МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
ВОЛОГОДСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Кафедра автоматики и вычислительной техники

РАСПРЕДЕЛЕННЫЕ ФАЙЛОВЫЕ СИСТЕМЫ, БАЗЫ И БАНКИ ДАННЫХ

Методические указания к лабораторным работам. Часть 2

Факультет электроэнергетический

Направления подготовки магистров:
09.04.01 – Информатика и вычислительная техника
09.04.04 – Программная инженерия

Вологда
2014

УДК 621.374.1

Распределенные файловые системы, базы и банки данных: методические указания к лабораторным работам. Часть 2. - Вологда: ВоГУ, 2014. - 26 с.

Приводятся подробные пояснения к лабораторному практикуму, содержащие теоретические сведения и указания, необходимые для успешного выполнения лабораторных работ, формирования и развития профессиональных компетенций в области распределенного хранения и обработки данных.

Утверждено редакционно-издательским советом ВоГУ

Составители: С.Ю. Ржеуцкая, канд. техн. наук, доцент
И.Н. Вахрушев, ассистент

Рецензент А.М. Водовозов, канд. техн. наук, профессор
зав. кафедрой УВС

Введение

Лабораторный практикум предназначен для ознакомления магистрантов с наиболее важными практическими аспектами создания и администрирования распределенных файловых систем и баз данных, организации эффективного доступа к данным. Данные методические указания содержат подробные пояснения по выполнению последних четырех лабораторных работ, каждая из которых позволяет освоить определенные темы лекционного курса и сформировать (развить) профессиональные компетенции в области распределенного хранения и обработки данных.

В дополнение к данным методическим указаниям следует использовать рабочую программу дисциплины, учебное пособие «Базы данных. Язык SQL» и дистанционный практикум по языкам SQL и PL/SQL, установленный на сервере кафедры АВТ и доступный по адресу http://atpp.vstu.edu.ru/cgi-bin/arh_problems.pl.

1. Лабораторная работа №5. Безопасность и целостность данных

Цель работы: *освоение подсистемы безопасности СУБД Oracle, приобретение умений разграничивать доступ пользователей с помощью системы привилегий и ролей, обеспечивать целостность данных в условиях параллелизма на основе механизма транзакций*

1.1 Пояснения к выполнению работы

Все СУБД предоставляют доступ к информации базы данных только зарегистрированным пользователям. При установке Oracle автоматически создается несколько учетных записей – это пользователи с именами SYS и SYSTEM, обладающие всеми правами АБД (DBA), а также несколько пользователей, созданных в демонстрационных целях.

Для создания учетной записи каждого нового пользователя требуется создать стандартный объект USER с обязательным указанием пароля данного пользователя. В Oracle команда CREATE USER … содержит еще ряд дополнительных параметров:

CREATE USER имя_пользователя IDENTIFIED BY пароль
[DEFAULT TABLESPACE имя_табличного_пространства]
[TEMPORARY TABLESPACE имя_временного_табличного_пространства]

[QUOTA UNLIMITED / размер предоставляемого дискового пространства]
[PROFILE имя_профиля]

Например:

```
CREATE USER user1 IDENTIFIED BY cneltyn
DEFAULT TABLESPACE USERS
TEMPORARY TABLESPACE TEMP
```

В приведенном примере размер части табличного пространства, предоставляемого пользователю, не ограничен (точнее, ограничен только размерами пространства). Специальный профиль для пользователя не создается.

Команда ALTER USER имя ..... позволяет изменять различные параметры учетной записи (кроме имени пользователя).

Например, для смены пароля:

```
ALTER USER user1 IDENTIFIED BY новый_пароль
```

Для того чтобы гарантированно удалить любого пользователя, требуется применить команду DROP USER с параметром CASCADE.

```
DROP USER user1 CASCADE
```

При создании учетной записи в Oracle каждый пользователь одновременно получает в распоряжение свою собственную схему в базе данных с тем же именем. Это самостоятельная часть базы данных, все имена объектов, создаваемых пользователем, должны быть уникальны только в пределах схемы, поскольку имя объекта (таблицы, представления и т.д.) в пределах всей базы данных складывается из двух составляющих: имя_схемы.имя_объекта

Однако, получив в распоряжение собственную учетную запись и собственную схему, только что созданный пользователь не может даже подключиться к базе данных. Для выполнения любого действия в базе данных пользователю требуются права на выполнение подобных действий. Для этих целей вводятся два понятия – привилегии и роли.

Привилегии позволяют использовать определенные команды языка SQL по отношению к определенным объектам и предоставляются с помощью команды GRANT… Например, если пользователь user1 владеет таблицей students и выполняет команду

```
GRANT SELECT ON students TO PUBLIC,
```

все пользователи (PUBLIC) смогут выполнять команду SELECT по отношению к таблице students, используя для этого составное имя user1.students или короткий синоним, созданный при помощи команды:

```
CREATE PUBLIC SYNONYM students FOR user1.students
```

Упрощают работу с привилегиями *роли* – именованные группы привилегий, которые можно предоставлять пользователем с помощью той же команды GRANT, что и отдельные привилегии. Роль – это

объект базы данных, поэтому она создается при помощи стандартной команды CREATE.

CREATE ROLE имя_роли

Например, создадим роль student, которую могут получить все пользователи-студенты, изучающие курс «РФСББД».

CREATE ROLE student

Наполнить роль набором привилегий можно при помощи известной команды GRANT.

Например, команда:

GRANT connect, resource TO student

присвоит роли student две стандартные роли, которые имеются во всех версиях Oracle – роль connect позволяет подключаться к базе данных, а роль resource позволяет создавать некоторые объекты в своей схеме.

Если роли student необходимо предоставить все права администратора, это можно сделать при помощи команды:

GRANT dba TO student

Для присвоения этой роли пользователю user1 .. используем команду:

GRANT student TO user1

Однако в реально функционирующих информационных системах использование стандартных ролей Oracle часто нежелательно, поэтому новые роли обычно вручную наполняются набором привилегий.

Как известно, основными составными частями языка SQL являются DDL и DML. В соответствии с этим выделяют две больших группы привилегий – системные и объектные.

Системные привилегии – это права на выполнение команд CREATE, ALTER DROP применительно к различным объектам базы данных. Если пользователи или роли наделяются привилегией на выполнение DDL в любой схеме, к имени объекта добавляется слово ANY.

Например:

GRANT CREATE TABLE TO student

включает в роль студент право создавать таблицы в своей схеме, а

GRANT CREATE ANY TABLE TO student WITH GRANT OPTION

право создавать таблицы в любой схеме, при этом роль student сможет передать эту привилегию другим пользователям или ролям.

Важно отметить, что пользователь, создавший таблицу (неважно где – в своей или чужой схеме), автоматически становится ее *владельцем* (owner), при этом он автоматически получает все права на эту таблицу.

Особо требуется отметить привилегию CREATE SESSION, которую должны получить все пользователи без исключения, поскольку это право на создание сеанса связи с Oracle.

Объектные привилегии позволяют пользователям выполнять конкретные действия на конкретном объекте. Они назначаются конечным пользователям, которые используют приложения базы данных для выполнения конкретных задач.

Объектные привилегии выдаются при помощи уже известной универсальной команды GRANT…

GRANT список_объектных_привилегий
ON имя таблицы/представления/процедуры
TO список_ролей/список_пользователей/PUBLIC
[WITH GRANT OPTION]

В ORACLE существуют следующие привилегии объектного уровня:

- SELECT позволяет выполнить запрос на выборку данных таблицы или представления.
- INSERT позволяет вставлять строки в таблицу.
- UPDATE позволяет обновлять строки.
- DELETE позволяет удалять строки таблицы.
- EXECUTE даёт возможность выполнять процедурные объекты.
- ALTER даёт возможность изменить структуру таблицы.
- INDEX позволяет пользователю создавать индексы на таблицу, владельцем которой он не является.

Например, пусть ранее был создан пользователь user2. Предоставим ему только некоторые права пользователя демонстрационной базы данных студентов и их оценок:

GRANT CREATE SESSION TO user2
GRANT SELECT, INSERT ON marks TO user2
GRANT SELECT ON students TO user2
GRANT EXECUTE ON change_phone TO user2

Далее приведем пояснения к механизму транзакций Oracle, необходимые для выполнения работы.

Транзакция – единица работы СУБД, которая может быть выполнена либо целиком, либо вообще не выполнена. Объем транзакции может варьироваться от одного SQL-оператора до всех действий с базой данных, выполняемых приложением. Транзакция характеризуется четырьмя свойствами, часто называемыми свойствами АСИД, – Атомарность, Согласованность, Изолированность, Долговечность.

Первые два свойства обеспечиваются стандартными командами COMMIT (фиксация результатов транзакции) и ROLLBACK (откат транзакции, т.е. возврат базы данных к начальному состоянию, которое было до начала транзакции),

Для обеспечения свойства изоляции транзакций в СУБД используется система блокировок. Команды CREATE/ALTER/DROP TABLE накладывают эксклюзивную блокировку на уровне таблицы (*блокировка TX*). Это значит, что нельзя выполнять никаких действий над таблицей, пока не будет закончена эта операция DDL.

Команды INSERT/DELETE/UPDATE используют ресурсы в более мягком режиме. Каждая из этих команд накладывает эксклюзивную блокировку только на ту строку, которую она в данный момент обрабатывает (*блокировка R*X). Однако одновременно накладывается эксклюзивная блокировка на всю таблицу (*блокировка TS*), и это означает, что никакая DDL операция не может быть выполнена над таблицей до тех пор, пока не закончатся все DML-операции над этой таблицей и не будет снята последняя TS-блокировка.

Команда SELECT не накладывает никакой блокировки на таблицы, данные из которых она выбирает. Сервер Oracle гарантирует каждой команде выборки неизменность данных в процессе ее выполнения.

Сервер Oracle поддерживает еще один вид блокировки – разделяемая на уровне строк (RS-блокировка). Эта блокировка накладывается специальной командой

SELECT * FROM имя таблицы WHERE условие FOR UPDATE,

которая гарантирует следующей за ней команде UPDATE невозможность наложения эксклюзивных блокировок со стороны транзакций других пользователей на строки таблицы, выбранные при помощи условия WHERE.

Механизм блокировок поддерживается Oracle автоматически, однако имеется возможность для ручного управления этим процессом при помощи команды LOCK TABLE, которая позволяет наложить на таблицу (но не на строку!) эксклюзивную или разделяемую блокировку. Так, команда

LOCK TABLE student IN EXCLUSIVE MODE

установит эксклюзивную блокировку на всю таблицу student, которая будет действовать до команды COMMIT или ROLLBACK.

1.2 Порядок выполнения работы

Задания 1 и 2 можно выполнять в произвольном порядке, поскольку они затрагивают два различных аспекта защиты данных – безопасность (конфиденциальность) данных и поддержка их целостности.

Задание 1. Используя схему учебной базы данных «Автовокзал», подробно изученную в предыдущих работах (описание схемы можно найти и в дистанционном практикуме на сайте кафедры АВТ), а также хранимый код, разработанный в лабораторной работе №3, создайте следующие роли пользователей информационной системы Автовокзала:

А. Администратор

Б. Кассир

В. Диспетчер

Наполните каждую из ролей необходимой совокупностью привилегий, достаточной для выполнения всех функций, предусмотренных бизнес-ролями.

Создайте пользователей с такими ролями. Полезно создать синонимы для обеспечения «прозрачности» доступа к данным.

Убедитесь, что ранее созданные хранимые процедуры выполняются с правами их владельцев, при этом пользователю (кассиру или диспетчеру) достаточно только привилегии на запуск процедуры (execute).

После сдачи работы преподавателю удалите созданные роли и пользователей, чтобы не загружать базу данных уже не нужными объектами.

Задание 2. Создайте два сеанса связи с сервером (запустите два экземпляра SQL*Plus с одним и тем же именем пользователя, в схеме которого имеется база данных Автовокзал). Это можно сделать на одном компьютере, или на двух (работа парами), результат будет одним и тем же.

В процессе работы необходимо вести журнал ответов на вопросы «Что будет?»

1. Убедитесь, что вы видите из двух сеансов одну и ту же таблицу trips.

   Внимание! Если вы используете не SQL*Plus, а другое средство связи с сервером Oracle (например, Apex), снимите флажок Autocommit, если он установлен.

2. Поэкспериментируйте с командой insert. Добавьте по одной строке из каждого сеанса сначала без фиксации транзакции. Что будет? Затем зафиксируйте, посмотрите, что изменилось в ваших сеансах.
3. Попробуйте удалить одновременно одну и ту же строку. Что будет?
4. Попробуйте написать update из разных сеансов на разные строки без фиксации. Что будет? Откатите сделанные изменения.
5. Теперь напишем select … for update на всю таблицу в одном из сеансов. Снова напишите update из разных сеансов на разные строки без фиксации. Что будет?
6. Напишите команду insert в обоих сеансах. Что будет?
7. Запишите команду lock table, чтобы перевести таблицу в эксклюзивный режим. Попробуйте подать все команды DML из разных сеансов. Что будет?

Обсудите ваши ответы на вопросы «Что будет?» с преподавателем и выполните предложенный им тест.

2. Лабораторная работа №6. Исследование подсистемы хранения данных. Настройка структур хранения данных

Цель работы *– исследование табличных пространств СУБД Oracle, анализ и устранение фрагментации на уровне сегментов, исследование альтернативных способов хранения данных и их влияние на производительность СУБД.*

2.1 Пояснения к выполнению работы

Табличные пространства

Технология хранения данных на основе табличных пространств используется многими СУБД. Рассмотрим ее на примере Oracle.

База данных разделяется на несколько логических частей, называемых табличными пространствами. Каждое табличное пространство состоит из одного или более файлов данных. Размер табличного пространства – это размер его файла данных или суммарный размер всех файлов.

В процессе создания БД Oracle автоматически строится несколько табличных пространств, названия которых могут отличаться для различных версий. По мере эксплуатации системы администратор может добавлять новые табличные пространства.

Каждая база данных ORACLE содержит табличное пространство SYSTEM, которое создается автоматически при создании базы данных. Табличное пространство SYSTEM содержит таблицы словаря данных для всей базы данных (начиная с версии Oracle 10g для этих целей появилось дополнительное табличное пространство SysAux). Для хранения пользовательских таблиц имеется пространство USERS, для функционирования UNDO-журналов - RBS (UNDO в версии Oracle 10g и выше).

Файлы данных

Файлы данных, ассоциированные с табличным пространством, хранят все данные этого табличного пространства. При создании файла данных для табличного пространства ORACLE распределяет ему указанное количество дисковой памяти. После первоначального создания файла данных соответствующее дисковое пространство еще не содержит никаких данных; однако это пространство уже зарезервировано за ассоциированным табличным пространством.

В СУБД Oracle контроль над дисковым пространством файлов данных происходит с использованием следующих логических структур:
- блоки данных
- экстенты
- сегменты

Сегмент – физическое хранилище одного объекта БД. Oracle использует четыре типа сегментов:
- сегмент данных содержит таблицу
- индексный сегмент содержит индекс
- сегмент отката (UNDO) хранит информацию для отката транзакций
- временный (промежуточный) сегмент используется для выполнения сортировок во внешней памяти.

Экстенты являются строительными блоками сегментов. Экстент – непрерывная область на диске, которая выделяется под какой-нибудь сегмент. Обычно при создании таблицы или индекса их сегмент содержит только один экстент. По мере разрастания данных для сегмента выделяются новые экстенты, которые могут располагаться как угодно, не обязательно в соседних областях диска. Таким образом, сегмент содержит целое число экстентов.

Блоки данных – это наименьшие единицы БД. Они физически хранятся на диске и представляют собой минимальную единицу обмена

между внешней и оперативной памятью. Размер блока устанавливается для каждой базы данных при ее создании. Этот размер кратен размеру блока операционной системы, но не превышает определенный максимум.

Формат блока данных один и тот же, независимо от того, содержит ли блок данные таблицы, индекса или UNDO-журнала. Каждый блок содержит заголовок, далее следует информация (например, строки таблицы), в конце блока обычно выделяется свободное пространство, необходимое для будущего наращивания размеров данных. Подробнее об этом будет разъяснено в следующей лабораторной работе.

Каждая строка любой таблицы базы данных Oracle имеет уникальный идентификатор – rowid, который является ее физическим адресом и состоит из трех частей:

- номер файла данных,
- номер блока в файле данных,
- номер строки в блоке.

Неизменность rowid гарантируется в течение всего времени жизни строки - от команды INSERT, которая создает новую строку, до команды DELETE, которая ее удаляет.

Однако в процессе выполнения команд обновления UPDATE может возникнуть ситуация, когда данные столбца типа VARCHAR, увеличившись в размере, уже не умещаются в том блоке, где первоначально была размещена строка. Строка при этом перемещается (мигрирует) в другой блок, но ее rowid не изменяется, поскольку по первоначальному адресу размещается ссылка на новый адрес. Так возникает самый опасный вид фрагментации базы данных - фрагментация на уровне строк.

Фрагментация строк существенно снижает производительность СУБД, поскольку увеличивает количество блоков, которые необходимо прочитать при выполнении запроса. В работе представлен способ диагностики и устранения данного вида фрагментации.

Далее в работе изучаются различные структуры хранения данных Oracle: «обычные» (heap-organized, HOT) таблицы, таблицы, организованные в виде индекса (index-organized, IOT), кластеры таблиц.

1. HOT и IOT

Мы создали схему данных для информационной системы «Автовокзал». Обратим внимание на таблицу points_routes, реализующую связь «многие ко многим» сущностей «Пункты назначения» и «Маршруты». Согласно логике приложения комбинация обоих ее полей (cod_point, cod_route) должна быть уникальной

(первичный ключ таблицы). Это ограничение реализуется в Oracle составным индексом по этим двум полям. Причем других полей, не входящих в индекс, в таблице нет. Таким образом, мы имеем явную избыточность данных: одни и те же пользовательские данные хранятся в таблице и в индексе. Причем наличие индекса обязательно согласно логике приложения. Такой случай как нельзя лучше подходит для использования таблицы, организованной по индексу (IOT), все данные которой хранятся в листовых блоках b*tree индекса. Заметим, что IOT в Oracle подобна кластерному индексу в MS SQL Server, т.е. данная технология хранения таблиц является достаточно распространенной.

В отличие от «обычной» таблицы, данные в IOT хранятся упорядоченно. Следовательно, операции, модифицирующие данные таблицы, в общем, более ресурсоемки, нежели для heap-таблиц без индексов. Однако в рассматриваемом нами случае (таблица points-routes), где индекс включает все столбцы таблицы и является обязательным, использование IOT не вызовет никаких дополнительных накладных расходов.

При использовании «базового» решения (heap-table + b*tree index) в запросах на выборку с большой долей вероятности будет использоваться индекс. В случае операторов, модифицирующих таблицу, необходимо модифицировать также и индекс. Так что, теоретически, использование IOT приведет к улучшению производительности операторов модификации таблицы и оставит без изменения (почти) производительность выборки данных.

2. Кластеры таблиц

Как известно, Oracle поддерживает кластеры двух типов – индексированные и хешированные. Для схемы Автовокзал не видится необходимости объединять связанные таблицы в индексированный кластер (объясните, почему). Однако есть смысл создать хешированный кластер на основе таблицы trips, поскольку для этой таблицы нужен очень быстрый точечный поиск (поиск рейса по его коду при продаже билетов). В процессе работы вы выполните данные действия.

2.2 Порядок выполнения работы

Задание 1. Исследуйте табличные пространства SYSTEM, USERS, UNDO. Все результаты исследования записывайте в таблицу 2.1

Таблица 2.1. Параметры текущего состояния табличных пространств

Название	Размер	Кол-во оставшегося свободного пространства	Наибольший свободный экстент	Возможность добавления новых сегментов	Степень фрагментированности
SYSTEM					
UNDO					
USERS					

Для выполнения работы вам потребуются следующие запросы к представлениям словаря данных Oracle:

Размеры табличных пространств:

```
SELECT TABLESPACE_NAME, SUM(BYTES) FROM
DBA_DATA_FILES GROUP BY TABLESPACE_NAME;
```

Количество свободного пространства:

```
SELECT TABLESPACE_NAME, SUM(BYTES) FROM
DBA_FREE_SPACE GROUP BY TABLESPACE_NAME;
```

Наибольший свободный экстент:

```
SELECT TABLESPACE_NAME, MAX(BYTES) FROM
DBA_FREE_SPACE GROUP BY TABLESPACE_NAME;
```

Для определения возможности расширения сегментов табличного пространства нужно выяснить параметр NEXT физического хранения для *всех сегментов этого табличного пространства***,** т.к. у разных сегментов он может быть разным.

Следующая команда покажет все сегменты заданного табличного пространства:

```
SELECT SEGMENT_NAME, SEGMENT_TYPE, OWNER,
INITIAL_EXTENT, NEXT_EXTENT, PCT_INCREASE FROM
DBA_SEGMENTS WHERE TABLESPACE_NAME=<имя>;
```

Запишите самостоятельно команду, которая определит максимальное значение NEXT_EXTENT для исследуемых табличных пространств и сравните полученные значения с наибольшими свободными экстентами для этих пространств. Сделайте вывод о возможности или невозможности дальнейшего расширения имеющихся сегментов в табличных пространствах.

Запишите еще одну команду, которая определит максимальное значение INITIAL_EXTENT для исследуемых вами табличных

пространств и сравните полученные значения с наибольшими свободными экстентами для этих пространств. Сделайте вывод о возможности или невозможности создания новых сегментов в табличных пространствах.

Запишите команду для определения максимального значения PCT_INCREASE для исследуемых пространств. Если оказались ненулевые значения, то попробуйте выяснить, каким сегментам они принадлежат.

Оцените степень фрагментированности исследуемых табличных пространств по следующим показателям:

- общее количество свободных экстентов в табличном пространстве,
- отношение размера наибольшего свободного экстента к общему размеру свободного пространства.

Задание 2. Цель данного задания — искусственно добиться фрагментации строк таблицы, а затем устранить мигрирующие строки (chained rows). Задание состоит в последовательном выполнении нескольких пунктов.

1. В вашей схеме выделите одну из таблиц для эксперимента с мигрирующими строками или создайте специально таблицу с одним числовым и одним текстовым полем, и заполните таблицу небольшим количеством строк. Проанализируйте эту таблицу.

```
ANALYZE TABLE  <имя таблицы> COMPUTE STATISTICS;
SELECT blocks, empty_blocks, avg_space FROM user_tables
WHERE table_name = '<имя таблицы заглавными буквами>';
```

2. Проверьте значения PCTFREE и PCTUSED для вашей таблицы .

```
SELECT pct_free, pct_used FROM user_tables
WHERE table_name = '<имя таблицы>' ;
```

3. Уменьшите значение PCTFREE до 0% (иначе будет трудно получить мигрирующие строки).

```
ALTER TABLE <ИМЯ> PCTFREE 0;
```

Добавьте в таблицу столько строк, чтобы один блок был почти полностью заполнен. Текстовое поле пока заполните не очень длинными текстами (мы будем увеличивать размеры текстов, чтобы получить мигрирующие строки).

4. Создайте в своей схеме таблицу *chained_rows***,** выполнив команду (или запустив файл *utlchain.sql):*

```
create table CHAINED_ROWS (
  owner_name        varchar2(30),
  table_name        varchar2(30),
```

```
    cluster_name       varchar2(30),
    partition_name     varchar2(30),
    subpartition_name  varchar2(30),
    head_rowid         rowid,
    analyze_timestamp  date
  );
```

5.Выполните команду ANALYZE.

```
ANALYZE TABLE <имя таблицы>
LIST CHAINED ROWS INTO chained_rows;
```

6. Сколько сцепленных строк вы видите в таблице *chained_rows* ? Наверное, ни одной, ведь мы еще не обновляли тексты в таблице.

Принудительно создайте в таблице мигрирующие строки. Это легче сделать, если в вашей таблице текстовое поле будет иметь приличный размер. Увеличьте его размер, например, так:

```
ALTER TABLE <имя>  MODIFY <имя столбца> VARCHAR2(500);
```

Придумайте какую-нибудь команду UPDATE, чтобы размеры текстового поля резко увеличились.

7. Снова выполните команду ANALYZE**.** Сколько мигрирующих строк вы видите? Если опять нет, то снова наращивайте текстовое поле.

8. Устраните фрагментацию строк. Для этого перепишите мигрирующие строки в какую-нибудь временную таблицу (например, с именем hold), затем удалите их и снова вставьте. Проверяйте результаты каждого шага.

```
CREATE TABLE hold
AS SELECT *  FROM <ВАША ТАБЛИЦА>
WHERE rowid in (SELECT head_rowid
                                  FROM chained_rows);
```

Убедитесь, что строки появились в таблице hold.

```
DELETE FROM  <ваша таблица>
WHERE rowid in (SELECT head_rowid
                                  FROM chained_rows) ;
```

Убедитесь, что они удалились из вашей таблицы.

```
INSERT  INTO   <ВАША ТАБЛИЦА>
SELECT  *  FROM  hold;
```

Теперь ваша таблица снова заполнена теми же строками, которые в ней были и раньше, но все они не фрагментированы.

Очистите таблицу *chained_rows* и снова выполните команду ANALYZE TABLE. Убедитесь, что на этот раз таблица *chained_rows* пуста. Если это не так, значит, вы что-то не аккуратно сделали. Снова выполните пункт 7.

Задание 5. В этом и следующих заданиях мы создадим копии таблиц базы данных «Автовокзал», но с альтернативными способами их внутреннего физического представления.

1. Создадим еще одну таблицу-связку (points_routes_iot) на основе таблицы points-routes и поместим в новую таблицу все ее данные, а саму таблицу сделаем IOT.

```
CREATE TABLE points_routes_iot (
  cod_point INTEGER NOT NULL REFERENCES POINTS,
  cod_route INTEGER NOT NULL REFERENCES ROUTES,
PRIMARY KEY (cod_point, cod_route)
)
ORGANIZATION INDEX
```

Заполните таблицу points_routes_iot данными из таблицы points_routes_iot (напишите запрос на вставку самостоятельно).

Это один случай, при котором «напрашивалось» использование IOT.

2. Также можно рассматривать применение IOT для таблиц-справочников. Эти таблицы также имеют, по логике приложения, первичный ключ – как правило, суррогатный, а также несколько полей-параметров (наименование и т.п.). Такие таблицы обычно нечасто модифицируются, а обращение к ним происходит, как правило, по первичному ключу. В таком случае для получения, скажем, наименования из справочника требуется найти соответствующий ключ в индексе (index unique scan), а затем считать строку из таблицы. Если же все поля строки будут храниться в листовых блоках индекса (IOT), мы сможем сэкономить последнее логическое чтение. Кроме того, IOT может оказаться значительно более эффектной при поиске по диапазону (index range scan).

В схеме информационной системы «Автовокзал» справочниками можно назвать таблицы POINTS, ROUTES, KM_PRICES, MODELS. Для них можно рассмотреть целесообразность использования IOT.

Создайте самостоятельно IOT на основе таблицы points.

3. Измените физические параметры таблиц для схемы Автовокзал (параметр PCT_FREE) исходя из того, что все таблицы-справочники обновляются редко (посмотрите предыдущие задания).

4. Создайте хешированный кластер и поместите туда таблицу – копию таблицы trips

```
CREATE CLUSTER имя(имя_кластерного_ключа, тип)
HASHKEYS размер хеш-таблицы;
```

CREATE TABLE trips_hash(……)
CLUSTER имя(имя_кластерного_ключа);

Оцените размер хеш-таблицы исходя из размеров таблицы trips.

Заполните таблицу данными из trips с помощью запроса на вставку (напишите самостоятельно).

Созданные копии таблиц схемы Автовокзал не удалять, т.к. они будут использоваться в следующих лабораторных работах по запросам и производительности сервера.

3. Лабораторная работа №7. Настройка производительности SQL-запросов

Цель работы: *исследовать различные режимы оптимизатора запросов и планы выполнения различных запросов.*

3.1 Пояснения к выполнению работы

Исходный текст SQL-запроса, переданный по сети из клиентского приложения серверу Oracle, сначала подвергается проверке на правильность всех синтаксических конструкций и наличие всех таблиц и столбцов с именами, заданными в тексте запроса. Для запроса, который признан правильным, затем формируется *план его исполнения (Query Plan)*, представляющий собой описание (во внутреннем формате СУБД) наиболее оптимального способа реализации тех реляционных операций, которые содержатся в тексте запроса. Все эти действия выполняет специальный компонент СУБД, который называется *оптимизатором запроса* **(**Query Optimizer**),** а сам этап формирования плана исполнения запроса называют компиляцией по аналогии с первым этапом обработки программы, написанной на любом языке программирования**.**

Следует отметить, что этап построения плана запроса может быть пропущен, если запрос выполняется повторно, поскольку планы всех выполненных запросов сохраняются в буфере Oracle. На этот факт следует обратить внимание при выполнении двух следующих лабораторных работ.

Оптимизатор запроса передает план исполнения запроса другому компоненту СУБД, который называется *процессором базы данных* (или

процессором SQL). Процессор БД исполняет все необходимые действия по извлечению и обработке данных. В результате формируется таблица с выходными данными, которая возвращается клиенту, или выполняется модификация данных таблиц БД, если на сервер передавался запрос на обновление данных.

В процессе выполнения работы будут исследованы механизмы работы оптимизатора запросов и способы управления оптимизатором с целью повышения производительности СУБД. Будут рассмотрены два основных режима работы оптимизатора запросов – на основе правил, когда выбор оптимального плана исполнения запроса выполняется только на основе SQL-текста, и оптимизация на основе издержек (стоимостной оптимизатор), в которой учитываются различные параметры хранения данных.

3.2. Порядок выполнения лабораторной работы

1. Работу можно выполнять либо с использованием утилиты SQL*Plus, либо используя Apex (в этом случае ее можно выполнять удаленно). Если вы используете SQL*Plus, создайте в вашей схеме служебную таблицу PLAN_TABLE при помощи команды:

```
create table PLAN_TABLE (
        statement_id        varchar2(30),
        timestamp           date,
        remarks             varchar2(80),
        operation           varchar2(30),
        options             varchar2(30),
        object_node         varchar2(128),
        object_owner        varchar2(30),
        object_name         varchar2(30),
        object_instance numeric,
        object_type     varchar2(30),
        optimizer       varchar2(255),
        search_columns  number,
        id                  numeric,
        parent_id           numeric,
        position numeric,
        cost                numeric,
        cardinality         numeric,
        bytes               numeric,
        other_tag       varchar2(255),
        partition_start varchar2(255),
```

```
partition_stop  varchar2(255),
partition_id    numeric,
    other               long,
    distribution    varchar2(30));
```

2. В меню Параметры/Конфигурация (в SQL*PLUS) установите параметр Autotrace в состояние "Вкл." (или введите команду set autotrace on). Попробуйте выполнить любой запрос к любой таблице, убедитесь что вы видите не только его результаты, но еще план и статистику выполнения.

Для того, чтобы видеть еще и время выполнения запроса, в Конфигурации можно включить опцию Timing.

В режиме автотрассировки таблица plan_table не заполняется, но для каких-то внутренних целей требуется ее наличие.

Если вы используете Apex, план выполнения любого запроса можно увидеть, перейдя на вкладку Explain.

Далее последовательно выполняем все пункты, следующие ниже. По ходу исследования готовим отчет – текстовый документ, содержащий по каждому пункту тексты запросов и их планы.

3. Для выполнения исследования понадобится большая и маленькая таблицы. Наша задача убедиться, что оптимизатор на основе правил будет использовать один и тот же план и для большой, и для маленькой таблицы, а оптимизатор на основе издержек будет учитывать размер таблицы.

Таблицы можно создать и заполнить любым способом, например, так:

```
CREATE TABLE t1(n NUMBER PRIMARY KEY, t VARCHAR2(20));

CREATE OR REPLACE PROCEDURE fill_table_t1(k number) AS
 n NUMBER;
 BEGIN
 DELETE FROM t1;
 FOR n IN 1..k LOOP
 INSERT INTO t1 VALUES(n,'text'||to_char(n));
 END LOOP;
 END;
```

Аналогично создается таблица t2 с теми же полями и процедура fill_table_t2.

Fill_table_t1(10) - это маленькая таблица

Fill_table_t2(10000) - это большая таблица

4. По умолчанию режим оптимизатора Choose, поэтому, пока нет статистических данных, будет работать оптимизация на основе правил (Rule). Выполните запросы

SELECT * FROM t1 WHERE n=5;

SELECT * FROM t2 WHERE n=5;

и посмотрите планы их выполнения. Вы увидите, что в обоих случаях поиск выполняется в уникальном индексе на основе первичного ключа. Этот запрос выполняется одинаково независимо от статистики таблиц, поэтому дальнейшее исследование будем выполнять на текстовом столбце (у нас он имеет имя t).

Выполним запросы

SELECT * FROM t1 WHERE t='text5';

SELECT * FROM t2 WHERE t='text5';

и убедимся, что используется полный просмотр (Full scan)

Создадим обычные древовидные индексы по текстовому полю t для обеих таблиц и выполним запросы

SELECT * FROM t1 WHERE t='text5';

SELECT * FROM t2 WHERE T='text5';

Мы увидим, что теперь поиск выполняется в индексе для обеих таблиц.

Внимание! Если вы хотите, чтобы оптимизатор перестроил план выполнения запроса, обязательно измените что-то в его тексте (например, поменяйте регистр у каких-либо символов или добавьте лишние пробелы). Иначе он возьмет старый план из кэш-буфера запросов.

5. Заставим оптимизатор работать в режиме оценки издержек (это будет режим All_rows – минимизация общего времени выполнения запроса). Для этого достаточно собрать статистику по таблицам.

ANALYZE TABLE t1 COMPUTE STATISTICS;

ANALYZE TABLE t2 cOMPUTE STATISTICS;

Снова введите чуть модифицированные запросы для поиска по текстовому полю и убедитесь, что теперь в большой таблице будет выполняться поиск в индексе, а в маленькой — полный просмотр таблицы (Full scan).

6. Можно заставить оптимизатор выполнять полный просмотр для большой таблицы при помощи подсказки +FULL

SELECT /*+ FULL(t2) */ * FROM t2 WHERE t='text5';

Аналогично можно заставить его выполнять поиск в индексе для маленькой таблицы

SELECT /*+ INDEX (t1 имя вашего индекса) */ * FROM t1 WHERE t='text5';

Понятно, что в данных запросах наши подсказки приведут только к ухудшению производительности.

7. Заставим оптимизатор самостоятельно принять решение о полном просмотре большой таблицы. Для этого резко уменьшим *селективность* текстового столбца, оставив в нем не более 20 уникальных значений (сейчас в нем вообще все значения уникальны). Напишите для этого любой update на ваше усмотрение и не забудьте обновить статистику при помощи analyze table. Желательно перестроить и индекс

ALTER INDEX имя REBUILD

Убедитесь, что теперь поиск будет выполняться полным просмотром, несмотря на наличие индекса.

8. Напишите следующие запросы и посмотрите планы их выполнения (обратите внимание, что наш древовидный индекс оптимизатор никуда не подключает):

1. Все уникальные значения текстового столбца большой таблицы (используйте distinct);
2. Все уникальные значения и количество их повторений (используйте группировку);
3. Все значения текстового столбца, которые есть в обеих таблицах, и соответствующие им значения ключа в каждой таблице (используйте внутреннее соединение таблиц);
4. Все значения текстового поля большой таблицы, которых нет в маленькой (используйте операцию minus);
5. Значения текстового поля (или одно значение), которые встречаются чаще всех (используйте вложенные запросы).

9. Удалите древовидный индекс по текстовому полю и создайте bitmap (create bitmap index …). СУБД Oracle не позволяет создать два разных индекса по одному полю одновременно. Теперь, при низкой селективности текстового столбца древовидный индекс практически перестал использоваться оптимизатором.

Придумайте не менее двух запросов, при которых оптимизатор без всяких подсказок подключит битовый индекс. Например, с использованием агрегатной функции count.

10. Проанализируем планы выполнения запросов с участием таблиц, организованных как индексы. В схеме Автовокзал имеется один

из справочников и таблица-связка Пункты-маршруты в двух вариантах (индекс и куча).

Напишите запросы и проанализируйте их планы для двух вариантов таблиц:

1. Любой точечный поиск по ключу в справочнике
2. Поиск по ограниченному и неограниченному диапазону по ключу в справочнике
3. Поиск по неключевому столбцу в справочнике
4. Запросы с группировкой в таблице-связке
5. Запросы с соединением таблицы-связки с таблицей пункты и/или маршруты

11. Таблица trips (рейсы) также существует в двух экземплярах – обычная куча и хешированный кластер. Проанализируйте планы выполнения запросов для двух экземпляров:

1. Точечный поиск по ключу
2. Поиск по диапазону по ключу
3. Поиск по неключевому столбцу
4. Соединение с таблицей маршрутов, с таблицей-связкой (тут получится уже четыре варианта запросов).

Обсудите с преподавателем подготовленный вами отчет по выполненному исследованию.

4. Лабораторная работа №8. Динамический SQL (Native Dynamic SQL). Анализ влияния размера таблицы на производительность

Цель работы *– развитие умений и навыков использования различных возможностей языка SQL, в том числе, умений динамически формировать запросы, углубление знаний в области технологий хранения и извлечения данных*

4.1. Пояснения к выполнению работы

Операторы *динамического SQL* - в отличие от операторов встроенного SQL - формируются не на этапе компиляции, а на этапе выполнения процедуры или функции, что придает им значительно большую гибкость. Например, с помощью динамического SQL (в Oracle его последняя

реализация получила название Native Dynamic SQL) можно разрабатывать универсальный программный код с переменными именами таблиц, выполнять любые команды DDL в блоке PL/SQL и т.д.

Операторы *динамического SQL* формируются как текстовые переменные. Например:

```
Str:='SELECT * FROM t1';
```

Для динамического формирования оператора можно выполнять конкатенацию строк. Например:

```
Str:= 'SELECT * FROM  '||t_name||
```

Для выполнения строки динамического SQL используется оператор EXECUTE IMMEDIATE, который имеет следующее формальное описание:

```
EXECUTE IMMEDIATE строка с текстом запроса;
```

В работе представлены примеры использования динамического SQL в контексте анализа производительности СУБД в зависимости от размеров таблиц БД.

4.2. Порядок выполнения лабораторной работы

1. СУБД Oracle 10g имеет высокопроизводительный процессор запросов (SQL engine), поэтому подавляющее большинство запросов на выборку работает быстро даже при очень больших размерах таблиц. Запросы на вставку и обновление работают существенно медленнее, особенно в случае использования NATIVE DYNAMIC SQL. Чтобы убедиться в этом, заполним 4 таблицы t1,t2,t3,t4 одинаковой структуры, добавив 1000, 10000, 100000 и 1000000 строк.

Ниже приведена хранимая процедура для заполнения произвольной таблицы с двумя столбцами (числовым и текстовым) произвольным количеством строк. В ней намеренно сделаны 3 синтаксические ошибки, после исправления которых она заработает.
Ошибки при создании процедуры можно посмотреть в user_errors:

```
SELECT line, text FROM user_errors
```

Текст процедуры:

```
CREATE OR REPLACE PROCEDURE fill_any_table(t_name VARCHAR,
n INTEGER)
 AS
i INTEGER
 BEGIN
  EXECUTE IMMEDIATE 'delete from'||t_name;
  FOR i IN 1..n LOOP
```

```
  EXECUTE IMMEDIATE 'insert into '||t_name||'
values('||to_char(i)||','||to_char(i)||');
   END LOOP;
 END;
```

Таблицы t1 и t2 уже есть, создайте еще таблицы t3 и t4 такой же структуры, например:

```
CREATE TABLE t3(n NUMBER PRIMARY KEY, t VARCHAR2(20));
```

Выполните процедуру fill_any_table и заполните таблицы t1,t2 и t3. Обратите внимание, что таблица t3 заполняется очень долго, поэтому для заполнения t4 миллионом строк используем процедуру без динамического SQL, которая работает существенно быстрее.

```
CREATE OR REPLACE PROCEDURE fill_table_t4(k NUMBER) AS
 n NUMBER;
 BEGIN
  DELETE FROM t4;
   FOR n IN 1..k LOOP
    INSERT INTO t4 VALUES(n,to_char(n));
   END LOOP;
END;
```

2. Проанализируем время выполнения запросов на выборку в этих таблицах с использованием (столбец n) и без использования индекса (столбец t, если на какой-то таблице был индекс на этот столбец, удалите его).

Каждый запрос выполните несколько раз и усредните результаты. Обратите внимание, что самое большое время выполнения любого нового запроса будет при самом первом его исполнении.

Результаты можно свести в таблицу 4.1:

Таблица 4.1. Время выполнения запросов для различных таблиц

	Точечный по n	Диапазон по n	Точечный по t	Диапазон по t	Select count
t1					
t2					
t3					
t4					

Выполните соединение таблиц в запросах в разных сочетаниях (в том числе самосоединение таблицы t4). Убедитесь, что все соединения выполняются быстро.

Придумайте сами несколько запросов (не менее пяти), на ваш взгляд, ресурсоемких, проанализируйте время их выполнения.

Заключение

Тематика лабораторных работ в данном лабораторном практикуме была подобрана таким образом, чтобы в процессе их выполнения магистранты получили представление о различных аспектах распределенного хранения и обработки данных, познакомились с современной распределенной файловой системой DFS и возможностями распределенной обработки данных СУБД Oracle. Некоторые работы заключали в себе элементы исследования, необходимые в процессе обучения в магистратуре.

Внимательное и добросовестное выполнение лабораторных работ может помочь в усвоении других дисциплин учебной программы, в работе над магистерской диссертацией, в будущем (и настоящем) профессиональном росте. В дополнение к практикуму рекомендуется познакомиться с материалами для углубленного изучения некоторых тем данного курса, выложенными на сервере кафедры АВТ.

Библиографический список

1. Нортроп, Т. Проектирование сетевой инфраструктуры Windows Server® 2008: учебный курс Microsoft /Т. Нортроп, Дж.К. Макин — М.: «Русская редакция», 2011. - 592 с.
2. Дейт, К. Введение в системы баз данных: пер. с англ. /К.Дж. Дейт. 8-е издание. – М.: Вильямс, 2006. – 1326 с.
3. Ульман, Д. Введение в системы баз данных: пер. с англ. /Д.Ульман, Д.Уидом. – М.: Лори, 2000. – 512 с.
4. Грибер, М. Введение в SQL / М. Грибер. М:, Лори, 1996. – 379 с.
5. Базы данных: учебник для ВУЗов / Под ред.А.Д.Хомоненко — СПб: Корона принт, 2000. – 416 с.
6. Полякова, Л. Основы SQL. Курс лекций: учебное пособие / Л.Н. Полякова – М.: ИНТУИТ.РУ, 2004. – 368 с.
7. Ржеуцкая, С. Базы данных. Язык SQL: учеб. пособие / С.Ю. Ржеуцкая – Вологда: ВоГТУ, 2010. – 160 с.

Подписано в печать 30.10.2014. Усл. печ. л. 1,7. Тираж экз.
Печать офсетная. Бумага офисная. Заказ № _____

Отпечатано: РИО ВоГУ, г. Вологда, ул. С. Орлова, 6