СИСТЕМА УПРАВЛЕНИЯ БАЗАМИ ДАННЫХ

ЛИНТЕР®

ЛИНТЕР БАСТИОН

ЛИНТЕР СТАНДАРТ

Рекомендации по настройке СУБД ЛИНТЕР

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 2
Назначение документа 2
Для кого предназначен документ 2
Дополнительные документы 2
Настройка параметров БД 3
Конфигурация размера очередей 3
Настройка размера используемой оперативной памяти 7
Автоматическое конфигурирование 8
Ручное конфигурирование 9
Настройка количества процессов сортировки 10
Оптимизация размеров рабочих файлов СУБД 11
Ручное управление размерами рабочих файлов 12
Автоматическое управление размерами рабочих файлов 14
Предварительная разметка дискового пространства для таблицы 14
Особенности работы с широкими записями и BLOB-файлами 15
Настройка параметров системного журнала 16
Рекомендации по настройке СУБД для работы с большими объемами данных 18
Оптимизация размеров табличных файлов БД 18
Оптимизация работы ядра СУБД 20
Пул ядра СУБД 20
Пул сортировки 21
Загрузка большого объема данных 21
Массовое добавление данных в пустую таблицу 21
Массовое добавление данных в рабочую таблицу 22
Управление ссылочной целостностью загруженных данных 23
Манипулирование данными большого объема 23
Использование таблиц в памяти 25
Расчет размера файла сортировки 28
Оценка задержки в выполнении стандартного запроса 29
Кэш SQL-транслятора 32
Оптимизация SQL-запросов 33
Внутренний уровень оптимизации 33
Замена оператора сравнения на противоположный 33
Перенос операции на другой уровень 34
Преобразование в эквивалентное условие 34
Преобразование OR в IN 34
Слияние нескольких предикатов в один 35
Изменение порядка обработки условий 35
Раскрытие подзапросов 36
Исключение лишних сортировок 37
Преобразование EXISTS в IN 38
Использование транзитивности 38
Замена выражений 39
Выбор оптимального пути доступа к данным 39
Стратегия доступа к данным при вычислении однопеременных предикатов 39
Стратегия доступа к данным при вычислении многопеременных предикатов 40
Внешний уровень оптимизации 41
Использование индексов 41
Подсказки 41

Предисловие

Назначение документа

Документ содержит рекомендации по настройке СУБД ЛИНТЕР для улучшения производительности.

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

Для кого предназначен документ

Документ предназначен для разработчиков и проектировщиков баз данных СУБД ЛИНТЕР.

Дополнительные документы

- Архитектура СУБД
- Создание и конфигурирование базы данных
- Запуск и останов СУБД ЛИНТЕР в среде ОС Windows
- Запуск и останов СУБД ЛИНТЕР в среде ОС Linux
- Сетевой администратор
- Справочник по SQL
- Интерфейс нижнего уровня
- Импорт данных
- Процедурный язык

Настройка параметров БД

Управление производительностью выполняется путем конфигурирования БД, ключами запуска ядра СУБД и SQL-командами с учетом конкретных особенностей клиентских приложений.

Наиболее важными параметрами, влияющими на эффективность работы СУБД являются:

- размеры очередей;
- размеры используемой оперативной памяти;
- количество процессов сортировки;
- размеры рабочих файлов;
- параметры разметки дискового пространства для таблицы;
- параметры системного журнала;
- оптимизация SQL-запросов.

Конфигурация размера очередей

СУБД ЛИНТЕР для своей работы использует очереди таблиц, файлов, колонок и пользователей. Размеры очередей, задаваемые по умолчанию при создании БД, достаточны для работы, но не являются оптимальными. Поэтому для улучшения производительности рекомендуется настроить параметры очередей вручную или воспользоваться режимом автоконфигурирования.

Настроить параметры очередей можно в любое время после создания БД при остановленном ядре СУБД с помощью утилиты `gendb` или приложения «Администратор СУБД ЛИНТЕР».

Значения параметров размера очередей зависят от размеров создаваемой БД.

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, т.е. происходит вытеснение элементов очереди.

Статическая очередь (т.е. 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 СУБД 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, т.е. 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.

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, т.к. 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 («дырки») образуются практически случайно, однако замечено, что чем ближе процентное отношение среднего размера упакованной записи к среднему размеру неупакованной к значению 100 %, тем менее случайным становится процесс появления «дырок». Если СУБД считает информацию несжимаемой (процентное отношение среднего

размера упакованной записи к среднему размеру неупакованной 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 процентное отношение среднего размера упакованной записи к среднему размеру неупакованной близко к 100%, то увеличение очереди файлов вдвое по сравнению с очередью таблиц почти не даст повышения эффективности. Чем меньше процентное отношение среднего размера упакованной записи к среднему размеру неупакованной, тем больше вероятность переключения СУБД с одного файла на другой. Вполне возможно, что при маленьком процентном отношении среднего размера упакованной записи к среднему размеру неупакованной 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.

Грубая оценка для требуемого размера очередей следующая:

- таблицы (tables): размер очереди должен вмещать все таблицы БД плюс дополнительное количество для временных таблиц;
- столбцы (columns): размер очереди должен вмещать в себя все колонки всех таблиц (из предыдущего пункта), плюс элементы составных индексов и именованных одноколоночных индексов;
- файлы (files): размер очереди должен вмещать в себя по два файла на каждую таблицу БД плюс 10 файлов под служебные нужды;
- пользователи (users): размер очереди должен вмещать всех пользователей системы и все назначения привилегий.

Пример для утилиты gendb (в примере использованы произвольные числовые значения)

```
set database directory <путь к БД>;
set tables 100;
set columns 1000;
set files 200;
set users 200;
```

Установка очередей через интерфейс приложения «Администратор СУБД ЛИНТЕР»

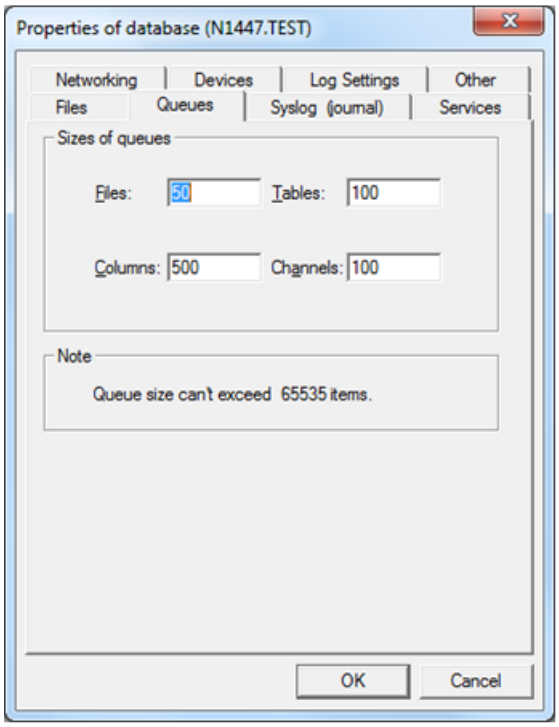


Рисунок 1. Установка размера очередей

Более подробное описание можно найти в документации:

- «Архитектура СУБД»;
- «Создание и конфигурирование базы данных»;
- «Сетевой администратор».

Настройка размера используемой оперативной памяти

СУБД ЛИНТЕР использует оперативную память компьютера для:

- пула ядра СУБД;
- пула сортировки;
- пула фразового поиска;
- пула таблиц «в памяти» (см. раздел Использование таблиц в памяти).

СУБД ЛИНТЕР можно настроить на работу в условиях ограниченных ресурсов (оперативной памяти). Параметры, задаваемые по умолчанию при создании БД, достаточны для работы, но не являются оптимальными для конкретной прикладной задачи.

В процессе своей работы СУБД ЛИНТЕР активно использует оперативную память для кэширования и сортировки информации. При этом используется только то количество оперативной памяти, которое было выделено при запуске ядра СУБД.

Для наибольшего ускорения работы СУБД требуется максимально возможное количество оперативной памяти.

Объем оперативной памяти задается в страницах размером 4 Кбайта.

Для улучшения производительности рекомендуется воспользоваться режимом автоконфигурирования или настроить параметры вручную.

Автоматическое конфигурирование

Рекомендуется установить режим автоконфигурирования сразу после создания БД. Это можно сделать с помощью утилиты `gendb` командой:

```
set autoconfig on;
```

или в настройках БД из графических средств управления (рис. 2).

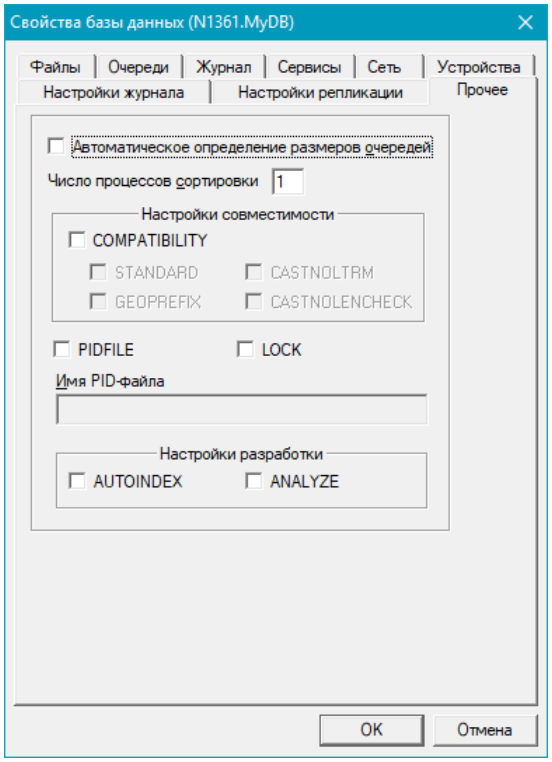


Рисунок 2. Автоматическое определение размеров очередей

В этом режиме память под кэши (очереди) таблиц, файлов, столбцов и пользователей выделяется автоматически в зависимости от текущих размеров соответствующих системных таблиц.

В режиме автоконфигурирования ядру СУБД выделяется четверть доступной оперативной памяти компьютера. Каждому процессу сортировки – эквивалент четверти памяти ядра СУБД.

Отключение автоконфигурирования производится в утилите `gendb` командой:

```
set autoconfig off;
```

или в настройках БД из графических средств управления (рис. 2).

Более подробное описание можно найти в документации:

- «Архитектура СУБД»;
- «Создание и конфигурирование базы данных».

Примечание
Значения используемой оперативной памяти, задаваемые ключами запуска ядра СУБД /POOL и /SPOOL имеют больший приоритет, чем значение установленное режимом автоконфигурации.

Ручное конфигурирование

В режиме ручного конфигурирования объем оперативной памяти, выделяемой ядру СУБД, можно задать в командной строке запуска ядра СУБД следующими ключами:

- /POOL – задает объем памяти для ядра СУБД (в страницах по 4 Кбайт);
- /SPOOL – задает объем памяти для процессов сортировки (в страницах по 4 Кбайт). Так как процессов сортировки может быть несколько, то значение ключа /SPOOL применяется к каждому из процессов сортировки;

Значения POOL и SPOOL можно задать как в команде запуска ядра СУБД, так и в утилите «Администратор СУБД ЛИНТЕР» при запуске ядра в окне свойств БД (рис. 3).

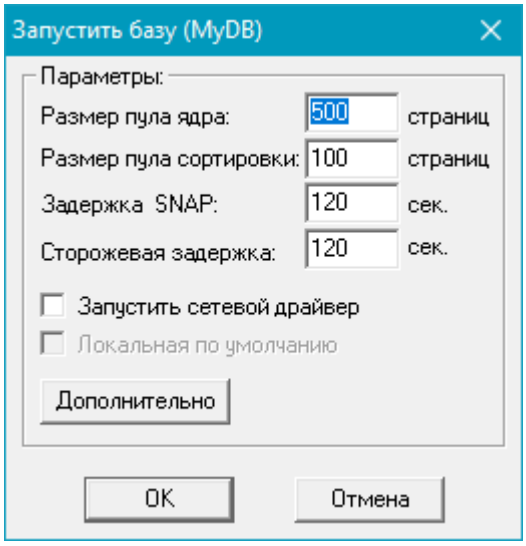


Рисунок 3. Запуск утилиты

- /PPOOL – задает объем памяти для подсистемы полнотекстового поиска (см. документ Полнотекстовый поиск в базе данных), в случае если клиентское приложение использует полнотекстовый поиск, настоятельно рекомендуется выделять память вручную, так как работа подсистемы полнотекстового поиска с параметрами по умолчанию может быть медленной;
- /INMEMPOOL – задает объем памяти для хранения страниц таблиц «в памяти» (предварительно СУБД должна быть настроена для работы с таблицами «в памяти» с помощью утилиты gendb);
- /PCONTCACHE - задает размер кэша в страницах по 4 Кбайт для контейнера, используемого при создании/модификации фразового индекса;
- /PBVCACHE - задает размер кэша в страницах по 4 Кбайт для бит-вектора, используемого при создании/модификации фразового индекса.

Пример установки значений в команде запуска ядра

```
linter64 /POOL=250000 /SPOOL=100000
```

Пример установки значений через интерфейс приложения «Администратор СУБД ЛИНТЕР»

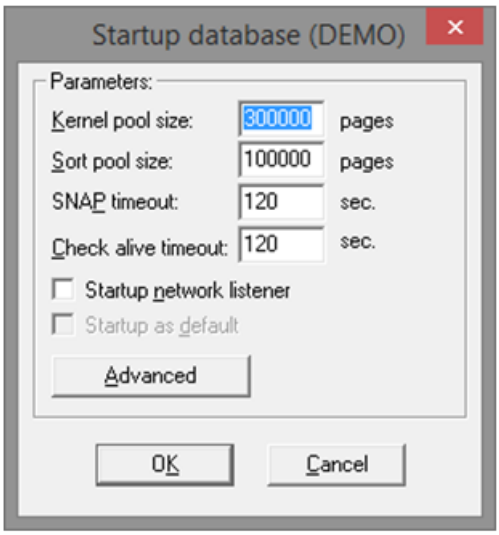


Рисунок 4. Настройка размера пулов

Оперативная память, используемая подсистемой фразового поиска, задается следующими ключами команды запуска ядра СУБД:

- ключ `/PPOOL` задает размер пула подсистемы фразового поиска;
- ключ `/PCONTCACHE` задает размер кэша в страницах по 4 Кбайт для контейнера, используемого при создании/модификации фразового индекса;
- ключ `/PBVCACHE` задает размер кэша в страницах по 4 Кбайт для бит-вектора, используемого при создании/модификации фразового индекса.

Более подробное описание можно найти в документации:

- «Архитектура СУБД»;
- «Создание и конфигурирование базы данных»;
- «Запуск и останов СУБД ЛИНТЕР в среде ОС Windows»;
- «Запуск и останов СУБД ЛИНТЕР в среде ОС Linux»;
- «Сетевой администратор».

Настройка количества процессов сортировки

Сортировка данных выполняется отдельным (от ядра СУБД) процессом сортировки.

В процессе своей работы СУБД ЛИНТЕР может использовать несколько процессов сортировки. Так как по умолчанию работает только один процесс сортировки, то процесс сортировки будет предоставляться поочередно каждому из них в соответствии с приоритетом SQL-запроса на установленный квант времени сортировки.

Если предполагается одновременная работа с БД большого количества пользователей, выполняющих запросы, требующие сортировки больших объемов данных, то рекомендуется увеличить число процессов сортировки.

Обычно нет смысла устанавливать количество процессов сортировки больше 4. Объем оперативной памяти, задаваемый параметром `SPOOL`, выделяется каждому процессу сортировки.

Пример для утилиты gendb

```
SET SYSSRT COUNT 2;
```

Пример для утилиты «Администратор СУБД ЛИНТЕР»

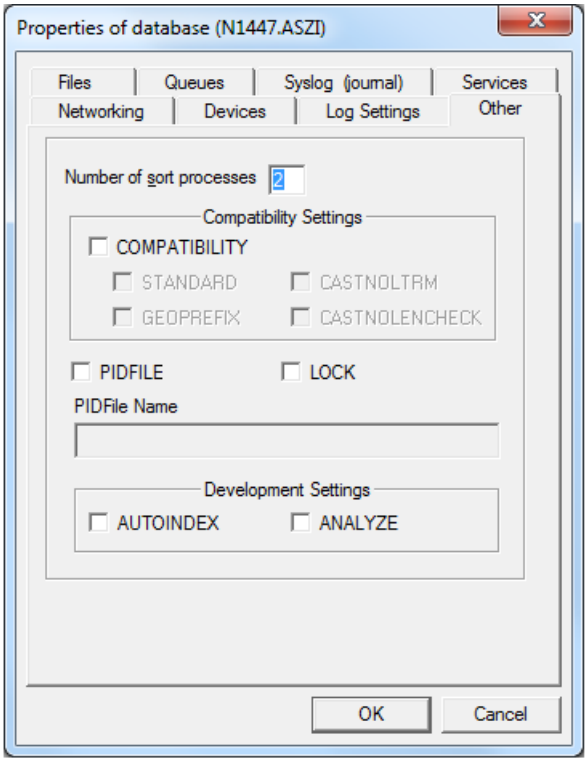


Рисунок 5. Установка количества процессов сортировки в окне свойств БД

Оптимизация размеров рабочих файлов СУБД

В процессе обработки SQL-запросов ядро СУБД использует следующие рабочие файлы:

- `SYSSRT`: файл сортировки результатов выборки данных.

  СУБД ЛИНТЕР выполняет сортировку данных при обработке SQL-запросов с конструкциями:
  - группировка данных (GROUP BY-спецификация);
  - сортировка данных (ORDER BY-спецификация);
  - аналитические функции (OVER-спецификация);
  - группировка значений агрегатных функций (WITHIN GROUP-спецификация);
  - SQL-запросы создания условий целостности таблицы (индексов, ключей).

  При сортировке данных вначале используется выделенная по умолчанию или заданная ключом `/SPOOL` в команде запуска ядра СУБД оперативная память компьютера. Если выделенной оперативной памяти недостаточно для сортировки всего объема данных обрабатываемого SQL-запроса, то часть сортируемых данных будет размещена в файле сортировки `SYSSRT` (файл `1.51`). Начальный размер файла сортировки равен 2 страницы, максимальный размер 500000 страниц.
- `SYSWBV`: файл бит-векторов.

При выполнении SQL-запросов выборки данных для хранения промежуточных результатов используется файл бит-векторов (содержит системную информацию о выбранных в SQL-запросах записях) SYSWBV (файл 1.31). Начальный размер файла бит-векторов равен 16 страниц, максимальный размер 500000 страниц.

- SYSWRK: рабочий файл.

  При выполнении SQL-запросов выборки данных для хранения промежуточные результатов обработки SQL-запроса используется рабочий файл SYSWRK (файл 1.41). Начальный размер файла равен 4 страницам, максимальный размер 500000 страниц.

При создании БД устанавливаются начальный и максимальный размер указанных файлов. Если начальных размеров файлов недостаточно, то СУБД автоматически расширяет их до необходимых для обработки SQL-запросов размеров, но не более установленного максимального значения и делает это она порциями по EXTSIZE <размер> страниц (расширение выполняется на 1/32 от текущего размера файла с выравниванием до кратного 64 Кбайт, но не больше чем на 32 Мбайт), что приводит к фрагментации файлов и требует дополнительного времени на работу с такими файлами по сравнению с файлами, которые сразу создаются в виде непрерывных файлов заданных размеров. При достижении максимального установленного размера будет выдан соответствующий код завершения.

Управлять размерами рабочих файлов можно двумя способами:

1) ручным (пользователь на основании некоторой информации задает требуемые начальные размеры рабочих файлов при создании (конфигурировании) БД);
2) автоматическим (СУБД автоматически расширяет вспомогательные файлы до нужных для обработки SQL-запроса размеров по заданному пользователем правилу).

Ручное управление размерами рабочих файлов

Оценка требуемых размеров рабочих файлов

Для оценки требуемых к конкретной задаче размеров рабочих файлов рекомендуется выполнить следующие действия:

1) запустить ядро СУБД со стандартными (по умолчанию) начальными размерами рабочих файлов;
2) выполнить типичные SQL-запросы, связанные с обработкой данных большого объема;
3) завершить работу ядра СУБД;
4) проанализировать в файле linter.out сообщения вида:

```
Attention: file SYSSRT truncated from 107856 to 2 pages
Attention: file SYSWRK truncated from 16 to 4 pages
Attention: file SYSWBV truncated from 385024 to 16 pages
```

Приведенные сообщения информируют, что в процессе обработки данных рабочие файлы были расширены ядром СУБД до необходимых для выполнения SQL-запроса размеров и при завершении работы ядра СУБД были урезаны до начальных размеров.

Рассчитать требуемые размеры рабочих файлов, увеличенные на 20% и округлённые в большую сторону.

Примечание
Начальное значение рабочих файлов не может превышать максимальное. Если рассчитанное значение получится больше установленного максимального значения, то необходимо вначале установить максимальное значение и затем установить начальное значение.

5) установить размеры файлов можно следующим способом:

С помощью утилиты `gendb`.

Для вышеприведенных размеров, указанных в файле `linter.out`:

```
set SY00=<путь к БД>
gendb
SET SYSSRT SIZE 130000;
SET SYSWBV SIZE 20;
SET SYSWRK SIZE 470000;
exit;
```

С помощью утилиты «Администратор СУБД ЛИНТЕР».

Коррекцию размеров рабочих файлов выполнить также при остановленном ядре СУБД в окне свойств БД во вкладке `Файлы` (рис. 6).

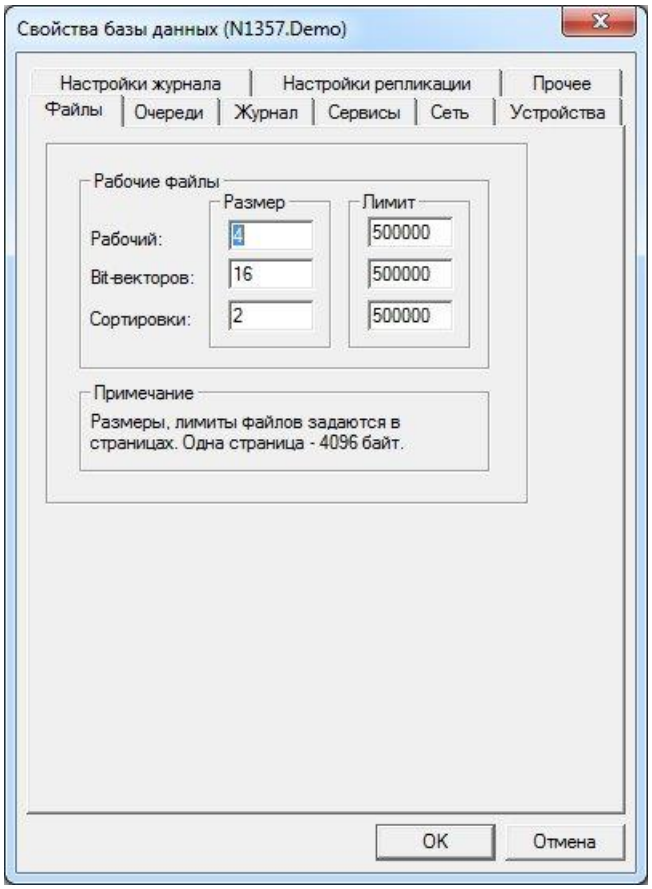


Рисунок 6. Установка размеров рабочих файлов

Автоматическое управление размерами рабочих файлов

Рекомендуется установить примерные требуемые максимальные значения рабочих файлов и увеличить значение параметра расширения EXTSIZE. Например,

```
set SY00=<путь к БД>
gendb
SET SYSSRT LIMIT 10000000;
SET SYSWBV LIMIT 10000000;
SET SYSWRK LIMIT 10000000;
SET EXTSIZE 256;
exit;
```

Предварительная разметка дискового пространства для таблицы

Для ускорения вставки в таблицу большого количества данных и повышения эффективности работы при изменении данных необходимо разметить пространство заранее (при создании таблицы) при помощи параметров MAXROWID, MAXROW, EXTSIZE, PCTFILL, PCTFREE, BLOBPCT. В этом случае при вставке данных не будет происходить динамического увеличения занимаемого таблицей пространства (до тех пор, пока вставляемые данные будут "умещаться" в установленных пределах), что положительно скажется на производительности. По умолчанию установлены усредненные параметры, достаточные для работы.

Значения параметров:

- MAXROWID задает максимальное планируемое количество номеров записей (ROWID) для создаваемой таблицы. По умолчанию значение этого параметра равно 1022;
- MAXROW задает максимальное планируемое число записей, которые будут одновременно храниться в таблице. По соображениям эффективности не следует задавать значение параметра MAXROW большим значения MAXROWID. Если значение MAXROW не задано, то оно также устанавливается равным значению MAXROWID. При задании этого параметра в БД происходит выделение пространства для таблицы под указанное количество записей;
- EXTSIZE задает количество страниц для расширения файлов таблиц и рабочих файлов, при превышении установленных начальных размеров. Значение по умолчанию 0 (расширение выполняется на 1/32 от текущего размера файла с выравниванием до кратного 64 Кбайт, но не больше чем на 32 Мбайт).

  Недостаточное в конкретной задаче значение параметра EXTSIZE (медленное расширение файлов таблиц, рабочих файлов по одной странице) может привести к резкому падению скорости загрузки данных, индексов, BLOB-значений или падению скорости выполнения SQL-запросов, поэтому рекомендуется установить требуемое в конкретной задаче значение.

  Пример

```
SET EXTSIZE 256;
```

- PCTFILL задает ожидаемое отношение среднего размера упакованной записи к среднему размеру неупакованной (в процентах). При выборе значения PCTFILL

предполагается, что количество упакованных записей с длиной меньше, чем PCTFILL, будет незначительным.

Это целое положительное число, не превышающее 100. По умолчанию оно равно 100%. Параметр влияет на размечаемое пространство вместе с параметром MAXROWID.

Значение параметра PCTFILL может быть критичным тогда, когда в таблице есть строковые поля большого размера, обычно заполняемые значительно более короткими значениями.

Оптимальный процент ожидаемого отношения среднего размера упакованной записи к среднему размеру неупакованной находится в пределах от 75% до 95%. В случае, если пользователь затрудняется указать значение PCTFILL, рекомендуется при создании таблицы задать его равным 90%.

- PCTFREE определяет для файла данных порог свободного места страницы в процентах. Страницы, заполненные меньше указанного порога, считаются свободными для добавления информации. В результате при добавлении новых данных в эту страницу можно добавить сразу несколько записей, прежде чем она будет помечена как заполненная, что в конечном итоге приводит к снижению фрагментации данных в файлах таблицы;
- BLOBPCT задает процент заполнения BLOB-страниц создаваемой таблицы. BLOB-страница, заполненная до этого процента, будет использоваться только для расширения уже содержащихся в ней BLOB-данных, новые BLOB-данные размещаться в эту страницу не будут. По умолчанию процент заполнения страницы BLOB равен 50%.

Пример

```
CREATE TABLE "TABLE1" ( … )  maxrowid 300000000 PCTFILL 90;
```

В данном примере будет размечено пространство под таблицу TABLE1 в размере достаточном для хранения 300 000 000 записей, также максимальный номер ROWID будет установлен в значение 300 000 000 и отношение среднего размера упакованной записи к среднему размеру неупакованной – 90%.

Более подробное описание можно найти в документации:

- «Архитектура СУБД»;
- «Справочник по SQL».
- «Создание и конфигурирование базы данных».

Особенности работы с широкими записями и BLOB-файлами

При работе с записями шириной более 4К и данными типа BLOB необходимо учитывать следующие особенности:

- широкие записи хранятся в BLOB-файлах, поэтому указанные при создании таблицы значения параметров PCTFREE и PCTFILL фактически не применяются, а применяется значение параметра BLOBPCT, которое по умолчанию имеет значение 50%;
- одна запись целиком пишется в один BLOB-файл;

- при записи нового значения (выполнение INSERT-команды) будет использован BLOB-файл с минимальным номером из всех BLOB-файлов, в которых есть свободные страницы, если их нет ни в одном BLOB-файле – BLOB-файл с минимальным размером из всех BLOB-файлов, которые можно расширить;
- при дописывании значения (выполнение UPDATE-команды) будет использован тот BLOB-файл, в котором расположено это значение, при необходимости он расширяется;
- если все BLOB-файлы достигнут максимального размеры будет выдан код завершения 1709 и даже после добавления еще одного BLOB-файла UPDATE-команда может завершиться с кодом завершения 1709.

Рекомендация: для предотвращения переполнения BLOB-файлов таблицы необходимо периодически следить за размером последнего BLOB-файла. Проверку можно выполнять с помощью запроса вида:

```
select cast GetByte($$$S14, 102) as int,
  linter_file_info($$$S11,2,GetByte($$$S14, 102),'size') * 4096
  from LINTER_SYSTEM_USER.$$$SYSRL, LINTER_SYSTEM_USER.$$$USR
  where $$$S32=0
    and $$$S31=$$$S12
    and $$$S34=<имя_схемы>
    and $$$S13=<имя_таблицы>;
```

Запрос возвращает количество BLOB-файлов и размер последнего BLOB-файла. Как только размер последнего BLOB-файла превысит 60 GB (или 1.5 GB для ОС не поддерживающих большие размеры файлов, например ЗОСРВ «Нейтрино»), рекомендуется добавить ещё один BLOB-файл командой:

```
alter table <имя_таблицы> add blobfile;
```

и далее следить за размером этого нового файла с помощью указанного запроса (новый BLOB-файл стал последним).

Настройка параметров системного журнала

Информация обо всех изменениях БД, производимых транзакциями, фиксируется в специальном наборе файлов – системном журнале. Запись информации в системный журнал всегда предшествует выполнению изменения БД. Системный журнал может располагаться в одном или нескольких файлах операционной системы. Запись информации в системный журнал выполняется циклически, т.е. при достижении конца файла новые записи размещаются на месте самых старых записей.

Для повышения производительности БД рекомендуется:

- при работе с большим объемом данных в одной транзакции установить небольшое количество больших файлов, т.к. вставка происходит большими порциями и интенсивная работа с множеством мелких файлов происходит очень медленно;
- при использовании большого количества мелких транзакций увеличить количество файлов журнала.

Допустимо произвольное изменение количества файлов журнала и размера файла журнала. Наиболее оптимальные значения можно подобрать экспериментально, следя, например, за производительностью БД и нагрузкой на файловую систему.

Для настройки параметров файлов журнала можно воспользоваться утилитой `gendb` или установкой через интерфейс приложения «Администратор СУБД ЛИНТЕР».

Пример для gendb

```
SET SYSLOG COUNT 10;
SET SYSLOG SIZE 100000;
```

Пример для интерфейса приложения «Администратор СУБД ЛИНТЕР»

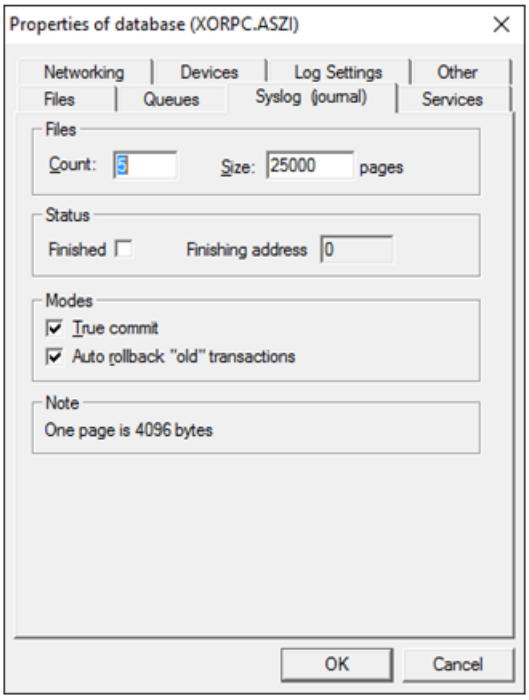


Рисунок 7. Установка параметров файлов журнала

Более подробное описание можно найти в документации:

- «Архитектура СУБД»;
- «Создание и конфигурирование базы данных»;
- «Сетевой администратор».

Рекомендации по настройке СУБД для работы с большими объемами данных

Под большими объемами данных понимается:

1) количество записей в таблице в диапазоне 10 000 000 – 1 000 000 000;
2) количество записей в формируемой ядром СУБД выборке данных в диапазоне 100 000 – 1 000 000 000.

Каждый вид работы с данными большого объема требует индивидуальной настройки, которая выполняется с помощью соответствующего программного компонента СУБД ЛИНТЕР:

1) настройки БД (оптимизация размеров рабочих файлов СУБД) – конфигурируются с помощью утилиты `gendb` или утилиты «Администратор СУБД ЛИНТЕР» (для ОС Windows);
2) настройки ядра СУБД – конфигурируются с помощью ключей команды запуска ядра СУБД;
3) настройки табличных объектов БД – конфигурируются в команде создания и модификации табличных объектов.

Настройка СУБД зависит от вида работы с большими данными:

1) создание табличных объектов для хранения большого объема данных;
2) периодическое массовое добавление в таблицу большого объема данных;
3) штатный режим работы с данными (выполнение единичных операций манипулирования данными);
4) периодические процедуры по оптимизации работы с большими объемами данных в процессе функционирования СУБД.

Оптимизация размеров табличных файлов БД

При создании таблиц БД ядро СУБД размечает файл данных и файл индексов таблицы под указанное количество записей. Если в процессе работы СУБД количество записей таблицы превысит запланированное, то СУБД будет расширять указанные файлы по правилам, изложенным в пункте «Автоматическое управление размерами рабочих файлов».

При попытке добавить данные в файл, достигший максимального размера, СУБД выдает соответствующий код завершения (например, 14 «Файл данных переполнен»). В этом случае необходимо создать новый файл нужного типа с помощью одной из SQL-конструкций:

```
ALTER TABLE … ADD DATAFILE;
ALTER TABLE … ADD INDEXFILE;
ALTER TABLE … ADD BLOBFILE;
```

Максимальное количество файлов (данных, индексов, BLOB) равно 63. При превышении их количества выдается код завершения 2107 «Недопустимое количество файлов».

Максимальный размер одного файла (данных, индексов, BLOB) в СУБД ЛИНТЕР может быть равен 512*32768*4096 = 64 Гбайт или ограничен в меньшую сторону файловой системой ОС.

Таким образом, максимальное количество записей (размером до 4 Кбайт), которое может быть размещено в таблице БД, вычисляется по формуле:

```
MaxCountRec = 63 * MaxFileSize / CompactSizeRec
```

где:

`MaxCountRec` – максимально возможное количество упакованных до размера CompactSizeRec записей в таблице;

`MaxFileSize` – максимально возможный размер одного файла данных (64 Гбайт или максимально допустимый размер файла в ОС);

`CompactSizeRec` – средний размер упакованной записи, вычисляется по формуле:

```
CompactSizeRec = RecSize * PCTFILL / 100
```

где:

`RecSize` – декларируемая при создании таблицы длина записи;

`PCTFILL` – параметр, характеризующий степень упаковки записей;

Примечание
Данная формула применима для расчета при декларируемой длине записей до 4 Кбайт. Более длинные записи (от 4 Кбайт до 64 Кбайт) хранятся не в файлах данных, а в BLOB-файлах.

Так, например, при размере упакованной записи в 500 байт в файлах данных таблицы можно разместить одновременно (теоретически максимально) примерно 2 млрд. записей (63*16*1000000000/500), хотя СУБД реально обеспечивает поддержку только до 1 млрд. записей.

Для планирования таблицы с большими объемами данных:

1) указать с помощью утилиты `gendb` количество страниц, добавляемых при расширении файлов таблиц.

   Пример

```
gendb
gendb>SET EXTSIZE 256;
```

2) указать в SQL-операторе создания базовой таблицы следующие параметры:

   - планируемое максимальное количество записей таблицы (параметр `MAXROWID`). Параметр указывает на то, что надо зарезервировать столько страниц конвертера данных таблицы, чтобы их хватило для размещения информации о `MAXROWID` записей (то есть `MAXROWID` влияет только на количество страниц бит-конвертера);
   - планируемое место на диске для хранения заданного количества записей (параметр `MAXROW`). Параметр указывает на то, что надо создать первоначальный

файл данных для одновременного хранения в нем MAXROW записей (размер рассчитывается исходя из декларируемого размера записей);

- вычисленные значения параметров PCTFILL, PCTFREE, BLOBPCT;
- количество индексных файлов таблицы и их атрибуты (параметр INDEXFILES). С целью уменьшения размеров индексных файлов (и, соответственно, сокращения времени операций ввода/вывода) рекомендуется для каждого индекса таблицы выделять отдельный индексный файл.

  После исчерпания выделенного файлу индексов пространства он будет, при необходимости, расширяться порциями, указанными в конструкции EXTSIZE <размер расширения файлов> утилиты gendb до максимально возможного размера 64 Гбайт;
- если планируемый объем данных таблицы может превышать максимально допустимый размер файла данных с учетом всех возможных расширений (64 Гбайт), то следует указать количество необходимых файлов данных/индексов (параметр DATAFILES/INDEXFILES).

Пример

```
CREATE TABLE [<имя схемы>.]<имя таблицы> ( … ) MAXROWID <значение>
 MAXROW <значение> PCTFILL <значение> PCTFREE <значение> BLOBPCT
 <значение> INDEXFILE 3 DATAFILE 2;
```

Оптимизация работы ядра СУБД

Пул ядра СУБД

При выборе размера пула ядра СУБД (ключ /POOL командной строки запуска ядра СУБД) необходимо руководствоваться следующими правилами:

1) идеальной для СУБД является ситуация, когда размер выделенной ядру оперативной памяти не меньше, чем суммарный размер всех файлов каталога БД;
2) так как для БД большого объема выполнить это условие бывает затруднительно, то крайне желательно, как минимум, соблюсти менее жёсткое ограничение: в оперативной памяти должны целиком помещаться таблицы БД, с которыми идёт наиболее интенсивная работа и одновременно вспомогательные файлы СУБД (рабочий файл SYSWRK и файл бит-векторов SYSWBV).

Пример определения размера наиболее часто используемых таблиц. Допустим, мы знаем, что ведётся интенсивная работа с таблицами AUTO и PERSON:

1) находим описание этих таблиц в системной таблице $$$SYSRL с помощью SQL-запроса:

```
select * from LINTER_SYSTEM_USER.$$$SYSRL where $$$S13 in ('AUTO',
 'PERSON');
```

2) запоминаем номера таблиц (идентификаторы) из этой выборки данных (значение поля $$$S11). Допустим, что для таблицы AUTO $$$S11=176, а для таблицы PERSON $$$S11=177;
3) в каталоге БД смотрим размер файлов 176.0X, 176.1X, 177.0X, 177.1X, 1.31, 1.41 и ориентируемся при задании параметра /POOL, как минимум, на суммарный размер этих файлов +20%.

 © Рекомендации по настройке СУБД ЛИНТЕР. АО НПП «РЕЛЭКС», 1990-2026

Пул сортировки

При выборе размера пула сортировки ядра СУБД (ключ `/SPOOL` командной строки запуска ядра СУБД) необходимо руководствоваться следующими правилами:

1) если для БД задан один процесс сортировки, то по умолчанию при автоконфигурировании СУБД выделяет для пула сортировки эквивалент 1/4 части от пула ядра СУБД (от значения ключа `/POOL`), на это значение надо ориентироваться при задании параметра `SPOOL` при обычной работе с СУБД;
2) для ряда DDL-операций (например, для массового создания или пересоздания индексов) бывает целесообразно задать сравнительно небольшое значение ключа `/POOL` и за его счёт увеличить значение пула сортировки. Например, если при обычной работе используются значения `/POOL=2000000` и `/SPOOL=500000`, то (при условии одного процесса сортировки) перед созданием индексов бывает целесообразно перезапустить ядро СУБД, например, с параметрами `/POOL=500000` и `/SPOOL=2000000`, а после создания индексов вновь перезапустить ядро СУБД со старыми параметрами `/POOL=2000000` и `/SPOOL=500000`.

Загрузка большого объема данных

Массовое добавление данных в пустую таблицу

Предварительные условия:

1) таблица должна быть создана согласно пункту «Оптимизация размеров табличных файлов БД»;
2) при создании таблицы не должны быть использованы опции, требующие индексации загружаемых данных (атрибут столбца `PRIMARY KEY`);
3) загружаемые данные могут индексироваться, но во время загрузки лучше все-таки отменить индексирование с помощью SQL-запроса `DROP INDEX <имя индекса>`, чтобы сократить время загрузки.



Примечание
Перед загрузкой данных рекомендуется создать полный архив БД (для повторной загрузки в случае неуспешной предшествующей загрузки).

Загрузка данных:

1) желательно завершить на компьютере работу всех посторонних программ;
2) запустить ядро СУБД ЛИНТЕР со следующими параметрами:
   - `/POOL=N`, где N – максимально возможное в этом сеансе значение (т.е. попытаться размещать добавляемые данные в пуле ядра СУБД в максимально возможном объеме перед сбросом их в файл данных);
   - не следует включать какую-либо трассировку выполнения SQL-запросов (по умолчанию трассировка выполнения SQL-запросов отменена) нет необходимости.
3) загрузку данных выполнять либо клиентским приложением, либо с помощью утилиты `loarel` СУБД ЛИНТЕР в транзакционном режиме AUTOCOMMIT. В обоих случаях использовать пакетный режим загрузки данных:
   - клиентское приложение (`call-интерфейс`): использовать команду `PUTM` (см. документ «Интерфейс нижнего уровня»);
   - `loarel`: выполнять загрузку с ключом `-s 2` (см. документ «Импорт данных»);

- `proc`: использовать команды `START APPEND`/`END APPEND` (см. документ «Процедурный язык»).

После загрузки данных:

1) создать, при необходимости, индексы.

   Если в таблице необходимо создать нескольких индексов, то рекомендуется хранить информацию об индексах в отдельных индексных файлах. Для этого:
   - указать количество индексных файлов (если это не было сделано при создании таблицы):

```
ALTER TABLE … ADD INDEXFILES <количество файлов>;
```

   - создать индексы и указать их местоположение в отдельных индексных файлах, начиная со второго (первый индексный файл используется в качестве конвертера):

```
CREATE INDEX … INDEXFILE 2 [BY APPEND];
```

   Атрибут `BY APPEND` заставляет индексировать данные без предварительной сортировки, что уменьшает размер используемого дискового пространства, но увеличивает время создания индекса;
2) создать, при необходимости, условия ограничения целостности:

```
ALTER TABLE … ADD PRIMARY KEY;
ALTER TABLE … ADD UNIQUE … [INDEXFILE <номер файла>] [BY APPEND];
ALTER TABLE … ADD FOREIGN KEY … [INDEXFILE <номер файла>] [BY
 APPEND];
```

3) если загрузка данных завершилась некорректно, необходимо выявить и устранить причину, восстановить БД из архива и повторить загрузку данных.

Массовое добавление данных в рабочую таблицу

Под рабочей таблицей понимается таблица БД, в которую уже загружена объемная часть данных либо в процессе первоначальной загрузки в пустую таблицу, либо в процессе работы клиентского приложения.

Предварительные условия:

1) создать полный архив БД (для повторной загрузки данных в случае неуспешной предшествующей загрузки);
2) завершить работу всех пользователей БД;
3) для таблицы должны быть определены индексы символьных данных большой размерности;
4) при добавлении в таблицу большого числа записей зачастую оказывается выгоднее предварительно удалить индексы, затем добавить данные и заново создать индексы, чем добавлять данные в таблицу одновременно с индексированием.



Примечание
Так как эффективность загрузки индексируемых данных сильно зависит от самих данных, то для выбора оптимального варианта можно провести тестовые замеры на сравнительно небольших объемах данных и на основе полученных результатов принять решение об индексировании данных в процессе загрузки данных или индексировать данные после их загрузки.

Загрузка данных:
Выполнять аналогично загрузке данных в пустую таблицу.

После загрузки данных:
Действия аналогичны при загрузке в пустую таблицу.

Управление ссылочной целостностью загруженных данных

Если в таблице создается N индексов, то необходимо, по возможности, размещать каждый индекс в отдельном файле, то есть создать столько индексных файлов таблицы, сколько в ней определено индексов (простых, составных, функциональных) с помощью SQL-запроса `CREATE INDEX`:

1) добавление индексных файлов таблицы:

```
ALTER TABLE … ADD INDEXFILE;
```

По умолчанию при создании таблицы всегда создается один индексный файл (содержит конвертер индексируемых данных), поэтому если в таблице N индексов, то необходимо добавить N индексных файлов и размещать индексы начиная со второго индексного файла.

2) перед загрузкой больших объемов данных рекомендуется удалять условия ссылочной целостности, а после загрузки данных – создавать их заново;

3) т.к. условия ссылочной целостности всегда реализуются как индексы, то рекомендуется размещать информацию о них в отдельных индексных файлах:

```
CREATE INDEX <имя индекса> ON <имя таблицы>
 INDEXFILE <номер файла индексов>;
ALTER TABLE <имя таблицы> ADD PRIMARY KEY …
 INDEXFILE <номер файла индексов>;
ALTER TABLE <имя таблицы> ADD UNIQUE (имя столбца)
 INDEXFILE <номер файла индексов>;
ALTER TABLE <имя таблицы> ADD FOREIGN KEY …
 INDEXFILE <номер файла индексов>;
```

Манипулирование данными большого объема

Рекомендации относятся к случаю интенсивного манипулирования данными (операции `INSERT`, `UPDATE`, `DELETE`).

Если транзакция выполняет массовое обновление записей некоторой таблицы и скорость такой операции значительно более медленная, чем ожидается, то рекомендуется заблокировать на время выполнения этой операции обновляемую таблицу (LOCK TABLE) и разблокировать ее по окончании операции (UNLOCK TABLE). Это исключает блокировки на уровне записей и тем самым значительно повышает скорость выполнения операции.

Рекомендации зависят от возможности размещения таблиц с большим объемом данных в пуле ядра СУБД:

1) если выполняются SQL-запросы типа `INSERT`, `SELECT`, `UPDATE`, `DELETE` с условиями, отличными от условий вида rowid=1, желательно загрузить таблицы в пул

ядра СУБД. Для этого для каждой используемой в SQL-запросе таблицы необходимо выполнить SQL-команду:

```
TEST TABLE <имя таблицы> PRELOAD;
```

После этой команды данные и индексы таблицы будут загружены в оперативную память, и фрагментация индекса не будет иметь значения.

2) при отсутствии возможности загрузить таблицу в пул ядра СУБД рекомендуется периодически пересоздавать условия ссылочной целостности, выполняя SQL-запросы:

```
DROP INDEX <имя индекса>;
CREATE INDEX <имя индекса> ON <имя таблицы> INDEXFILE <номер файла
 индекса>;
```

Регулярная перестройка индексов устраняет дефрагментацию индексного файла и повышает производительность обработки индексированных данных.

Примечание

Предварительная загрузка таблицы в оперативную память не сильно влияет на выполнение простых запросов вида:

```
insert into table1 values(10);
delete from test1 where rowid=1;
```

Использование таблиц в памяти

Таблицы в памяти полезны в случае необходимости высокопроизводительного манипулирования временными данными. Таблицы в памяти не подходят для длительного хранения данных, т.к. очищаются при перезапуске ядра СУБД, но работают быстрее обычных таблиц за счет преимущества в скорости оперативной памяти перед файловой системой. Примером использования таблиц в памяти может являться подготовка данных для отчетов или подготовка данных из различных источников перед загрузкой их в основное хранилище внутри одной транзакции.

По умолчанию СУБД ЛИНТЕР не предусматривает работу с таблицами «в памяти». Для поддержки этой функциональности необходимо:

1) сконфигурировать СУБД с помощью утилиты `gendb` (см. документ «Создание и конфигурирование базы данных»). Установить значения следующих параметров:
   - задать размер очереди для таблиц «в памяти» (команда вида `SET IN-MEMORY TABLES;`);
   - задать количество столбцов у таблиц «в памяти» (команда вида `SET IN-MEMORY COLUMNS;`);
   - задать количество файлов (команда вида `SET IN-MEMORY FILES;`).

   Пример

```
set in-memory tables 30;
set in-memory files 60;
set in-memory columns 300;
```

2) установить максимально допустимое количество страниц в пуле страниц ядра СУБД ЛИНТЕР для таблицы «в памяти».

   При каждом запуске ядра СУБД ЛИНТЕР указывать ключ `/INMEMPOOL=<размер>`, где `<размер>` задает максимально допустимое количество страниц в пуле страниц ядра СУБД, выделяемых для размещения таблиц «в памяти» (см. документ «Запуск и останов СУБД ЛИНТЕР в среде ОС Windows») либо установить через интерфейс приложения «Администратор СУБД ЛИНТЕР». Значение задается в страницах размером 4 Кбайт.

   Для расчета значения количества страниц можно воспользоваться следующим алгоритмом:
   - вычислить максимальный объем, занимаемый одной записью таблицы, сложив максимальный размер каждого столбца;
   - умножить значение из предыдущего пункта на максимальное предполагаемое количество строк в таблице;
   - таким образом, получаем размер памяти, занимаемый одной таблицей;
   - повторить предыдущие пункты для всех таблиц «в памяти»;
   - сложить значения размеров памяти для всех таблиц «в памяти» и разделить на размер страницы 4 Кбайт;
   - полученное значение будет являться максимальным количеством необходимых страниц, которое необходимо установить в значении ключа `/INMEMPOOL`, добавив некоторый запас.

В виде формулы описанный алгоритм имеет следующий вид:

$$INMEMPOOL = \frac{1}{4096}\sum_{i=1}^{N}\left(K\sum_{j=1}^{M_i}L_{ij}\right)$$

где

N – количество таблиц;

K_i – максимальное количество строк i-й таблицы;

M_i – количество столбцов i-й таблицы;

L_{ij} – максимальный размер значения j-го столбца i-й таблицы в байтах.

Пример

```
linter /INMEMPOOL=100000
```

Также настроить параметр можно через интерфейс приложения «Администратор СУБД ЛИНТЕР»:



Рисунок 8. Установка параметров таблиц в памяти

3) в случае активной работы с таблицами «в памяти» рекомендуется увеличить размер памяти канала, выполнив команду:

```
ALTER DATABASE SET CHANNEL MEMORY LIMIT 1048576;
```

и перезапустить ядро СУБД.

Для того, чтобы сохранить на диск внесенные в таблицу данные, необходимо выполнить команду:

```
SAVE TABLE <имя_таблицы>;
```

а для восстановления сохраненной ранее таблицы необходимо выполнить команду:

```
RESTORE TABLE <имя_таблицы>;
```

Для автоматической загрузки и сохранения данных таблиц «в памяти» можно воспользоваться опциями таблиц «в памяти»: `AUTOLOAD` и `AUTOSAVE`. Операции над таблицами «в памяти» не являются транзакционными, на них не действуют операции `COMMIT` и `ROLLBACK`. При ошибке операции над таблицей «в памяти» может быть выполнено автоматическое восстановление к старому состоянию, при этом на консоль ядра и в `linter.out` выдается сообщение: "In-memory table ИМЯ_ТАБЛИЦЫ was rolled back to last saved/initial state after error КОД_ОШИБКИ".

Более подробное описание можно найти в документации:

- «Архитектура СУБД»;
- «Создание и конфигурирование базы данных»;
- «Запуск и останов СУБД ЛИНТЕР в среде ОС Windows»;
- «Сетевой администратор».

Расчет размера файла сортировки

При построении ключей, индексов или при выполнении команды `REBUILD TABLE...` СУБД ЛИНТЕР использует файл сортировки.

Примечание
При создании индекса и модификации таблицы с использованием опции `BY APPEND` не требуется дополнительной дисковой памяти под файл сортировки, но сортировка данных будет выполняться несколько медленнее.

Далее приведен порядок расчета размера файла сортировки в страницах по известным параметрам таблицы.

Пусть:

N – общее число столбцов в индексе/ключе;

V – из них число столбцов типов переменной длины (VARCHAR, VARBYTE и т.д.);

S – суммарная максимальная длина всех столбцов в байтах;

Тогда каждая запись сортировки для индекса/ключа занимает (7 + N*3 + V*2 + S). Кроме собственно значений столбцов туда еще входят ROWID, длины значений, NULL-флаги, мандатный доступ и т.д.

Исходя из размера одной записи сортировки, можно посчитать, сколько их поместится в одну страницу, и, соответственно, сколько страниц файла сортировки достаточно будет для создания индекса.

По умолчанию лимит файла сортировки равен 524287 страниц (т.е. файл длиной 4 GB). Но `gendb` сейчас разрешает поставить и больший лимит.

Пример

```
create table test_tabl(col_vc(100), col_vc2(100), col_vc3(100));
(7 + 3*3 + 3*2 + 300) = 322 байт
В таблице 60 000 000 записей, следовательно,
общий требуемый размер файла сортировки будет
(322*60 000 000) : 4 000 (байт в странице) = 4 830 000 страниц.
```

Оценка задержки в выполнении стандартного запроса

Входными данными для вычисления прогнозируемого времени обработки простого запроса являются:

- текст простого SQL-запроса;
- прогнозируемая таблица (таблица, из которой производится выборка по простому запросу, время исполнения которого следует спрогнозировать);
- тестовая таблица (эталонная таблица со структурой, идентичной прогнозируемой, расположенная на том же носителе, но с меньшим числом записей).

Формула оценки времени исполнения простого запроса и расшифровка ее компонентов:

$$Tp = Tt * \left(\frac{Cp}{Ct}\right),$$

Рисунок 9. Формула оценки времени исполнения простого запроса

где:

Tp – время выполнения запроса по прогнозируемой таблице;

Tt – время выполнения запроса по тестовой таблице;

Cp– число записей в прогнозируемой таблице;

Ct– число записей в тестовой таблице.

Примечание
Формула корректна при условии, что размер тестовой таблицы меньше размера прогнозируемой.

Пример

Допустим, созданы три таблицы: `test` – `7500` записей, `prognoz1` – `15000` записей, `prognoz2` – `31000` записей.

Запустить программу `inl` от имени пользователя `SYSTEM/MANAGER8`:

```
inl -u SYSTEM/MANAGER8
```

Выполнить команду для изменения режима отображения временной статистики программы `inl`:

```
SQL> time HH:MI:SS:MS
```

Выполнить select-запрос из тестовой таблицы:

```
SQL> select * from test where vch = 'sel';
INL : начальное время : 02:43:38:776
      конечное время :  02:43:38:861
```

Вычислить время выполнения запроса из тестовой таблицы (для этого необходимо найти разность между конечным временем выполнения запроса и начальным).

Провести расчет прогнозируемого времени выполнения аналогичных запросов для таблиц `prognoz1` и `prognoz2` согласно формуле оценки времени исполнения простого запроса. Для таблицы `prognoz1`, содержащей `15000` записей, формула, с подставленными в неё значениями, будет выглядеть следующим образом:

$$Tp1 = 0.085 * \left(\frac{15000}{7500}\right) = 0.17;$$

Рисунок 10. Вычисление для таблицы prognoz1, содержащей 15000 записей

Для таблицы prognoz2, содержащей 31000 записей:

$$Tp2 = 0.085 * \left(\frac{31000}{7500}\right) \approx 0.351;$$

Рисунок 11. Вычисление для таблицы prognoz2, содержащей 31000 записей

Выполнить select-запрос из прогнозируемой таблицы `prognoz1`:

```
SQL> select * from prognoz1 where vch = 'sel';
INL : начальное время : 02:47:35:767
      конечное время :  02:47:35:932
```

Вычислить реальное времени выполнения запроса из таблицы `prognoz1`.

Сравнить прогнозируемое время выполнения запроса для таблицы `prognoz1`, то есть, с реальным (вычисленным на предыдущем шаге).

Выполнить select-запроса из прогнозируемой таблицы prognoz2:

```
SQL> select * from prognoz2 where vch = 'sel';
INL : начальное время : 02:48:23:248
      конечное время :  02:48:23:596
```

Вычислить реальное время выполнения запроса из таблицы `prognoz2`.

Сравнить прогнозируемое время выполнения запроса для таблицы `prognoz2`, то есть, с реальным (вычисленным на предыдущем шаге).

Примечание

При расчётах и сравнении следует учитывать погрешность измерения времени в 10-15%, поскольку некоторое машинное время тратится на обмен данными между компонентами системы, управление кешем СУБД и прочие накладные и периодические расходы.

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.

Оптимизация SQL-запросов

Оптимизация SQL-запросов – важный шаг при обработке любого оператора языка манипулирования данными с заданными во фразе WHERE условиями поиска. Цель оптимизатора – выбор наиболее эффективного способа выполнения SQL-запроса. Оптимизатор встроен в ядро СУБД, и в своей работе использует универсальный метод оптимизации, применяемый ко всем SQL-запросам определенной структуры. В силу универсальности оптимизатора результаты его работы могут оказаться не достаточно эффективными для некоторого конкретного SQL-запроса. Часто разработчик клиентского приложения знает об обрабатываемых данных гораздо больше, чем оптимизатор, поэтому он может выбрать более эффективный способ выполнения SQL-запроса, чем оптимизатор, или дать оптимизатору необходимые подсказки (hints).

В СУБД ЛИНТЕР оптимизация выполнения SQL-запросов осуществляется на двух уровнях:

1) внутренний уровень – оптимизация ядром СУБД на основе синтаксиса SQL-запроса. При использовании этого метода учитывается иерархическое старшинство операций. Если для какой-либо операции существует более одного пути ее выполнения, то выбирается тот путь, чей ранг выше, т.к. в большинстве случаев он выполняется быстрее, чем путь с более низким рангом;
2) внешний уровень – с помощью подсказок программиста. Оптимизатор не может использовать индексный путь доступа, основываясь только на существовании индекса; эффективный путь доступа к данным можно подсказать в тексте SQL-запроса.

Внутренний уровень оптимизации

Разбирая фразу WHERE в SQL-запросе, оптимизатор полностью, где возможно, вычисляет выражения, и преобразует некоторые синтаксические конструкции в эквивалентные им конструкции. Такие преобразования позволяют СУБД быстрее вычислить результирующее выражение по сравнению с исходным.

Замена оператора сравнения на противоположный

Оптимизатор всегда упрощает условие, чтобы избавиться от логического оператора NOT.

Это преобразование подразумевает замену исходного оператора сравнения эквивалентным противоположным оператором.

Например,

Исходная конструкция	Оптимизированная конструкция
NOT(<выражение1> = <выражение2>)	<выражение1> != <выражение2>
NOT(<выражение1> != <выражение2>)	<выражение1> = <выражение2>
NOT(<выражение1> LIKE <выражение2>)	<выражение1> NOT LIKE <выражение2>
NOT(<выражение1> NOT LIKE	<выражение1> LIKE

```
 <выражение2>)                                  <выражение2>
```

Часто условие, содержащее логический оператор NOT, можно записать многими разными способами. Оптимизатор пытается трансформировать первоначальное условие таким образом, чтобы составляющие его условия стали как можно более простыми, даже если результирующее условие будет содержать больше операторов NOT.

Например,

```
Исходная конструкция              Оптимизированная конструкция
-------------------------------------------------------------------
NOT (<условие1> OR <условие2>)   (NOT <условие1>) AND (NOT
 <условие2>)
NOT (<условие1> AND <условие2>) (NOT <условие1>) OR (NOT
 <условие2>)
```

Перенос операции на другой уровень

Оптимизатор переносит операцию AND на более низкий уровень, чем OR.

Например:

```
 Исходная конструкция            Оптимизированная конструкция
-------------------------------------------------------------------
 <условие1> AND (<условие2> OR   (<условие1> AND <условие2>) OR
  <условие3>)                    (<условие1> AND <условие3>)
```

Преобразование в эквивалентное условие

Оптимизатор преобразует условие, использующее оператор сравнения IN, в эквивалентное условие, использующее оператор равенства, если список состоит из одного элемента, а оператор NOT IN, соответственно, в оператор неравенства.

Например,

```
 Исходная конструкция             Оптимизированная конструкция
-------------------------------------------------------------
 <выражение1> IN (<выражение2>)    <выражение1> = <выражение2>
```

Преобразование OR в IN

Если во фразе WHERE на один и тот же столбец наложено несколько условий, соединенных операторами OR, то оптимизатор всегда преобразует группу этих условий в одно условие с оператором IN.

Например,

```
 Исходная конструкция             Оптимизированная конструкция
-------------------------------------------------------------
 (<столбец> = <значение1>) OR     <столбец> IN (<общий список>)
 (<столбец> IN <список1>)  OR
 (<столбец> = <значение2>) OR
 (<столбец> IN <список2>)
```

В этой конструкции значения или элементы списков могут быть параметрами.

Слияние нескольких предикатов в один

Если во фразе WHERE значение столбца сравнивается с константами и эти условия объединены логической операцией AND, то оптимизатор преобразует группу таких условий в одно условие с оператором BETWEEN.

Например,

```
 Исходная конструкция              Оптимизированная конструкция
-------------------------------------------------------------------
 <столбец>  = <константа1> AND     <столбец> BETWEEN <константа1>
 AND
 <столбец> <= <константа2>         <константа2>
```

В этой конструкции значения констант могут быть параметрами.

Кроме обычного варианта BETWEEN, включающего в выборку данных совпадения на обоих концах отрезка, могут быть созданы еще три варианта BETWEEN с исключением одного или обоих концов отрезка.

Изменение порядка обработки условий

Оптимизатор изменяет порядок вычисления условий в группах SQL-запроса таким образом, чтобы минимизировать количество промежуточных записей при вычислении каждой группы условий.

Группа условий – это совокупность предикатов, соединенных операциями AND. Записи в ранее вычисленных предикатах группы будут использоваться при вычислении последующих предикатов.

Если группа содержит несколько предикатов одного ранга, то они будут обрабатываться в том порядке, в котором записаны в условии.

Оптимизатор применяет следующий порядок вычисления предикатов в группах:

1) сначала вычисляются константные предикаты.

   Например,

```
1 = 1, (2+4) <= 5
```

   Подобные условия формируются, в основном, при автоматическом создании SQL-запросов;
2) затем вычисляются предикаты типа

```
<первичный ключ> <OP>  <константа>
```

   здесь <OP> – это одна из операций: =, LIKE или IN;
3) следующими вычисляются однопеременные предикаты, в которых участвуют столбцы, не являющиеся первичными ключами и содержащие условия типа «=», LIKE или IN.

   Например,

```
<столбец1> IN (1, 2, 3);
```

```
<столбец1> LIKE '%alt';
```

4) далее вычисляются однопеременные предикаты (т.е. содержащие условия, отличные от «=», LIKE, IN).

Например,

```
B.Y <= 15;
A.X between 12 and 17;
```

5) в последнюю очередь вычисляются многопеременные предикаты.

При этом несколько следующих друг за другом многопеременных предикатов упорядочиваются таким образом, чтобы переменные, входящие в последующий предикат, в максимальном количестве входили в предыдущие предикаты группы.

Например,

```
 Исходная конструкция              Оптимизированная конструкция
-------------------------------------------------------------
 <условие A.X> = B.X AND           <условие A.X> = B.X AND
 C.Z = D.Z           AND           B.Y = C.Y           AND
 B.Y = C.Y                         C.Z = D.Z
```

Из многопеременных предикатов первыми будут вычислены те, которые содержат условия типа «=», LIKE или IN, и только потом остальные.

Запрос вида

```
"SELECT ... FROM
  (SELECT ... FROM A WHERE CONDITION1
UNION ALL
  SELECT ... FROM B WHERE CONDITION2)
WHERE
  CONDITION3;"
```

преобразуется в запрос

```
"SELECT ... FROM
(SELECT ... FROM A WHERE CONDITION1 AND CONDITION3
 UNION ALL
 SELECT ... FROM B WHERE CONDITION2 AND CONDITION3);"
```

Чтобы преобразование произошло, требуется:

- в качестве CONDITION1, CONDITION2, CONDITION3 должны выступать предикаты и группы предикатов (CONDITION1 и CONDITION2 также могут вообще отсутствовать);
- внутренние SELECT-запросы не должны использовать функции.

Раскрытие подзапросов

В зависимости от сложности SQL-запроса оптимизатор пытается преобразовать вложенные подзапросы в одноуровневые запросы.

Например,

```
Исходная конструкция                          Оптимизированная
конструкция
-------------------------------------------------------------------------
SELECT * FROM                                 SELECT * FROM AUTO
(SELECT * FROM AUTO WHERE MAKE                WHERE MAKE = 'FORD' AND
='FORD')                                      MODEL LIKE '%V8%';
WHERE MODEL LIKE '%V8%';
```

Это преобразование не выполняется для запросов:

- с конструкциями GROUP BY и HAVING;
- со сложными вариантами OUTER JOIN;
- с внешними ссылками и функциями;
- содержащих представление (VIEW). (Исключение составляют запросы, содержащие представления без фразы WHERE).

К разновидности этой оптимизации относится преобразование стандартной формы OUTER JOIN в ORACLE-подобную форму записи.

Например,

```
Исходная конструкция                  Оптимизированная конструкция
-------------------------------------------------------------
SELECT * FROM PERSON                  SELECT * FROM PERSON,AUTO
LEFT OUTER JOIN AUTO                  WHERE PERSON.PERSONID =
ON PERSON.PERSONID =                  AUTO.RERSONID(+)
AUTO.RERSONID                         AND NAME = 'KIM';
WHERE NAME = 'KIM';
```

Исключение лишних сортировок

Оптимизатор исключает из SQL-запроса предложение ORDER BY в следующих случаях:

1) одновременное использование GROUP BY и ORDER BY.

   Например, поступил на обработку SELECT-запрос с GROUP BY и ORDER BY, в котором все столбцы, указанные в ORDER BY, указаны также и в GROUP BY.

   В этом случае конструкция ORDER BY убирается из запроса, а сортировка по GROUP BY выполняется таким образом, чтобы соблюдались порядок и направление сортировки в ORDER BY;

2) одновременное использование DISTINCT и ORDER BY.

   Например, в главном SELECT-запросе присутствуют одновременно конструкции DISTINCT и ORDER BY, и все элементы списка ORDER BY входят в список SELECT.

   В этом случае элементы списка ORDER BY оптимизатор ставит первыми в списке SELECT-запроса для обработки условия DISTINCT, причем в том же порядке, в каком

они заданы в ORDER BY. При этом элементам списка присваиваются спецификации ASC/DESC, а конструкция ORDER BY убирается из запроса.

Например,

```
 Исходная конструкция                Оптимизированная конструкция
-------------------------------------------------------------------
 SELECT DISTINCT(MAKE), MODEL,       SELECT DISTINCT(MAKE), COLOR,
 MODEL,
 COLOR,WEIGHT FROM AUTO              WEIGHT FROM AUTO
 WHERE MODEL LIKE '%V8%'             WHERE MODEL LIKE '%V8%';
 ORDER BY MAKE, COLOR;
```

Преобразование EXISTS в IN

Преобразование SQL-запроса, содержащего оператор EXISTS, в запрос с оператором IN происходит в том случае, когда связка между внутренним и внешним запросами идет через один предикат.

Например,

```
Исходная конструкция                  Оптимизированная конструкция
-------------------------------------------------------------------
SELECT CLIST1 FROM TLIST1             SELECT CLIST1 FROM TLIST1
WHERE COND1 AND EXISTS                WHERE COND1 AND T1.C1 IN
(SELECT CLIST2 FROM TLIST2 WHERE      (SELECT T2.C2 FROM T2 WHERE
 COND2);
T1.C1=T2.C2 and COND2);
```

где T1 входит в TLIST1, T2 в TLIST2 и COND2 не содержит обращений к T1.

В результате такого преобразования исключается внешняя ссылка и запрос отрабатывается значительно быстрей.

Использование транзитивности

Если во фразе WHERE общий столбец участвует в двух условиях, то оптимизатор может иногда вывести из них третье условие, используя принцип транзитивности.

Например,

```
A.X = B.X AND A.X = C
```

заменяются на

```
A.X = C AND B.X = C
```

Принцип транзитивности применим ко всем операторам сравнения, т.е. =, !=, ^=, <>, <, >, <=, >=.

Оптимизатор может заменять лишь такие условия, которые связывают столбцы с константными выражениями, но не с другими столбцами.

Например, пусть фраза WHERE содержит два условия следующего вида:

```
WHERE A.X = B.Y AND A.X = C.X
```

В этом случае принцип транзитивности не используется, т.е. оптимизатор **не** сформирует фразу WHERE со следующим условием:

```
B.Y = C.X
```

Замена выражений

В случае если в группе соединенных по условию AND предикатов есть предикат равенства первичного ключа и не константного выражения (которым, чаще всего, является внешний ключ), то в других предикатах этой группы условий внешний ключ можно заменить соответствующим первичным ключом, что ускорит выполнение запросов. Оптимизация имеет обычные для подобных оптимизаций ограничения (отсутствие необходимости преобразований операндов, специальных типов соединения и т.д.).

Выбор оптимального пути доступа к данным

Одним из наиболее важных решений, принимаемых оптимизатором при формировании плана исполнения SQL-запроса, является выбор метода доступа к данным.

Основные методы доступа, посредством которых СУБД может обращаться к данным, следующие:

- просмотр только по индексам;
- сначала просмотр по индексам, а затем полный просмотр таблицы (перебор);
- просмотр таблицы.

Просмотр индекса позволяет извлекать данные из индекса на основании значения одного или нескольких столбцов в индексе. Помимо каждого индексированного значения, индекс содержит ROWID тех записей таблицы, на которые он ссылается.

Полный просмотр таблицы извлекает строки непосредственно из таблицы. При выполнении полного просмотра СУБД считывает из таблицы все строки и проверяет каждую строку на соответствие условию фразы WHERE SQL-запроса.

Ниже рассматривается выбор оптимизатором стратегии доступа к данным.

Стратегия доступа к данным при вычислении однопеременных предикатов

Рассмотрим однопеременный предикат вида

```
<столбец> <OP> <значение>
```

где <OP> – некоторая логическая операция.

Для обработки подобных предикатов будет применен индекс, если выполняется одно из условий:

1) <OP> входит в множество операций: =, <>, >=, <=, >, <, IS NULL, IN, а <значение> – константа, внешняя ссылка, список констант (для IN) или подзапрос;

2) <OP> – это оператор LIKE, а <значение> – текстовая константа, которая не начинается с '_' или '%';

3) <столбец> – индексированный столбец или столбец производного множества, получаемый из индексированного столбца без каких-либо преобразований;

4) <столбец> – это ROWID и при вычислении предиката не требуется преобразования типов;

5) <столбец> – это столбец, для которого задан предикат на равенство и у которого существует одностолбцовый составной индекс, а при вычислении не требуется преобразования типов;

6) <столбец> – столбец, входящий в многостолбцовый составной индекс, первый элемент которого тоже входит в условие, вычисляемое по индексу. При наличии нескольких составных индексов делается попытка выбрать наилучшее множество этих индексов для обработки группы предикатов.

Если предикат не может быть вычислен по индексу, однако зависит только от одного индексированного столбца, то он вычисляется с помощью прохода по индексу.

Например,

```
LOWER(PASSWORD) = 'manager'
```

где PASSWORD – индексированный столбец.

При вычислении предикатов с использованием индексов выполняется проверка (после завершения обработки каждого следующего предиката) количества оставшихся в выборке данных записей.

Если число записей стало меньше некоторой границы (200 записей), то все оставшиеся предикаты вычисляются перебором (полным просмотром таблицы), в противном случае оптимизатор вычисляет приблизительно, сколько страниц нужно считать для обработки каждого предиката по индексу и сколько – при обработке его перебором, и принимает решение, переводить ли этот предикат на обработку перебором.

Предикат типа «<столбец> IN <подзапрос>» может реально вычисляться как двухпеременный предикат.

Стратегия доступа к данным при вычислении многопеременных предикатов

Выбор стратегии доступа для вычисления многопеременного предиката осуществляется по следующим критериям:

1) предикат типа <столбец1> <OP> <столбец2> будет обрабатываться слиянием индексов, если <OP> – операция «=» или «!=», <столбец1> и <столбец2> – либо индексированные столбцы, либо ROWID, либо выражения, для которых построен временный индекс;

2) для многопеременных предикатов используются составные индексы, если существуют два таких индекса T1(C1,C2,...,CN) и T2(C1,C2,...CN), что в группу входят все предикаты типа T1.C1=T2.C1, T1.C2=T2.C2, ... T1.CJ=T2.CJ для J >= 2;

3) если в одной части предиката находится индексированный столбец (или ROWID, или производный столбец, полученный из индексированного без каких-либо преобразований), а в другой части – выражение, не зависящее от переменной этого

столбца, то один предикат обрабатывается по индексу, и происходит перебор по значениям всех остальных переменных.

Полный перебор для многопеременных предикатов осуществляется с помощью циклического прохода по каждой из участвующих переменных в отдельности (либо по всем значениям этой переменной, либо по значениям, отобранным в процессе обработки однопеременных предикатов по этой переменной).

При выборе любой стратегии, кроме использования составных индексов, осуществляется проверка на возможность оптимизации выполнения предиката с помощью построения временных индексов. Временные индексы полезны, если в одной из частей предиката стоит выражение с одной переменной или неиндексированный столбец. Временный индекс создается также в случае, если столбец индексирован, но содержит значительно больше значений по сравнению с уже отобранным множеством. Временные индексы удаляются сразу же после завершения обработки предиката.

Внешний уровень оптимизации

Использование индексов

Оптимизатор СУБД ЛИНТЕР может анализировать SQL-запросы с точки зрения существования индексов, позволяющих ускорить их обработку. Для этого ядро СУБД должно быть запущено с ключом `ANALYZE`. Список рекомендуемых индексов будет выведен на экран консоли ядра и в файл `linter.out`.

Если ядро СУБД запущено с ключом `AUTOINDEX`, то перед выполнением SQL-запроса оптимизатором будут созданы необходимые индексы. Построенные индексы сохраняются в БД и после выполнения SQL-запроса.

Подсказки

Оптимизировать выполнение SQL-запроса можно с помощью механизма подсказок. Подсказки задаются программистом (разработчиком приложения) непосредственно в тексте SQL-запроса и используются оптимизатором при планировании обработки запроса. Ядро СУБД распознает следующие типы подсказок:

- об очередности слияния результатов вычисления предикатов;
- о раскрытии представлений, задействованных в SQL-запросе;
- о сортировке (применять указанный индекс без использования конструкции ORDER BY).

Кроме того, существует общая рекомендация по оптимизации SQL-запроса: запрос должен быть составлен таким образом, чтобы условия (во фразе WHERE) шли в порядке уменьшения объема выбираемых данных, т.е. первыми должны проверяться условия, по которым предполагается небольшой процент выбираемых строк таблицы.