфейс версий 5.x и 7.x. Предназначены для доступа к БД из программ, написанных на языке программирования PHP
Perl DBI	Perl-интерфейсы. Предназначены для доступа к БД из программ, написанных на языке программирования Perl

Обозначение интерфейса	Описание
TCL/TK	TCL/TK-интерфейс. Предназначен для доступа к БД из программ, написанных на языке программирования TCL/TK
Python	Python-интерфейс. Предназначен для доступа к БД из программ, написанных на языке программирования Python
.NET	.NET 6.x/7.x/8.x/9.x-интерфейс (включая поддержку LINQ и Entity Framework)
Qt	Qt (4.x, 5.x, 6.x) - интерфейс для мультиплатформенной C++ Qt-библиотеки
Ruby	Интерфейс для доступа к БД ЛИНТЕР из приложений, разработанных на языке программирования Ruby

Обеспечение сохранности данных

Резервное сохранение и восстановление БД

В процессе функционирования СУБД всегда существует вероятность отказа ОС, вычислительного оборудования или самой СУБД. Если в результате неисправности диска файлы БД будут физически повреждены, то для их восстановления необходимо иметь резервные копии. Поэтому создание резервных копий БД – необходимое условие для надежного и бесперебойного функционирования СУБД.

Способы получения копии файлов БД ЛИНТЕР:

1) простое пофайловое копирование БД при остановленном ядре СУБД на другой (резервный) носитель данных;
2) полное или выборочное архивирование БД, осуществляемое сервисными средствами СУБД ЛИНТЕР;
3) оперативное архивирование БД, которое осуществляется непосредственно ядром СУБД параллельно с обработкой SQL-запросов клиентских приложений.

В двух последних случаях возможна как полная, так и инкрементная (нарастающая) архивация.

Полное резервное сохранение БД (архивация) – это физическое сохранение всей структуры БД и хранящихся в ней данных в файл архива.

Инкрементная архивация предполагает создание сначала полного архива БД, а затем периодическое добавление к архивному файлу накапливаемых в БД изменений.

Автономное архивирование БД

Автономное архивирование выполняется специальными сервисными утилитами СУБД ЛИНТЕР. Утилиты позволяют выполнять не только «горячее» архивирование БД (то есть архивирование без останова ядра СУБД), но и восстановление БД, тестирование файлов архива, осуществлять операции с архивами на магнитной ленте. При архивировании информация из БД сохраняется в одном или нескольких бинарных файлах (томах).

Помимо полного сохранения БД возможно выборочное архивирование объектов БД путем задания маски сохраняемых объектов, например, можно сохранить отдельные таблицы (как с данными, так и без них – только схему таблицы), внешние ключи, представления и синонимы конкретного пользователя.

Поскольку архивирование является достаточно ресурсоемкой операцией, предусмотрены ключи установки приоритета процесса архивирования. Приоритет имеет три уровня градации: LOW, NORMAL и HIGH. При самом низком приоритете процесс архивирования делает перерывы в работе, позволяя, таким образом, ядру СУБД обработать запросы клиентских приложений. При самом высоком приоритете таких задержек нет, и процесс архивирования выполняется быстрее.

Для создания инкрементных архивов утилита архивирования использует механизм контрольных точек, то есть при старте инкрементного архивирования в системном журнале СУБД устанавливается контрольная точка, и при последующем запуске инкрементного архивирования сохранение информации из БД будет вестись только

с этой контрольной точки. Чтобы уменьшить рост размера БД (за счет накопления в системном журнале подлежащей архивированию информации), неиспользуемые контрольные точки необходимо регулярно удалять.

Архивирование БД должно выполняться регулярно, в зависимости от динамики изменения хранящейся в ней информации. Для автоматизации этого процесса в СУБД ЛИНТЕР имеется язык описания сценария архивирования (BSL). Конструкции языка позволяют:

- задавать действия утилиты архивирования до начала сеанса архивирования (например, создавать нужные каталоги на диске);
- указывать объекты БД, подлежащие архивированию;
- устанавливать периодичность и время начала архивирования (например, каждый день (раз в неделю, в конце месяца и т.п.) в 3 часа по московскому времени);
- задавать действия утилиты архивирования после завершения сеанса архивирования (например, удалять самую старую версию архивного файла, сохраняя только последнюю и предпоследнюю версии).

Сценарий архивирования запускается на выполнение один раз, после чего утилита архивирования автоматически повторяет его в соответствии с заданной периодичностью.

Оперативное архивирование

Оперативное архивирование выполняется непосредственно ядром СУБД параллельно с обработкой SQL-запросов клиентских приложений. Эта операция инициируется с помощью соответствующего SQL-запроса, подаваемого клиентским приложением (в том числе и с удаленной сетевой станции).

С помощью SQL-запроса можно запустить процессы как полного, так и инкрементного архивирования и установить желаемый режим архивирования:

- синхронный. В этом режиме клиентское приложение, подавшее SQL-запрос на запуск процесса архивации, перейдет в ожидание завершения процесса архивирования и получит от ядра СУБД уведомление (код возврата) о завершении процесса архивации только по его окончании. Естественно, в этом режиме необходимо обеспечивать соединение клиентского приложения с СУБД в течение всего процесса архивации;
- асинхронный. В этом режиме клиентское приложение, подавшее SQL-запрос на запуск процесса архивации, сразу получит от ядра СУБД уведомление (код возврата) об успешном или неуспешном запуске процесса архивирования и при успешном запуске архивирование будет выполняться параллельно с работой клиентского приложения. В случае успешного старта процесс архивации работает с БД по собственному каналу. Соединение клиентского приложения с БД в течение всего времени архивации не требуется.

Создаваемый при оперативном архивировании файл архива полностью совместим с форматом архивных файлов, создаваемых при автономном архивировании, поэтому к такому файлу архива применимы все возможности, предоставляемые сервисной утилитой архивирования, в том числе восстановление БД, тестирование архива и т.п.

При оперативном архивировании сохранение БД осуществляется через определенные кванты времени и не препятствует работе остальных каналов или клиентских приложений. В СУБД ЛИНТЕР возможна одновременная работа нескольких процессов

 © Архитектура СУБД. АО НПП «РЕЛЭКС», 1990-2026

оперативного архивирования. Во время работы процесса архивации файл архива открывается в эксклюзивном режиме, то есть модификация или удаление этого файла средствами ОС не допускается.

Меры безопасности при архивировании БД

Процедура архивирования/восстановления БД является ответственным мероприятием. Допущенные ошибки могут быть непоправимы (например, случайно удален или запорчен архив БД перед ее восстановлением). Чтобы минимизировать (а в некоторых случаях и вообще исключить) ущерб от неправильных действий пользователей БД, утилита архивирования выполняет максимально возможную проверку корректности заданного сценария архивирования.

Например, в команде на создание архива пользователь по какой-либо причине вместо правильного имени указал ошибочное имя (файл операционной системы или же архивный файл от другой БД). Если при этом в команде архивирования будет присутствовать опция `REWRITE`, то этот файл будет удален утилитой перед созданием файла архива. Тем самым при создании архива нарушится целостность либо операционной системы, либо функционирующих в ней программ.

Чтобы избежать подобных ситуаций, перед процедурой удаления указанного в команде архивирования файла СУБД будет выполнять проверку корректности заданного файла по следующим критериям:

- это должен быть файл архива, созданный в результате автономного или оперативного архивирования (то есть файл с расширением `.lhb`);
- версия БД, в которой создавался архив, должна совпадать с текущей версией СУБД ЛИНТЕР;
- имя пользователя, создавшего файл архива, должно совпадать с именем пользователя, осуществляющего попытку перезаписи этого файла. Исключение делается только для пользователя, создавшего БД (по умолчанию – пользователь SYSTEM);
- спецификация пути к файлу не должна содержать абсолютного пути;
- в спецификации пути к файлу не должны присутствовать каталоги и переходы на уровень вверх на каталог (/../);
- в качестве имени устройства должны выступать устройства, описанные в системной таблице `$$$DEVICE` СУБД ЛИНТЕР (либо устройство `SY00`).

Если указывается имя файла без имени устройства, сохранение будет происходить в каталоге, на который указывает переменная среды окружения `SY00`.

Переменная среды окружения `SY00` определяет расположение основных файлов БД, её системных таблиц. Устанавливаются переменные среды окружения средствами ОС.

Если в процессе оперативного архивирования выявилось недостаточное количество свободного места на диске для размещения архивного файла, то СУБД можно указать дальнейший порядок действий: остановить либо все работающие в асинхронном режиме процессы архивации, либо подпадающие под определенные критерии. Клиентскому приложению будет возвращено число остановленных процессов архивации.

Останавливать процессы архивации может либо пользователь, который их запустил, либо пользователь DBA-категории с соответствующей привилегией.

Асинхронная репликация данных

Репликация данных – это приведение нескольких БД, включенных в систему репликации, в идентичное (по возможности) и непротиворечивое (не вызывающее нарушений целостности) состояние, сопровождающееся взаимным внесением изменений. Все БД системы репликации должны иметь одинаковую структуру. СУБД ЛИНТЕР поддерживает механизм асинхронной репликации между серверами БД.

Для управления системой репликации на логическом уровне используются правила репликации, которые создаются с помощью языка SQL. Правила репликации содержат информацию о том, какие объекты БД, куда и каким образом должны быть реплицированы.

Механизм асинхронной репликации

Для поддержки целостности БД, входящих в системы репликации, СУБД ЛИНТЕР использует механизм асинхронного тиражирования (далее по тексту – репликации) транзакций. Его суть состоит в том, что обработка данных сначала выполняется локально, на отдельном сервере, а подлежащие репликации данные позднее копируются на те сервера, где они должны использоваться. При таком методе поддержки логической целостности БД системы репликации имеет место некоторая рассинхронизация состояния локальных БД во времени по отношению друг к другу, то есть изменение состояния одной локальной БД отстает от изменения другой локальной БД во времени.

Если один из серверов системы репликации, требующий обновления тиражируемых данных, выходит из строя, то средства репликации СУБД ЛИНТЕР продолжают работать с остальными серверами, при этом обновление данных на вышедшем из строя сервере после его ремонта произойдет автоматически. Таким образом, ошибка на одном узле системы реплицируемых БД не влияет на работу остальных узлов.

Механизм асинхронного тиражирования транзакций гарантирует доставку измененных данных на вторичные серверы непосредственно после завершения транзакции на основном сервере, если вторичный сервер доступен, или сразу после подключения его к сети. Такой подход предполагает хранение дублирующей информации в различных узлах сети и может обеспечить, по сравнению с другими подходами к репликации, снижение трафика, улучшение времени ответа системы, а также позволяет оптимизировать нагрузку на серверы.

Асинхронная репликация, в отличие от двухфазной синхронизации, не обеспечивает полной синхронности информации на всех серверах системы репликации в любой момент времени. Синхронизация происходит через некоторый, обычно небольшой, интервал времени, величина которого определяется быстродействием соответствующего канала связи. Для большинства задач кратковременное наличие устаревших данных в удаленных узлах вполне допустимо.

Вместе с тем, асинхронная репликация транзакций принципиально обеспечивает целостность данных в системе репликации, так как объектом обмена данными здесь является логическая единица работы – транзакция, а не просто данные из измененных таблиц.

Модель асинхронной репликации

Взаимодействие компонентов репликации данных приведено на рисунке 15.

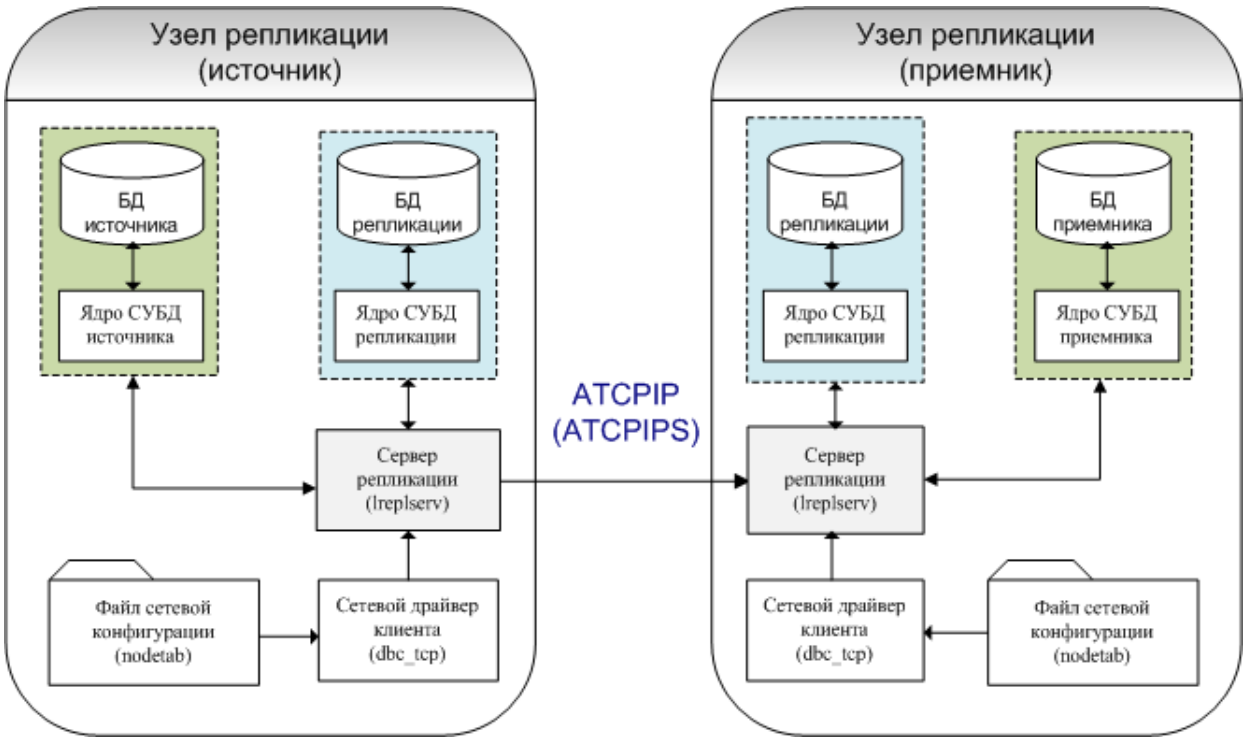


Рисунок 15. Модель репликации в СУБД ЛИНТЕР

В системе асинхронной репликации участвуют два или более ЛИНТЕР-серверов, на каждом из которых работает:

- ядро СУБД источника (ядро СУБД приемника);
- ядро СУБД репликации;
- сетевой драйвер клиента
- сервер репликации.

Объектами репликации являются таблицы БД, список которых вместе с правилами и адресами рассылки хранится в БД-источнике репликации.

БД может быть только приемником, только источником или одновременно и источником и приемником данных репликации. На одном сервере может быть запущено несколько серверов репликации для разных БД.

Очереди репликации – это таблицы БД репликации. БД репликации используется сервером репликации для хранения данных, подлежащих рассылке и принятых от других серверов системы репликации. Таблицы БД репликации могут быть использованы администратором системы репликации для анализа сбойных ситуаций или же для сбора статистики.

Сервер репликации на стороне БД-источника, получает информацию об изменении данных в БД-источнике и сохраняет ее в очереди рассылки (таблица БД репликации). Элемент очереди рассылки включает в себя полную информацию о старом и новом состоянии реплицируемой записи, адрес назначения, номер канала, производящего операцию, номер транзакции и время операции. В качестве первичного ключа этой таблицы используется время операции. Затем сервер репликации устанавливает соединение с сервером репликации приемника и отправляет ему измененные данные. Соединение устанавливается один раз и далее процесс рассылки и приема данных работает по этому соединению.

Сервер репликации приемника, получает данные от сервера репликации источника и помещает их в приемную очередь (таблица БД репликации).

После этого сервер репликации приемника формирует ответ, уведомляющий отправителя о нормальном завершении приема данных. А затем (после того, как получена команда завершения транзакции `COMMIT`) сервер репликации приемника вносит изменения в таблицы БД-приемника репликации. Коды завершения транзакций сохраняются в таблице приемной очереди.

Репликация данных происходит по первичным ключам, то есть каждая таблица, подлежащая репликации, должна содержать первичный ключ, значение которого используется для идентификации (при удалении и изменении значений) в реплицируемых таблицах.

Есть три модели поведения системы при возникновении рассогласования между значениями записей в БД реплицируемых серверов. Если такая ситуация возникла во время выполнения операции корректировки/удаления данных, можно предпринять следующие действия:

- игнорировать несовпадение старого значения;
- обязательно проверить старое состояние и вернуть ошибку, если нет полного совпадения;
- если не совпадают числовые значения, сохранить разницу между старым и новым значением.

Неразрешимые коллизии могут возникнуть при одновременной вставке во взаимно реплицируемые таблицы одинакового значения первичного ключа.

В качестве протокола проделанной работы используется файл трассировки. Он может формироваться как в виде одного файла, так и в виде группы файлов.

При необходимости администратор системы репликации может очистить очереди сервера репликации, при этом будут удалены все уже реплицированные записи, или только записи за указанный период.

Управление асинхронной репликацией в среде ОС Linux, ЗОСРВ Нейтрино выполняется с помощью командного файла или специальных скриптов, а в среде Windows может выполняться как с помощью командного файла, так и с помощью сервисных средств СУБД ЛИНТЕР.

Функции компонентов системы репликации

Функции компонентов системы асинхронной репликации СУБД ЛИНТЕР приведены в таблице 8.

Таблица 8. Функции компонентов системы репликации и их функции

Компонент	**Функциональное назначение**
Ядро СУБД источника	Экземпляр ядра СУБД, запущенный на БД источника. Выдает серверу репликации измененные записи. Ядро СУБД должно быть запущено на БД-источника и БД-приемника, которые участвуют в процессе репликации.
Ядро СУБД приемника	Экземпляр ядра СУБД, запущенный на БД приемника. Применяет изменения, полученные от сервера репликации приемника к БД приемника.

Компонент	Функциональное назначение
Ядро СУБД репликации	Экземпляр ядра СУБД, выполняющий прием для временного хранения реплицируемых данных в БД репликации от сервера репликации и последующую передачу этих данных серверу репликации для отправки серверу репликации приемника.
Сервер репликации (`lreplserver`)	На стороне источника: • отслеживает транзакции изменения данных реплицируемых таблиц в БД-источнике; • передает все необходимые для репликации изменения ядру СУБД репликации источника, для сохранения в БД репликации данных репликации; • отправляет данные репликации из БД репликации источника серверу репликации приемника и получает от него ответы, о успешном приеме данных; • позволяет выполнять минимальное администрирование. На стороне приемника: • получает все необходимые для репликации изменения от сервера репликации источника и передает их ядру СУБД репликации приемника, для сохранения их в БД репликации; • передает данные репликации из БД репликации приемника ядру СУБД приемника для внесения изменений в БД приемника; • позволяет выполнять минимальное администрирование.
Сетевой драйвер клиента	Считывает параметры доступа к ядру СУБД репликации и удаленному ЛИНТЕР-серверу (на источнике) из файла сетевой конфигурации (`nodetab`) и передает серверу репликации.

Параметры эффективности СУБД

Одной из главных функций администратора СУБД является повышение производительности СУБД и клиентских приложений. Управление производительностью выполняется путем настройки параметров запуска ядра СУБД и конфигурирования БД с учетом конкретных особенностей клиентских приложений. Ниже рассматриваются параметры запуска СУБД и их влияние на эффективность обработки данных.

Распределение оперативной памяти

Количество оперативной памяти (параметр `pool`), выделяемой ядру СУБД, можно задать двумя способами:

1) в командной строке при запуске ядра СУБД. Память под системные очереди будет выделяться индивидуально для данного сеанса запуска ядра СУБД;
2) при создании БД (или модификации параметров функционирования БД) установить режим автоконфигурации системных очередей ядра СУБД. Память под системные очереди будет выделяться автоматически в зависимости от текущих размеров БД.

После старта ядра СУБД параметры распределения оперативной памяти изменению не подлежат.

Параметр `pool` задает размер пула памяти ядра СУБД в страницах по 4 Кбайт. В пуле размещаются все очереди ядра СУБД: очередь файлов, очередь таблиц, очередь столбцов, очередь страниц, очередь каналов и др.

Параметр `spool` задает размер пула памяти процесса сортировки в страницах. Размер страницы по умолчанию 65535 байт. Для ускорения процесса сортировки можно увеличить размер пула сортировки и перезапустить ядро СУБД. При невозможности разместить сортируемые данные в пул сортировки она будет выполняться с помощью файла сортировки (см. раздел Рабочие файлы).

Параметр `ppool` задает размер пула подсистемы фразового поиска СУБД в страницах по 4 Кбайт.

По умолчанию пул ядра содержит 20000 страниц, пул фразового поиска содержит 200 страниц, пул сортировки содержит: 4000 страниц для размера страницы по умолчанию 4 Кбайт или 12800 страниц для размера страницы по умолчанию 64 Кбайт.

Параметр `pcontcache` задает размер кэша в страницах по 4 Кбайт для контейнера, используемого при создании/модификации фразового индекса.

Параметр `pbvcache` задает размер кэша в страницах по 4 Кбайт для бит-вектора, используемого при создании/модификации фразового индекса.

Таким образом, при старте ядро СУБД ЛИНТЕР занимает оперативную память ОС в размере:

```
(pool+spool+ppool+pcontcache+pbvcache) * 4096 +
  <размер кода + размер статических переменных>
```

байт.

При работе ядро СУБД не делает попыток получения дополнительной памяти.

Для максимального ускорения работы ядра СУБД требуется максимально возможное количество оперативной памяти. Эксперименты показали практически пропорциональную зависимость эффективности работы СУБД ЛИНТЕР от выделенной ей оперативной памяти. Однако в то же время необходимо, чтобы и операционная система располагала некоторым количеством свободной оперативной памяти. Если предполагается, что на данном компьютере, кроме ядра СУБД и его сетевого драйвера сервера, ничего функционировать не будет, то всю оставшуюся память можно предоставить ядру СУБД, увеличивая параметры `pool`, `spool` или `ppool`.

Рекомендуемое соотношение между параметрами `pool` и `spool`: 4 к 1. Но реально максимальная производительность ядра СУБД сильно зависит от требований к обработке данных, предъявляемых клиентскими приложениями. Например, если в БД хранятся большие объемы часто модифицируемых в разных таблицах данных, а сложных поисковых запросов к СУБД не поступает, то есть смысл увеличить размер пула ядра СУБД. И, наоборот, в случае частых сложных поисковых SQL-запросов с группировками и сортировками по нескольким столбцам результирующих выборок имеет смысл увеличить пул процесса сортировки.

В общем случае можно воспользоваться следующим алгоритмом настройки параметров памяти для ядра СУБД:

- выделить максимально возможное количество виртуальной памяти с разделением ее между пулом ядра СУБД и пулом процесса сортировки в соотношении 2 к 1;
- проанализировать реальный поток SQL-запросов к СУБД на обработку данных;
- экспериментально подобрать наилучшее соотношение.

Влияние размеров очередей на производительность СУБД

Память, выделенная ядру СУБД, используется следующим образом:

1) сначала в ней размещаются системные очереди;
2) затем все очереди ядра СУБД;
3) оставшееся место отводится под кэш ядра СУБД.

Примечание
При задании избыточных размеров очередей ядро СУБД не стартует.

Размер кэша может быть до 2 млрд. страниц. Если размер кэша недостаточен для запуска ядра СУБД, то оно не запустится.

Размер очередей (то есть максимально возможное количество хранящихся в них элементов) устанавливается параметрами команды SET утилиты `gendb` (см. документ «Создание и конфигурирование базы данных»), например:

- TABLES – размер очереди дескрипторов таблиц (по умолчанию 100);
- COLUMNS – размер очереди дескрипторов столбцов (по умолчанию 500);
- FILES – размер очереди файлов (по умолчанию 200);
- USERS – размер очереди пользователей (по умолчанию 100);
- CHANNELS – размер очереди каналов (по умолчанию 100).

В СУБД ЛИНТЕР только очередь каналов является статической, остальные очереди динамические.

Очередь является динамической, если при ее переполнении последний элемент очереди удаляется и замещается новым, то есть происходит вытеснение элементов очереди.

Статическая очередь (то есть очередь каналов) переполниться не может. Если СУБД настроена на работу с 5 каналами, то при попытке открыть 6-й канал клиентскому приложению, пославшему команду на открытие канала, будет возвращен соответствующий код завершения.

Размер любой очереди вычисляется как

```
<длина очереди> * <размер элемента очереди>
```

Размер элементов разных очередей колеблется в среднем от 50 до 1500 байт. Для поиска страниц в пуле ядра СУБД ЛИНТЕР используется механизм хеширования. Размер хеш-таблицы по возможности устанавливается равным числу страниц в пуле, но не больше 64 Кбайт. Память таблице выделяется во внутренней памяти ядра перед системными очередями. Хеш-значение вычисляется по номеру страницы и параметрам файла (номер таблицы, тип файла, номер файла).

Поиск свободной страницы в пуле ограничен просмотром не более 1/100 размера пула или 50-ю страницами (если размер пула меньше 5000 страниц), при этом поиск выполняется с учетом уровня страниц. Значение уровня странице присваивается на основе встроенного алгоритма оптимизации работы с пулом ядра СУБД (например, страницы индексных файлов имеют более высокий уровень по сравнению со страницами файла сортировки, поэтому вероятность их использования в качестве свободной страницы, то есть вытеснение, ниже). После завершения поиска выбирается наиболее подходящая из просмотренных страниц.

Очередь таблиц

Очередь таблиц предназначена для хранения в пуле ядра СУБД системного описания таблиц, которые активно используются. В идеале в очереди таблиц должно содержаться описание всех таблиц БД. Если это невозможно, то имеет смысл проанализировать поток запросов к БД из клиентских приложений и определить количество и параметры наиболее часто используемых в SQL-запросах таблиц (усреднённый размер очереди).

В случае, когда размер очереди таблиц близок к этому усредненному значению, работа клиентского приложения будет, скорее всего, устраивать пользователя БД, и только в редких случаях могут возникать небольшие задержки. Таким образом, существует зависящая от клиентских приложений граница («пороговое значение»), характеризующее очередь как медленную или быструю. Если текущий размер очереди таблиц превышает эту границы, то клиентские приложения работают достаточно быстро, и дальнейшее расширение очереди таблиц практически не приведет к ускорению их работы.

При размере очереди таблиц меньше «порогового значения» (медленная очередь) скорость работы СУБД ощутимо снижается.

Очередь столбцов таблиц

Очередь столбцов таблиц предназначена для хранения в пуле ядра СУБД системного описания столбцов обрабатываемых таблиц.

 © Архитектура СУБД. АО НПП «РЕЛЭКС», 1990-2026

Очередь столбцов является наиболее динамичной, поэтому ее размер существенно влияет на скорость обработки данных СУБД.

Если очередь столбцов слишком мала, то первые замедления в работе СУБД возникнут уже при трансляции SQL-запроса, поскольку транслятор проверяет корректность использования каждого столбца таблицы в транслируемом SQL-запросе путем обращения к СУБД за полным его описанием.

Ядро СУБД сначала ищет описания столбцов среди тех описаний, которые находятся в текущий момент в очереди столбцов. При этом для поиска находящегося в очереди элемента обычно нужно просмотреть в среднем половину очереди, а для того, чтобы установить отсутствие элемента, необходимо просмотреть всю очередь. Если дескриптор искомого столбца в очереди не найден, ядро СУБД ищет описание этого столбца в системной таблице описания атрибутов `$$$ATTRI`, где находятся описания всех столбцов всех таблиц БД. Если таблица `$$$ATTRI` не индексирована (по имени столбца), то поиск осуществляется простым перебором всех находящихся в нем описаний. Очевидно, это длительный процесс, особенно когда суммарное количество столбцов в БД довольно велико (средняя БД включает до 1000 столбцов).

Дескриптор найденного в таблице `$$$ATTRI` столбца включается в очередь столбцов, при этом из очереди столбцов вытесняется какой-то элемент (вполне возможно, что в этом случае понадобится еще и записать в БД информацию о вытесняемом столбце, если она изменилась). На этапе трансляции этот длительный (но разовый!) процесс обработки выполняется для каждого из использованных в SQL-запросе, но отсутствующих в очереди, столбцов.

Если бы процесс обработки SQL-запроса был непрерывным (или СУБД функционировала бы в однопользовательском режиме), то не возникало бы связанных с очередью замедления обработки, так как все необходимые столбцы уже находились бы в очереди и не вытеснялись бы из нее до окончания обработки запроса. Однако в многопользовательском режиме возникает потребность в параллельном выполнении нескольких SQL-запросов. При этом СУБД регулярно переключается на обработку очередного SQL-запроса, уделяя каждому из них квант обработки. Если размер очереди столбцов недостаточен, то описания столбцов текущего SQL-запроса могут вытеснить из очереди описания столбцов предыдущего SQL-запроса, и этот процесс будет повторяться для всех параллельно выполняемых SQL-запросов. Чем меньше квант обработки (чаще переключение между запросами), тем сильнее замедляется выполнение запросов.

Рекомендуется размер очереди столбцов задавать как можно больше. В идеале очередь столбцов должна вмещать описание всех столбцов всех таблиц БД.

Очередь файлов

При работе с таблицей БД ядро СУБД должно открыть файлы этой таблицы. Дескриптор открытого файла помещается в *очередь файлов*, вытесняя при этом, в случае необходимости, дескриптор другого файла с предварительной синхронизацией страниц файла в оперативной памяти и на диске.

Каждая таблица БД это минимум два файла: файл данных и файл индексов.

Данные/индексы могут размещаться в нескольких файлах (до 63). Таким образом, число файлов таблицы может быть до 63*2. При работе с таблицей ее файлы открываются только при необходимости. Чтобы начать работу с таблицей, СУБД открывает первый файл данных и первый файл индексов. Чаще всего пользователи БД размещают таблицу в двух файлах (один файл данных и один индексный файл). Разнесение таблицы более

чем в два файла необходимо в редких случаях. Ниже приведен анализ эффективности работы СУБД в зависимости от размера очереди файлов.

Желательно, чтобы при работе с таблицей описания всех её файлов находились в очереди файлов. Таким образом, размер очереди файлов должен быть, как минимум, в 2 раза больше размера очереди таблиц, чтобы вместить описания одного индексного файла и одного файла данных каждой таблицы.

Если некоторые таблицы размещены более чем в двух файлах, то дополнительное увеличение очереди файлов не приводит к ощутимому росту эффективности клиентского приложения. Дело в том, что только первый индексный файл используется на протяжении всей обработки SQL-запроса, все остальные индексные файлы используются лишь для быстрого поиска записей по индексированному атрибуту. Команда «выдать следующую запись» уже не затрагивает остальные индексные файлы. При обработке SQL-запроса записи просматриваются порциями. Порядок просмотра почти совпадает с порядком поступления записей.

СУБД упаковывает данные, и добавляемая упакованная запись помещается на первое свободное место, достаточное для ее размещения. Эти свободные места образуются практически случайно, однако замечено, что чем ближе процент сжатия к значению 100 %, тем менее случайным становится процесс появления свободных мест. Если СУБД считает информацию несжимаемой (процент сжатия 100%), то порядок расположения записей в файле будет почти совпадать с порядком их поступления на обработку. Таким образом, если очередная запись выборки данных извлекается из какого-либо файла данных, то следующие записи, вероятнее всего, будут в том же файле. Смена файлов в очереди файлов будет последовательной, что сделает работу СУБД в многопользовательском режиме более эффективной.

Если процент сжатия близок к 100%, то увеличение очереди файлов вдвое по сравнению с очередью таблиц почти не даст повышения эффективности. Чем меньше процент сжатия, тем больше вероятность переключения СУБД с одного файла на другой. Вполне возможно, что при маленьком проценте сжатия будет недостаточно очереди файлов вдвое большей, чем очередь таблиц (могут начаться ощутимые замедления). Однако это происходит только в том случае, когда таблицы состоят более чем из двух файлов.

Характеристики физической структуры БД

Физическая структура каждой таблицы имеет следующие характеристики, которые влияют на эффективность работы СУБД:

- число и размеры файлов таблицы;
- мощность множества системных номеров (MAXROWID);
- упаковка пользовательских данных;
- индексы таблицы.

Характеристики файлов таблицы

Для того чтобы разбить таблицу на файлы (а затем файлы разместить на разных внешних устройствах с целью повысить скорость работы СУБД при обращении к данным такой таблицы), требуется глубокое знание операционной среды, особенностей ее работы с файлами, поэтому в большинстве случаев достаточно создавать таблицу из одного файла данных и одного индексного файла. Однако в некоторых случаях разбиение файла данных или индексного файла на несколько файлов может привести к увеличению эффективности работы СУБД.

 © Архитектура СУБД. АО НПП «РЕЛЭКС», 1990-2026

Решение вопроса о целесообразности такого разбиения опирается на механизм использования файлов таблицы ядром СУБД. При загрузке этот механизм схематично выглядит следующим образом (для файлов данных):

1) все новые данные поступают только в первый файл до тех пор, пока будут исчерпаны возможности его естественного расширения (исчерпается файловое пространство на диске или для дальнейшего расширения потребуется расширение битовой карты файла);
2) только после этого СУБД начинает помещать данные в следующий файл (опять же, до тех пор, пока будут исчерпаны возможности его естественного расширения);
3) после этого (если есть возможность) СУБД расширит битовую карту первого файла и возобновит запись в первый файл, расширяя его (переход на пункт 1 и т.д.).

Из приведенной схемы следует, что пользователь БД не может прямо управлять размещением своей информации в файле данных (а только косвенно и достаточно грубо). Например, в случае, когда объем данных планируется настолько большим, что для их размещения требуется файл с двумя страницами битовой карты, то данные можно разместить и в двух файлах с одной страницей битовой карты в каждом. При этом первая часть информации, вносимой в таблицу, обязательно попадет в первый файл, вторая часть во второй файл. Если при этом файл загрузки информации отсортирован по какому-то полю, то записи с меньшими значениями этого поля попадут в первый файл, а записи с большими значениями во второй.

В этом случае SQL-запросы, использующие обе половины данных (из обеих частей файла), будут работать быстрее, если разместить файлы данных на разных дисковых носителях.

Если же отсортировать информацию в порядке убывания частоты использования (предположим, что пользователь обладает подобной статистикой), то в первый файл попадет наиболее часто используемая информация. Многие операционные системы позволяют создавать виртуальные диски в оперативной памяти (или на дисках с большей скоростью доступа). Именно на них рекомендуется расположить файл с часто используемой информацией.

Необходимость в нескольких файлах данных может возникнуть еще и при переполнении дисков (аналогично при ограничениях квоты использования дискового пространства). В этом случае гибкая многофайловая система, принятая в СУБД ЛИНТЕР, позволит более полно использовать имеющиеся ресурсы.

Работа СУБД с индексными файлами управляется пользователем. Он указывает, в каком из индексных файлов расположить тот или иной индекс. При этом весь индекс (B*-дерево) будет располагаться в указанном индексном файле.

Целесообразность построения индексов в нескольких отдельных файлах опять же заключается в том, чтобы расположить часто используемые индексы на быстрых носителях. Даже простое разнесение индексов по разным файлам на независимые физические носители данных может дать выигрыш.

СУБД ЛИНТЕР в определенных пределах может расширять файлы таблицы, что делается только по необходимости, то есть в случае, если для записи какой-либо информации в файле больше нет места.

Таким образом, если заранее нельзя указать размер файлов, достаточный для размещения данных, то можно остановиться на минимальных размерах. СУБД увеличит файлы именно до таких пределов, чтобы поместить туда необходимую информацию.

Однако не следует слишком часто пользоваться «расширяемостью» файлов, т.к. этот режим работы с файлами наименее эффективен.

СУБД ЛИНТЕР максимально использует возможности файловой системы ОС. Файловая система при расширении файла обычно присоединяет к нему первую свободную область нужного размера, поэтому использование расширяемых файлов приведет к фрагментации диска и к замедлению некоторых операций. Используя расширяемые файлы, можно минимизировать размер файлов (несколько проиграв в эффективности).

Множество системных номеров

При создании таблицы пользователь (явно или неявно) задает параметр MAXROWID, определяющий максимальный системный номер записи таблицы и влияющий на эффективность работы СУБД ЛИНТЕР. В каждый момент времени множество системных номеров записей (ROWID) любой таблицы разделено на три подмножества:

1) подмножество ROWID, занятых под номера записей;
2) подмножество ROWID удаленных записей;
3) подмножество еще неиспользованных (свободных) ROWID.

Стратегия использования этих подмножеств следующая:

- при удалении записи ее ROWID переходит в подмножество удаленных номеров;
- при добавлении записи ей присваивается первый неиспользованный ROWID, если подмножество неиспользованных ROWID не пусто, в противном случае первый ROWID из подмножества удаленных записей.

При создании таблицы все ROWID являются неиспользованными.

Новой записи присваивается первый (наименьший) неиспользованный ROWID. Это будет происходить до полного исчерпания подмножества неиспользованных ROWID, после чего номер для записи СУБД будет искать среди ROWID удаленных записей, что делает процедуру добавления менее эффективной.

Таким образом, пользователь при работе с таблицей может с какого-то момента обнаружить небольшие замедления при загрузке данных, что будет означать начало переиспользования ROWID удаленных записей. При переполнении таблицы (после того, как подмножества неиспользуемых и удаленных ROWID стали пустыми), СУБД приступает к расширению конвертера (на одну страницу), то есть к увеличению параметра MAXROWID (на 1022 номера).

При наличии большого числа удаленных записей рекомендуется выполнять сжатие таблицы (с помощью специального SQL-запроса). При сжатии таблицы СУБД производит перенумерацию записей и все ROWID удаленных записей переводит в конец подмножества неиспользованных ROWID. Следовательно, число номеров (параметр MAXROWID) выгоднее задавать с некоторым «запасом», чтобы приходилось реже запускать процедуру сжатия. Однако этот запас не должен быть слишком большим, иначе СУБД придется использовать длинные бит-векторы ответов и это может снизить общую эффективность работы СУБД.

В этой связи можно дать следующую рекомендацию: число MAXROWID должно быть в полтора-два раза больше планируемого среднего числа записей, одновременно находящихся в таблице.

Упаковка пользовательских данных

СУБД ЛИНТЕР хранит записи таблиц в файлах данных таблиц в упакованном виде, то есть все правосторонние концевые пробелы символьных данных и двоичные нули байтовых данных усекаются. При поиске свободных страниц файла данных в процессе добавления новой записи СУБД проверяет наличие в ней свободного места не для всей декларированной записи, а для её упакованной версии.

Определять, что страница файла данных свободна для добавления в неё новой записи, можно по двум критериям: количеству свободных байтов в ней (например, 459) или по проценту свободного места в ней (например, 20%). СУБД ЛИНТЕР использует второй критерий. Если свободного места на странице меньше, чем указанный владельцем создаваемой таблицы размер упакованных записей этой таблицы (параметр `PCTFILL` в SQL-операторе создания таблицы, то есть меньше, чем на одну упакованную запись), то СУБД помечает страницу как занятую и запрещает добавление в неё новых записей до тех пор, пока свободного места на странице не станет больше, чем задано параметром `PCTFREE` (значение параметра задается владельцем создаваемой таблицы в SQL-операторе создания таблицы и указывает, при каком проценте неиспользуемого в странице места она считается свободной). Значение параметра `PCTFREE` зависит от средней длины упакованной записи.

Пример вычисления параметров PCTFILL и PCTFREE см. в документе «Рекомендации по настройке СУБД ЛИНТЕР».

Кэш SQL-транслятора

Кэширование – это способ повышения скорости обработки данных, хранящихся на внешних носителях.

Кэш SQL-транслятора состоит из набора записей в оперативной памяти. Каждая запись кэша является копией соответствующего элемента данных в БД. Все записи кэша имеют идентификатор, устанавливающий соответствие между элементами данных в кэше и их копиями в БД.

Если нужные данные не найдены в кэше, то они извлекаются из БД в кэш, и становятся доступным для последующих обращений к ним.

Память кэша ограничена в объёме, поэтому при добавлении в него новых данных в случае отсутствия в кэше свободного места записи, к которым было мало обращений, вытесняются из кэша и заменяются добавляемыми записями.

В случае модификации элементов данных в кэше синхронно выполняется их обновление и в БД, возможно, с некоторой задержкой.

SQL-транслятор выполняет кэширование информации:

- о пользователях (владельцах объектов) БД;
- о столбцах SQL-запросов;
- о хранимых процедурах SQL-запросов;
- о параметрах хранимых процедур SQL-запросов;
- о кодовых страницах;
- о таблицах SQL-запросов.

Размеры кэшей SQL-транслятора задаются (при необходимости) при создании БД и могут быть изменены в процессе её эксплуатации. Если размеры кэшей не заданы, используются значения по умолчанию.



Примечание

Описанные выше очереди ядра СУБД (очереди файлов, таблиц, столбцов и др.), по сути, также являются кэшируемыми данными. Ведение двух, частично дублирующих друг друга кэшей, вызвано тем, что SQL-транслятор может запускаться в ОС как отдельный процесс. В этом случае ведение собственного кэша ускоряет трансляцию SQL-запросов и снижает нагрузку на ядро СУБД.

Совместимость продуктов

Версия СУБД ЛИНТЕР СТАНДАРТ не может работать с БД, созданной версией СУБД ЛИНТЕР БАСТИОН, если было выполнено одно из перечисленных действий:

- включен аудит;
- создан уровень мандатного доступа (существует запись в системной таблице `$$$LEVEL`);
- создана группа мандатного доступа (существует запись в системной таблице `$$$GROUP`).

Если ни одно из этих действий не выполнялось, то с БД, созданной версией ЛИНТЕР БАСТИОН, версия ЛИНТЕР СТАНДАРТ будет работать корректно.

Версия ЛИНТЕР СТАНДАРТ поддерживает все операции версии ЛИНТЕР БАСТИОН, за исключением:

- нельзя подавать любые команды аудита;
- нельзя создавать уровни мандатного доступа;
- нельзя создавать группы мандатного доступа.

Версия ЛИНТЕР БАСТИОН поддерживает выполнение всех операций ЛИНТЕР СТАНДАРТ, без исключений.

Базу данных СУБД ЛИНТЕР можно переносить на другую ОС и платформу, если у нее такой же порядок байтов. Не допустим перенос БД между платформами с разным порядком байтов - прямым порядком байтов (little-endian) и обратным порядком байтов (big-endian).