Министерство образования Российской Федерации
Владимирский государственный университет

КОМПЛЕКСНАЯ ЗАЩИТА
ОБЪЕКТОВ ИНФОРМАТИЗАЦИИ
КНИГА 3

А.А. АЛЕКСАНДРОВ

АДМИНИСТРИРОВАНИЕ БАЗ ДАННЫХ

Учебное пособие

Владимир 2003

УДК 519.6 (075)
А46

Редактор серии кандидат технических наук М.Ю. Монахов

Рецензенты:

Кандидат технических наук, доцент
зав. кафедрой информатики и вычислительной техники
Владимирского государственного педагогического университета
Ю.А. Медведев

Доктор физико-математических наук, профессор
Владимирского государственного педагогического университета
Ю.А. Алхутов

Печатается по решению редакционно-издательского совета
Владимирского государственного университета

Александров А.А.
А46 Администрирование баз данных: Учеб. пособие / Владим. гос. ун-т. Владимир, 2003. 92 с. (Комплексная защита объектов информатизации. Кн. 3 / Под ред. М.Ю. Монахова).
ISBN 5-89368-453-2

Содержит материалы по различным аспектам администрирования современных СУБД, хранения, обработки и защиты информации на примере СУБД Oracle. В пособии рассматриваются системные объекты и процессы в СУБД, необходимые для эффективной работы.

Предназначено для студентов и аспирантов, уже знакомых с теорией реляционных СУБД.

УДК 519.6 (075)

ISBN 5-89368-453-2

ПРЕДИСЛОВИЕ РЕДАКТОРА

Данное учебное пособие содержит материал, необходимый для изучения и практического освоения средств обработки и защиты информации, имеющихся в СУБД Oracle.

Пособие предназначено для студентов и аспирантов, уже знакомых с основными концепциями реляционной алгебры, реляционных СУБД и языком SQL. Оно освещает широкий круг вопросов, связанных с установкой и тонкой настройкой СУБД Oracle, а также с основными операциями, обеспечивающими функционирование СУБД.

Следует отметить то, что автор особое внимание уделял особенностям, отличающим СУБД Oracle от остальных СУБД.

Описаны системные аспекты функционирования СУБД Oracle, что позволяет очень точно настраивать ее на максимальную производительность в каждых конкретных условиях. Приведены примеры настройки в зависимости от тех или иных требований к приложению. Рассмотрены также устройство самой СУБД Oracle и ее системные объекты, показано, как они взаимодействуют друг с другом. Даются рекомендации по их рациональному использованию.

Освещены вопросы импорта и экспорта данных, их защита и восстановление в случае потери. Описаны все необходимые для работы типы данных и даны рекомендации для их использования в тех или иных случаях.

Рассмотрены вопросы обеспечения целостности данных средствами самой СУБД.

В пособии приведены простые примеры с подробными комментариями, многие из которых показывают, как использовать средства, встроенные в СУБД, для ее изучения, анализа производительности и облегчения работы.

Таким образом, это учебное пособие будет особенно полезно тем, кто, уже освоив работу с той или иной СУБД, испытывает затруднения с освоением СУБД Oracle.

М.Ю. Монахов,
канд. техн. наук,
зав. кафедрой информатики
и защиты информации

Глава 1. УСТАНОВКА СИСТЕМЫ. СОЗДАНИЕ, УДАЛЕНИЕ И МОДИФИКАЦИЯ БАЗЫ ДАННЫХ

1. Эта глава описывает процедуру первоначальной установки и настройки СУБД Oracle, процедуры создания, удаления и модификации баз данных. Даны некоторые основные характеристики СУБД и баз данных, которые необходимо понимать для правильной установки и настройки системы.

УСТАНОВКА ORACLE 8i ENTERPRISE EDITION

2. Необходимо помнить, что Oracle в большинстве случаев устанавливается на ОС Windows только один раз. Удалить его полностью практически невозможно, поэтому при некорректной установке самым легким путем решения проблем является переустановка ОС.

3. Необходимо обязательно поставить американскую локаль (locale, Панель управления → Язык и стандарты).

4. Запустите файл setup.exe с диска 1. Нажмите Install/Deinstall Products. Для просмотра списка уже установленных частей Oracle нажмите Installed Products, для продолжения – Next. Появится диалог, в котором задается месторасположение файлов. Source – исходный путь, его трогать не надо. Destination – надо выбрать имя и месторасположение домашнего каталога Oracle. Можно оставить все по умолчанию, но, если есть необходимость в нескольких домашних каталогах (что вряд ли) – можно задать путь и имя.

5. **Диалог типа установки.** Надо выбрать Enterprise Edition – то, что предлагается по умолчанию. Далее лучше выбрать «Типичную установку» (также предлагается по умолчанию): там есть все, что нужно для работы.

Могут появиться предупреждающие сообщения – надо со всем согласиться и везде нажать Next. Возможно, после первой потребуется одна или несколько перезагрузок в процессе инсталляции – это нормально, надо просто подождать.

6. **Начальная конфигурация (идентификационная информация).** Надо ввести уникальное глобальное имя БД Oracle, а также уникальный системный идентификатор. Каждая БД относится, по крайней мере,

к одному экземпляру Oracle (его системный идентификатор и нужен). По умолчанию этот идентификатор инсталлятор делает таким же, как и глобальное имя БД (лучше оставить его таким же, поскольку разницы никакой, а путаницы меньше). По этой же причине БД по умолчанию рекомендуется назвать ORCL (таким же будет и SID). После этого запускается основной процесс копирования файлов на диск.

7. Далее в автоматическом режиме запускается Net8 Configuration Assistant. Если произошла ошибка, надо смотреть в логе, который выводится в этом же окне ниже. В автоматическом режиме создается БД по умолчанию с именем ORCL, создаются пользователи: *INTERNAL* с паролем «oracle», *SYS* с паролем «change_on_install» и *SYSTEM* с паролем «manager». После этого надо нажать Exit – установка завершена.

8. **Проверка работоспособности установленного продукта**. В меню «Программы» надо выбрать пункт Oracle → OraHome81 → Network Administration → Net8 Easy Config. В появившемся окне выбрать радиокнопку Test, затем экземпляр ORCL из списка (скорее всего, будет выбран по умолчанию), нажать Next. Если в логе появится запись, то тест прошел успешно.

СОЗДАНИЕ, УДАЛЕНИЕ И МОДИФИКАЦИЯ БД

9. Создание, удаление и модификация БД выполняется с помощью утилиты Database Configuration Assistant, которую можно найти в меню Oracle → OraHome81 → Database Administration.

10. Для **модификации БД** необходимо выбрать требуемую БД из списка и предоставить пароль пользователя *INTERNAL*. Далее выбрать режим, в котором будет работать БД. Их может быть два: выделенный и разделяемый. В *выделенном режиме* для каждого клиентского соединения будет выделен индивидуальный ресурс для обслуживания только этого соединения. Этот режим следует использовать, когда общее число подключений к БД планируется небольшим или когда клиенты будут создавать долгие, с долго выполняющимися запросами подключения к БД. В *разделяемом режиме* клиенты разделяют общий выделенный пул ресурсов. Этот режим желательно использовать, когда с БД работают одновременно много пользователей, в этом случае работа сервера будет более эффективной.

В целом более предпочтительным представляется разделяемый режим, поэтому после создания БД по умолчанию в процессе инсталляции режим ее использования рекомендуется изменить.

11. В следующем экране мастера можно настроить *параметры соединения* для БД. Здесь менять, как правило, ничего не нужно: все, что может потребовать изменения когда-либо, – это хост-имя машины, на которой работает БД, и максимальное количество соединений. Если количество соединений заранее известно и меньше предлагаемого по умолчанию, лучше ограничить максимальное количество соединений чуть меньшим числом: это уменьшит потребляемые сервером ресурсы.

12. На следующем экране мастера можно настроить параметры запуска серверных процессов, но делать это неопытным пользователям не рекомендуется, поскольку значения по умолчанию почти всегда работают хорошо.

13. **Создание БД** возможно в двух режимах – стандартном и пользовательском. В большинстве случаев стандартного режима будет достаточно.

На первом экране мастера *стандартного режима* будет предложение создать новую БД, взяв существующую с компакт-диска Oracle, или создать новые файлы БД. Далее (как и на первом экране мастера пользовательской настройки) будет предложено выбрать окружение (цель), в котором будет работать БД:

- OLTP-окружение – много пользователей осуществляют одновременный конкурентный доступ к транзакциям, которые читают и изменяют данные. Быстрый отклик системы и надежность в этом случае критичны;
- DSS-окружение – это когда небольшое количество пользователей выполняют сложные запросы, и полное использование ресурсов сервера требуется для минимизации времени обработки запроса. Этот режим лучше всего подходит для приложений, связанных с анализом данных или с созданием отчетов, которые в основном читают данные.

Следует заметить, что чаще всего встречаются гибридные приложения, которые нельзя отнести к той или иной группе. Для таких приложений лучше выбирать предлагаемый по умолчанию гибридный вариант.

14. Далее предлагается выбрать приблизительное число одновременно работающих пользователей. Это число ни в коем случае не означает, что большее количество пользователей не сможет работать одновременно. Речь здесь идет о точной и более оптимальной настройке. Это число лучше сделать близким к реальному, поскольку в этом случае скорость работы и использование системных ресурсов будут оптимальными.

15. Затем следует выбрать компоненты, которые будут присутствовать в БД (лучше оставить все по умолчанию), а также указать SID имя БД. После этого предлагается либо создать БД сразу, либо сохранить скрипт для создания БД в пакетный файл, который можно выполнить позднее.

16. **Структура каталогов** в деталях различается для разных версий Oracle, работающих под управлением разных ОС, поэтому здесь приведены общие тенденции распределения файлов:

- *BIN* – в данном подкаталоге находятся выполняемые программы и библиотеки Oracle;
- *DBS* – в данном подкаталоге находятся по умолчанию файлы параметров (init.ora);
- *DATABASE* (или *ORADATA*) – место расположения файлов баз данных по умолчанию;
- *NETWORK* – один из самых важных каталогов, в нем хранятся файлы конфигурации и sql-скрипты, связанные с сетевыми операциями.

В большинстве каталогов есть подкаталоги *ADMIN*, в которых хранятся настройки и sql-скрипты для административных целей. Также в большинстве каталогов есть подкаталоги *TRACE* (*DUMP*, *AUDIT*), предназначенные для трассировки.

ЗАПУСК И ОСТАНОВКА ЭКЗЕМПЛЯРОВ.

17. Для запуска и останова экземпляров существует специальная утилита – svrmgrl. Запустить экземпляр можно в нескольких режимах: запустить экземпляр без подмонтирования БД; запустить экземпляр и подмонтировать БД, но оставить ее закрытой; запустить экземпляр, подмонтировать и открыть БД как в общедоступном (для всех пользователей), так и в ограниченном (для администраторов).

Чтобы запустить экземпляр, надо подсоединиться как администратор, для чего набрать, зайдя в svrmgrl:

```
connect system/manager as sysdba
```

18. Далее надо воспользоваться командой startup, основной синтаксис которой следующий:

STARTUP [FORCE] [PFILE=FILESPEC] [EXCLUSIVE | SHARED]
[MOUNT DBNAME | OPEN DBNAME] [NOMOUNT]

Опция *NOMOUNT* позволяет запустить экземпляр без подмонтирования БД. Эта операция относительно редка и применяется в основном при создании БД (если создание происходит не с помощью мастера).

19. Опция *MOUNT* позволяет подмонтировать БД без ее открытия. Это может понадобиться в следующих случаях: при переименовании файлов данных; добавлении, удалении или переименовании сегментов отката; включении или выключении некоторых опций сегментов отката; выполнении полного восстановления БД.

20. Опция *OPEN* позволяет подмонтировать БД и открыть ее в обычном режиме, чтобы БД была доступна всем пользователям.

Ограниченный режим запуска

21. Для ограниченного режима запуска используется команда *STARTUP RESTRICT*. Другой способ – изменить параметры системы так:

ALTER SYSTEM <ENABLE | DISABLE> RESTRICTED SESSION

При этом БД будет подмонтирована и открыта, но доступ к ней смогут получить лишь администраторы. Это делается в основном для проведения работ по изменению структуры БД (например для перестройки индексов и т.п.); произведения экспорта или импорта данных; загрузки данных из текстового файла; для временного ограничения доступа обычных пользователей по тем или иным соображениям. Обычно к БД могут подсоединяться пользователи, у которых есть привилегия *CREATE SESSION*. К БД, открытой в ограниченном режиме, могут получить доступ только пользователи, у которых есть еще и привилегия *RESTRICTED SESSION*.

22. В некоторых, нетипичных, случаях может потребоваться применить опцию *FORCE*. Применение ее крайне нежелательно, но неизбежно в следующих случаях: если не удается остановить экземпляр с помощью команд *SHUTDOWN NORMAL* или *SHUTDOWN IMMEDIATE* либо когда не удается запустить экземпляр ни в одном из режимов. Если экземпляр запущен, команда *STARTUP FORCE* выполняет сначала закрытие экземпляра в режиме *ABORT*.

Остановка экземпляра

23. Остановка экземпляра производится с помощью команды *SHUTDOWN* с одной из опций:

- Опция *NORMAL* наиболее предпочтительна, поскольку при ее применении производятся следующие действия: ни одному новому пользователю не разрешается подсоединяться к БД, прежде чем будет остановлен экземпляр, ожидается завершение всех активных пользовательских соединений, и последующий запуск экземпляра не потребует никаких специальных действий по восстановлению.
- Опция *IMMEDIATE* не ждет завершения всех пользовательских сеансов, а сначала откатывает все неподтвержденные транзакции (в случае, если существовали достаточно длинные неподтвержденные транзакции, откат может потребовать весьма значительного времени), а потом прерывает все открытые сеансы.
- Опция *ABORT* позволяет мгновенно остановить работу экземпляра. Однако ее следует применять только в следующих случаях: БД или приложения ведут себя странно (неконтролируемо) и ни один другой метод остановки не работает, нужно немедленно остановить работу БД (например, известно, что через 30 секунд отключат электропитание), при запуске экземпляра есть проблемы. При такой остановке все клиентские запросы и сеансы прерываются немедленно, неподтвержденные транзакции не откатываются.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Как установить Oracle и проверить его работоспособность после установки?
2. Как создать новую БД?
3. Какие бывают режимы создания и модификации БД?
4. Какова структура каталогов Oracle? Назовите назначение основных каталогов.
5. Как запустить и остановить экземпляр? Какие возможности запуска и остановки экземпляров существуют?

Глава 2. ОСНОВНЫЕ СИСТЕМНЫЕ ФАЙЛЫ БД

1. Системные объекты БД имеют отношение к внутренним функциям управления БД и недоступны пользователям напрямую. Они создаются и настраиваются администратором БД. В их число входят: файлы параметров экземпляра, управляющие файлы, оперативные и архивные файлы журналов регистрации транзакций, файлы трассировки.

ФАЙЛЫ ПАРАМЕТРОВ ЭКЗЕМПЛЯРА

2. *Экземпляр* (часто называют «инстанс», от английского instance) – это набор фоновых процессов и структур памяти, используемых Oracle. Каждый экземпляр должен иметь *файл параметров*, имя которого, как правило, имеет вид init<InstanceName>.ora. После запуска экземпляра файл параметров не используется; таким образом, для того чтобы изменения в файле параметров вступили в силу, надо перезапускать экземпляр. Это обычный текстовый файл, параметры в нем можно указывать в любом порядке и в любом регистре (конечно, в UNIX-системах регистр в именах файлов имеет значение!). Многие файлы параметров имеют минимальное и максимальное значения. Если значение, установленное в файле параметров, выходит за пределы допустимого диапазона – экземпляр не запустится.

3. *Единственным* параметром, который обязательно должен присутствовать в файле параметров, является параметр *CONTROL_FILES*, который задает *один или несколько управляющих файлов*, предназначенных для этого экземпляра. Управляющие файлы используются во время запуска экземпляра для определения имени и места нахождения базы данных и файлов журнала обновлений.

4. Этот параметр определяет *идентификатор базы данных* длиной до восьми символов. В случае, когда он специфицирован, он должен соответствовать имени, которое было указано в предложении *CREATE DATABASE* (или в мастере). Хотя использование параметра *DB_NAME* необязательно, обычно он должен быть установлен перед вызовом *CREATE DATABASE*, и это предложение должно ссылаться на него. В имени базы данных допустимы следующие символы: буквы, цифры, подчеркивание (_), решетка (#), доллар ($). Любые иные символы недействительны. Двойные кавычки уда-

ляются перед обработкой имени базы данных. Они могут окружать само имя, но не могут окружать отдельные символы внутри имени. Строчные буквы рассматриваются как эквивалентные им прописные.

5. Параметр *DB_NAME.*

Умалчиваемое значение: пусто
Интервал значений: любое действительное имя базы данных

Этот параметр определяет *число блоков базы данных, кешируемых в памяти в SGA* (один блок равен одному буферу). Он является наиболее существенным из определяющих размер SGA и производительность базы данных. *Преимущество* высокого значения в том, что, когда пользователю необходим блок базы данных, этот блок с большей вероятностью находится в памяти, что сокращает ввод-вывод. *Недостаток* высокого значения в том, что потребляется больше памяти. Размер каждого буфера равен значению параметра *DB_BLOCK_SIZE*. В том случае, если значение *DB_BLOCK_BUFFERS* мало, пользователи не получают достаточного количества оперативной памяти для эффективного функционирования своих приложений. Если же значение *DB_BLOCK_BUFFERS* велико, система начинает активно использовать swap и может произойти остановка.

6. Параметр *DB_BLOCK_BUFFERS.*

Умалчиваемое значение: 32 буфера
Интервал значений: 4 или больше

Общее правило гласит, что лучше установить размер *DB_BLOCK_BUFFERS примерно в 25 % от общего объема памяти*. Однако такая установка обязательно должна учитывать и число работающих одновременно пользователей, а также то, выделена или нет машина исключительно под Oracle.

7. Существует методика для определения *оптимального* использования памяти и достаточного количества буферов блоков данных, основанная на вычислении так называемого *коэффициента попадания по чтению*. Если коэффициент попадания по чтению (read hit ratio) меньше 95 %, следует увеличить значение параметра *DB_BLOCK_BUFFERS*, чтобы повысить производительность. Однако следует иметь в виду, что, если коэффициент близок к 100 %, а число операций чтения исчисляется в миллионах, очень высока вероятность того, что исполняемое предложение построено

крайне неоптимально. Получить требуемые числа можно из *системной таблицы статистики* (об этих таблицах – ниже) с помощью следующего запроса:

```
SELECT name, value FROM v$sysstat WHERE name
IN ('db blocks get', 'consistent gets', 'phisical reads');
```

8. В результате будут возвращены три значения: db blocks get (число чтений буферов блоков), consistent gets (число логических операций чтения в режиме целостного чтения), phisical reads (число операций физического чтения). На основании этих чисел можно вычислить коэффициент попадания:

```
Коэффициент попадания = 1-физические/логические чтения*100 %
```

9. Следующий по значимости параметр *SHARED_POOL_SIZE*.

Умалчиваемое значение: 3.5 мегабайт
Интервал значений: 300 килобайт … зависит от ОС

Данный параметр управляет размером разделяемого пула, в байтах. Разделяемый пул содержит разделяемые откомпилированные и вызванные каким-либо пользователем запросы, курсоры и хранимые процедуры. Большие значения улучшают производительность в многопользовательских системах. Меньшие значения экономят память. Этот параметр задает размер памяти для размещения библиотек и кеша словаря данных.

10. Для данного параметра также существует методика расчета *оптимального значения*, заключающаяся в определении *коэффициента пропуска кеша словаря данных*. Если величина коэффициента, вычисляемого по приведенному ниже запросу, больше 15 %, следует увеличить значение параметра *SHARED_POOL_SIZE* в файле *INIT.ORA*.

```
SELECT SUM(gets), SUM(getmisses),
SUM(getmisses)/SUM(gets) «Ratio of Misses»
FROM v$rowchache;
```

11. Как правило, для получения *оптимальных результатов* значение этого параметра надо установить в примерно от 50 % до 75 % от принятого значения *DB_BLOCK_BUFFERS*.

12. Параметр *DB_FILES*.

Умалчиваемое значение: зависит от операционной системы
Интервал значений: минимум: *MAXDATAFILES* для монтируемой базы данных
максимум: зависит от ОС

Данный параметр определяет *максимальное число* файлов базы данных, которые могут быть открыты во время выполнения для этой базы данных. Уменьшать это значение имеет смысл только в том случае, если необходима память SGA, а появления новых файлов базы данных не предвидится.

Параметр *DB_FILES аналогичен* аргументу *MAXDATAFILES* для предложения *CREATE DATABASE*, который устанавливает абсолютное максимальное число файлов данных во время создания базы данных.

13. Параметр *DB_FILE_MULTIBLOCK_READ_COUNT***.**

Умалчиваемое значение: зависит от операционной системы
Интервал значений: зависит от операционной системы

Данный параметр используется при *многоблочных операциях ввода-вывода*. Он задает ***максимальное число блоков***, считываемых за одну операцию при последовательном просмотре. Умалчиваемое значение – функция от *DB_BLOCK_BUFFERS* и *PROCESSES*. Имеют смысл значения в интервале от 4 до 16 или даже до 32. Действительный максимум параметра зависит от операционной системы.

14. Параметр *PROCESSES***.**

Умалчиваемое значение: 50
Интервал значений: 6 .. зависит от операционной системы

При *многопроцессной работе* этот параметр специфицирует *максимальное число пользовательских процессов* операционной системы, которые могут быть одновременно соединены с сервером. Это число должно включать 6 для фоновых процессов, плюс 1 для подключения; таким образом, значение этого параметра, установленное в 20, допускает 13 одновременно работающих пользователей.

15. Параметр *CONTROL_FILES***.**

Умалчиваемое значение: зависит от операционной системы
Интервал значений: от 1 до 8 имен файлов

Данный параметр должен описывать одно или более имен файлов, разделяемых запятыми. Oracle Corporation рекомендует использовать несколько файлов на разных устройствах.

УПРАВЛЯЮЩИЕ ФАЙЛЫ

16. Управляющий файл – самый важный компонент БД. В нем хранится вся информация о БД (о файлах данных, файлах журналов, наборе символов, статусах обновления файлов данных и т.д.). Большинство параметров, которые содержатся в управляющем файле, обновляются относительно редко. Формат управляющих файлов – бинарный, поэтому прочесть его содержимое без инструментов Oracle невозможно. Формируется управляющий файл одновременно с БД. При порче или отсутствии управляющего файла БД просто нельзя открыть. Поэтому был придуман специальный механизм размножения управляющих файлов. Рекомендуется всегда создавать не менее трех копий управляющих файлов на трех разных физических дисках.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Какие основные системные файлы БД вы знаете?
2. Какова роль файла параметров экземпляра?
3. Каковы основные параметры, которые можно задавать в файле параметров экземпляра? В чём их назначение?

Глава 3. ВНЕШНЯЯ ПАМЯТЬ СУБД

1. В этой главе речь пойдет об организации файлов самой базы данных. Здесь не рассматриваются управляющие файлы и журналы обновлений.

ТАБЛИЧНЫЕ ПРОСТРАНСТВА И ФАЙЛЫ БАЗЫ ДАННЫХ

2. Из соображений *управляемости, безопасности и производительности* база данных *логически* разделяется на одно или несколько *табличных пространств* – логических разделов базы данных, состоящих из одного или нескольких файлов базы данных. Файл базы данных всегда связан только с одним табличным пространством.

3. В каждой базе данных Oracle есть табличное пространство с именем *SYSTEM*, с которым связывается самый первый файл базы данных. В табличное пространство *SYSTEM* по умолчанию помещаются все объекты при создании базы данных. Самая простая конфигурация базы данных – один файл базы данных в табличном пространстве *SYSTEM*. Однако такая конфигурация малофункциональна и практически никогда не встречается.

4. Обычно создают много табличных пространств, чтобы разделить различные части базы данных *по месту их физического хранения или по функциональности*. Например, можно создать одно табличное пространство для таблиц, другое – для хранения индексов и т.д., и каждое из этих табличных пространств будет содержать один или несколько файлов базы данных, связанных с ним.

5. Когда создаются объекты (такие, как таблицы), которые хранятся в базе данных, можно явно задать табличное пространство для размещения объекта в составе оператора *CREATE* для объекта. В табличном пространстве *SYSTEM* в большинстве случаев следует хранить только системные таблицы, например tab$, col$, ind$ и fet$.

6. Такие объекты, как *синонимы и представления*, не занимают места в базе данных, кроме места в таблице словаря данных, где хранятся их определения наряду с определениями всех других типов объектов.

7. Табличные пространства могут быть *добавлены, удалены, переведены в автономное и оперативное состояния*. К ним можно «привязать» дополнительные файлы базы данных. Добавляя следующий файл к табличному пространству, вы увеличиваете размер этого пространства и, следовательно, общий размер самой базы данных.

8. Табличное пространство *SYSTEM* не может быть удалено; это уничтожило бы базу данных, поскольку там располагаются системные таблицы. Нельзя также перевести табличное пространство *SYSTEM* в автономный режим.

СЕГМЕНТЫ

9. Сегмент – общее название, обозначающее какой-либо объект, который занимает место в файлах базы данных. Например, бывают сегменты таблиц (сегменты данных), индексные сегменты, сегменты отката, временные сегменты и сегмент кеша (сегмент начальной загрузки). Сегмент использует определенное число *блоков Oracle*, которые находятся в одном табличном пространстве (хотя сами блоки могут принадлежать различным файлам, образующим табличное пространство).

ЭКСТЕНТЫ

10. *Внешняя память* для какого-либо объекта в базе данных выделяется порциями из определенного числа блоков. В файлах базы данных эти блоки должны быть смежными. *Группа смежных блоков* называется *экстентом*. Например, когда создается таблица с использованием установок по умолчанию, для самого первого экстента таблицы распределяются пять блоков Oracle (*начальный экстент*). По мере добавления и модификации строк в таблице эти пять блоков заполняются данными. Если последний блок заполнен и должны быть вставлены новые строки, база данных автоматически выделяет для таблицы новый набор блоков (пять блоков) и новые строки вставляются в новый набор блоков. Это выделение дополнительных областей хранения (*дополнительных экстентов*) продолжается до тех пор, пока не будет исчерпано свободное место в табличном пространстве. Таблица начинается с одного начального экстента, а затем выделяется другой (следующий) экстент.

11. После того как *экстент* выделен *сегменту* (таблице), эти блоки не могут использоваться другим объектом базы данных, например таблицей, даже если все записи этой таблицы удалены. Таблицу нужно *уничтожить (drop)* или *усечь (truncate)*, чтобы освободить внешнюю память, выделенную данной таблице. Исключение из этого правила – сегменты отката, которые могут *динамически* освобождать выделенную им память.

БЛОКИ ORACLE

12. Oracle «разбивает» файлы базы данных при их первоначальном создании на ***блоки Oracle***. В результате для программного обеспечения СУБД упрощается и унифицируется работа с файлами и чтение данных в области оперативной памяти.

Блоки Oracle – наименьшая единица внешней памяти. Увеличение размера блока Oracle может улучшить производительность; он задается только при создании базы данных.

Нельзя изменять размер блока Oracle после создания базы данных.

13. Эти блоки обычно имеют размер 1 Кб (значение по умолчанию для системы PC-DOS), 2 Кб (значение по умолчанию для большинства систем UNIX и VAX/VMS), 4 Кб (значение по умолчанию для мэйнфреймов типа IBM) или больше. Файл базы данных размером 50 Мб содержал бы 25600 блоков Oracle при размере блока в 2 Кб (50 Мб/2 Кб).

14. Размер блока должен быть *кратен* размеру блока файловой системы используемой операционной системы. Нужно отметить, что *не весь* блок доступен для хранения данных; Oracle занимает некоторое пространство под заголовок для управления содержимым блока. Этот заголовок блока имеет минимальный размер, но он может увеличиваться.

15. Когда база данных создается, она занимает часть блоков первого файла, а остальные блоки остаются свободными. В *словаре данных* Oracle ведет список свободных блоков для каждого файла данных в каждом табличном пространстве.

16. Блоки Oracle нумеруются *последовательно*, начиная с 1, для каждого файла базы данных. Два блока могут иметь одинаковый номер, если они находятся в различных файлах базы данных.

ROWID В ORACLE

17. *ROWID – уникальный* во всей базе данных *физический адрес* каждой записи каждой таблицы. Будучи однажды назначенным (при вставке записи в базу данных), он никогда не изменяется, пока не будет удалена запись или вся таблица.

18. *ROWID* состоит из следующих четырех компонентов, комбинация которых уникально идентифицирует физическое расположение записи:

- идентификатор объекта базы данных, владеющего этой записью;
- номер файла базы данных Oracle, который содержит блок с записью;
- номер блока Oracle, который содержит запись;
- номер записи внутри блока (поскольку каждый блок может содержать много записей).

19. СУБД использует *ROWID* в индексах для быстрого отбора записей с указанным значением ключа. Разработчики приложений также используют его в операторах SQL как быстрый способ обращения к записи, если известен ее *ROWID*.

СВОБОДНОЕ ПРОСТРАНСТВО И АВТОМАТИЧЕСКАЯ ОРГАНИЗАЦИЯ НЕПРЕРЫВНЫХ УЧАСТКОВ

20. Когда файл базы данных создается или добавляется к табличному пространству, все блоки этого файла *пусты*. По прошествии времени блоки файла базы данных или используются сегментом (таблицей), или остаются свободными. Oracle отслеживает свободные блоки файла в списке *словаря данных*. По мере создания и удаления таблиц свободное пространство становится *фрагментированным*; фрагменты находятся в различных частях файла базы данных. Когда свободные блоки распределены таким образом, Oracle не может автоматически собрать их вместе.

21. Если два фрагмента свободного пространства физически следуют один за другим (расположены *смежно*) в файле базы данных, два меньших фрагмента могут быть объединены в один больший, который регистрируется в списке свободного пространства. Это уменьшает затраты ресурсов, когда Oracle фактически нуждается в свободном пространстве (например, когда таблице нужно выделить следующий экстент). Такую работу автоматически выполняет фоновый процесс *SMON*.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое файлы БД и табличные пространства? Как они соотносятся друг с другом?
2. Для чего нужны табличные пространства?
3. Что такое сегменты и как они используются?
4. Что такое экстенты и как они используются?
5. Что такое блоки Oracle? Какова их роль?
6. Что такое *ROWID* и для чего он применяется?
7. Как происходит автоматическая организация непрерывных участков памяти?

Глава 4. СИСТЕМНЫЕ ОБЪЕКТЫ БАЗЫ ДАННЫХ

1. В этой главе рассматриваются некоторые системные объекты, поддерживающие работу архитектуры Oracle и содержащие информацию о структуре базы данных.

СЛОВАРЬ ДАННЫХ

2. Первыми таблицами, создаваемыми в любой базе данных, являются системные таблицы, или ***словарь данных*** Oracle. Эти таблицы относятся к первой учетной записи пользователя Oracle, которая создается автоматически для пользователя с именем *SYS*. Системные таблицы хранят информацию о структуре базы данных и объектов внутри нее, и Oracle обращается к ним, когда нуждается в информации о базе данных либо когда выполняет оператор *DDL* (Data Definition Language – язык определения данных), либо оператор *DML* (Data Manipulation Language – язык манипулирования данными). Эти таблицы никогда непосредственно не обновляются, однако обновление в них происходит *в фоновом режиме* всякий раз, когда выполняется оператор *DDL*.

3. Главные таблицы ***словаря данных*** содержат нормализованную информацию, которая является довольно трудной для восприятия человеком, так что в Oracle предусмотрен *набор представлений*, выдающих информацию о главных системных таблицах в более понятном виде. Просмотреть имена более чем 170 представлений в ***словаре данных*** можно с помощью команды

```
SELECT * FROM DICT;
```

4. Oracle запрашивает информацию из таблиц ***словаря данных*** для интерпретации любого оператора SQL. Информация кэшируется в области словаря данных совместно используемого пула SGA.

5. Поскольку первым создается табличное пространство *SYSTEM*, таблицы ***словаря данных*** записываются в файлы базы данных, связанные с этим табличным пространством.

СЕГМЕНТЫ ОТКАТА

6. Когда данные в Oracle изменяются, изменение должно быть или подтверждено, или отменено. Если изменение *отменяется* (выполняется откат), содержимое блоков данных восстанавливается в исходное состояние, существовавшее до изменения. ***Сегменты отката*** – это системные объекты, которые поддерживают этот процесс. Всякий раз, когда осуществляются какие-либо изменения в таблицах приложения или в системных таблицах, в ***сегмент отката*** автоматически помещается предыдущая версия изменяемых данных, так что старая версия данных всегда доступна, если требуется откат.

7. Другие пользователи при необходимости чтения данных, в то время как изменение не завершено, всегда имеют доступ к *прежней версии* из ***сегмента отката***. Им предоставляется *непротиворечивая по чтению* версия данных. После того как изменение *фиксируется*, доступной становится измененная версия данных.

8. ***Сегменты отката*** всегда принадлежат пользователю *SYS*, и никакой пользователь Oracle не может обратиться к ним с целью просмотра.

9. Они получают внешнюю память таким же образом, как другие сегменты, – экстентами. ***Сегменту отката***, однако, нужно первоначально распределить минимум *два экстента*.

10. Первый ***сегмент отката*** создается автоматически, когда создается база данных, имеет имя *SYSTEM* и использует табличное пространство *SYSTEM*.

11. Путаницу может вызвать тот факт, что в Oracle используется одно и то же имя для трех различных типов объектов: первое табличное пространство называется *SYSTEM*, первый ***сегмент отката*** называется *SYSTEM* и одна из первых создаваемых учетных записей Oracle также называется *SYSTEM*. Все это – различные типы объектов, и их не следует путать.

ВРЕМЕННЫЕ СЕГМЕНТЫ

12. ***Временные сегменты*** используют пространство в файлах базы данных, чтобы создать *временную рабочую область* для промежуточных стадий обработки SQL и для больших операций сортировки.

13. Oracle создает ***временные сегменты*** в процессе работы, и они автоматически удаляются, когда фоновый процесс SMON больше в них не нуждается. Если требуется только небольшая рабочая область, Oracle не создает ***временного сегмента***, но вместо этого как временная рабочая область используется часть памяти PGA (глобальная область программы).

14. Ниже приводятся операции, которые могут приводить к созданию ***временных сегментов*** Oracle:

- Создание индекса.
- Использование фраз *ORDER BY*, *DISTINCT* или *GROUP BY* в операторе *SELECT*.
- Использование операторов работы с множествами *UNION*, *INTERSECT*, *MINUS*.
- Создание соединений таблиц.
- Использование некоторых подзапросов.

15. Администратор базы данных может определять, в каких табличных пространствах будут располагаться ***временные сегменты*** для различных пользователей.

СЕГМЕНТ НАЧАЛЬНОЙ ЗАГРУЗКИ / (КЕША)

16. Сегмент начальной загрузки (или кеш-сегмент) – специальный тип объекта в базе данных, выполняющий начальную загрузку кеша *словаря данных* в область совместно используемого пула SGA.

17. Oracle использует кеш-сегмент только при запуске экземпляра и не обращается к нему вплоть до *перезапуска* экземпляра. Сегмент необходим, чтобы выполнить начальную загрузку кеша *словаря данных*, после чего занимаемая им память освобождается.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое словарь данных? Какова его роль в БД?
2. Что такое сегменты отката? Для чего они нужны и как ими пользоваться?
3. Что такое временные сегменты? Для чего они нужны и как ими пользоваться?
4. Что такое кеш-сегмент? Для чего он нужен и как им пользоваться?

Глава 5. ОБЪЕКТЫ БАЗЫ ДАННЫХ, ТИПЫ ДАННЫХ И ВСТРОЕННЫЕ ФУНКЦИИ СУБД

1. Объекты базы данных – это логические элементы, из которых она состоит. Они определенным образом взаимодействуют между собой.

ИМЕНА ОБЪЕКТОВ БАЗЫ ДАННЫХ

2. Имена объектов базы данных (таблиц, представлений, последовательностей, «снимков», пакетов, процедур и функций) должны содержать не более 30 символов и начинаться с буквы. После начальной буквы имя может содержать любые буквы, цифры и символы «$», «#» и «_», однако

полученное сочетание (имя) не может быть зарезервированным словом Oracle (см. приложение). Если же возникает необходимость использовать в имени любые символы кроме кавычек и (или) составлять его из нескольких слов, то такое имя надо *заключить в кавычки*.

СОЗДАНИЕ ПОЛЬЗОВАТЕЛЯ

3. Для создания пользователя нужно воспользоваться следующим SQL-скриптом:

```
CREATE USER AAA
IDENTIFIED BY AAA
DEFAULT TABLESPACE USERS
TEMPORARY TABLESPACE TEMP
/
COMMIT
/
```

4. Однако при попытке подсоединиться к БД с этим именем будет получена ошибка, поскольку для того, чтобы пользователь мог хотя бы присоединяться к БД, надо дать ему права на роль *CONNECT*:

```
GRANT CONNECT TO AAA
/
COMMIT
/
```

5. Тем не менее, сделать с этой ролью все равно практически ничего нельзя. Надо дать еще прав:

```
GRANT COMMENT ANY TABLE TO AAA
/
GRANT CREATE DATABASE LINK TO AAA
/
GRANT CREATE PROCEDURE TO AAA
/
GRANT CREATE SEQUENCE TO AAA
/
GRANT CREATE SESSION TO AAA
/
GRANT CREATE SYNONYM TO AAA
/
GRANT CREATE TRIGGER TO AAA
/
```

```
GRANT CREATE TYPE TO AAA
/
GRANT CREATE VIEW TO AAA
/
GRANT DELETE ANY TABLE TO AAA
/
GRANT EXECUTE ANY INDEXTYPE TO AAA
/
GRANT EXECUTE ANY LIBRARY TO AAA
/
GRANT EXECUTE ANY OPERATOR TO AAA
/
GRANT EXECUTE ANY PROCEDURE TO AAA
/
GRANT EXECUTE ANY TYPE TO AAA
/
GRANT SELECT ANY SEQUENCE TO AAA
/
GRANT SELECT ANY TABLE TO AAA
/
GRANT UPDATE ANY TABLE TO AAA
/
COMMIT
/
```

6. Права эти достаточно обширные, обычным пользователям давать столько прав не стоит.

Теперь, когда пользователь создан и у него достаточно прав, можно приступать к созданию таблиц. Однако перед тем как создавать таблицы, необходимо ознакомиться с типами данных, которыми оперируют объекты Oracle.

СИМВОЛЬНЫЕ ТИПЫ ДАННЫХ

7. Типы данных *CHAR* и *VARCHAR* хранят алфавитно-цифровые данные; в столбце одного из этих типов данных можно хранить любые символы. Символьные данные хранятся как строки символов, где байтовые значения соответствуют схеме кодирования символов (обычно называемой набором символов или кодовой страницей); набор символов базы данных устанавливается при создании базы данных и никогда не изменяется. При-

мерами наборов символов служат 7-битовый ASCII (американский стандартный код для обмена информацией), кодовая страница 500 набора символов EBCDIC (расширенный двоично-кодированный десятичный код обмена) или Japan Extended UNIX. Oracle поддерживает как однобайтовые, так и мультибайтовые схемы кодирования.

8. Так как Oracle дополняет пробелами значения в столбцах *CHAR* и не дополняет пробелами значения в столбцах *VARCHAR*, столбцы *VARCHAR* более экономны в смысле затрат памяти на хранение. По этой причине при полном просмотре большой таблицы, содержащей столбцы *VARCHAR*, необходимо прочитать меньше блоков, чем для аналогичной таблицы со столбцами *CHAR*. Если приложение часто выполняет полные просмотры больших таблиц, содержащих символьные данные, можно улучшить производительность, выбрав для таких данных тип данных *VARCHAR* вместо *CHAR*.

9. Однако производительность – не единственный фактор, который необходимо рассматривать при выборе между этими типами данных. Например, если нужно, чтобы Oracle игнорировал *хвостовые пробелы* при сравнении символьных значений, то надо хранить такие значения в столбцах *CHAR*.

Тип данных *CHAR*

10. Тип данных *CHAR* хранит строки *фиксированной* длины. При создании таблицы со столбцом *CHAR* для этого столбца задается длина (в байтах, а не в символах) от 1 до 255 (по умолчанию 1). Затем Oracle гарантирует соблюдение следующих правил:

- если входное значение коротко, оно дополняется пробелами до фиксированной длины;
- если входное значение слишком длинно, Oracle возвращает ошибку.

11. Oracle сравнивает значения *CHAR*, используя *дополняющую семантику* сравнения. Если сравниваемые значения имеют *разную длину*, то Oracle дополняет более короткое значение пробелами до равной длины. Если два значения отличаются лишь числом хвостовых пробелов, то они считаются *равными*.

Тип данных *VARCHAR*

12. Тип данных *VARCHAR* используется для хранения символьных строк *переменной длины*. При создании таблицы со столбцом *VARCHAR* для этого столбца задается максимальная длина (в байтах, а не в символах)

от 1 до 2000. Для каждой строки значение столбца *VARCHAR* записывается как поле переменной длины (если входное значение превышает максимальную длину столбца, Oracle возвращает ошибку). Oracle сравнивает значения *VARCHAR*, используя *недополняющую семантику* сравнения. Два значения считаются равными лишь тогда, когда они состоят *из одних и тех же* символов и имеют *одинаковую длину*.

Тип данных *NUMBER*

13. Тип данных *NUMBER* используется для хранения чисел с *фиксированной и плавающей* точками. Для этого типа данных гарантируется *переносимость* между любыми операционными системами, которые поддерживает Oracle, с точностью *до 38 цифр*. Следовательно, можно хранить числа в интервале от $1 \cdot 10^{-130}$ до $9.99..9 \cdot 10^{125}$.

Для числовых столбцов можно просто указать тип *NUMBER*, а можно указать еще и *точность* (общее число цифр), и *масштаб* (число цифр справа от десятичной точки).

Если точность не указана, столбец хранит значения *так, как они задаются*. Если не указан масштаб, он считается *нулевым*. Масштаб может принимать значения от -84 до 127. Однако в современных версиях Oracle используются типы *INTEGER* и *FLOAT*.

Тип данных *DATE*

14. Тип данных *DATE* запоминает год (включая век), месяц, день, часы, минуты и секунды. Oracle может хранить даты в диапазоне от 1 янв. 4712 г. до н.э. до 31 дек. 4712 г. н.э. Если в маске формата не указано *BC* (до н.э.), предполагается по умолчанию *AD* (*наша эра*).

Oracle использует для хранения дат собственный *внутренний формат*. Данные хранятся в фиксированных полях длиной семь байт, соответствующих веку, году, месяцу, дню, часу, минуте и секунде.

Стандартный формат даты Oracle для ввода и вывода имеет вид DD-MON-YY, например '13-NOV-92'. Этот умалчиваемый формат даты можно изменить для экземпляра с помощью параметра *NLS_DATE_FORMAT*. Его можно также изменить на время сессии пользователя с помощью предложения *ALTER SESSION*. Для ввода дат в формате, отличном от текущего умалчиваемого формата даты, можно также использовать функцию *TO_DATE* с маской формата, например:

```
TO_DATE ('November 13, 1992', 'Month DD, YYYY')
```

Если используется стандартный формат DD-MON-YY, то YY указывает год в 20-м веке (например, 31-DEC-92 означает 31 дек. 1992 г.).

15. *Время* хранится в 24-часовом формате HH:MM:SS. Если не введено значение времени, по умолчанию предполагается полночь (12:00:00 A.M.). Если вводится только порция, содержащая время, то за дату принимается *первый день текущего месяца*. Для того чтобы ввести в дату время, необходимо это указать в маске формата функции *TO_DATE*.

16. Для *сравнения дат*, содержащих время, необходимо использовать функцию *TRUNC*, если нужно проигнорировать время. Текущие системные дату и время можно получить с помощью функции *SYSDATE*.

АРИФМЕТИКА ДАТ

17. Арифметика дат Oracle учитывает аномалии исторически применяемых календарей. Например, при переходе с юлианского календаря на грегорианский календарь 15 окт. 1582 г. были потеряны предыдущие 10 дней (с 5 по 14 октября). Кроме того, год 0 не существует.

18. *Пропущенные даты* могут быть введены в базу данных, но они игнорируются в арифметике дат и рассматриваются как следующая «реальная» дата. Например, следующим днем за 04 окт. 1582 г. будет 15 окт. 1582 г., а следующим днем за 5 октября 1582 будет 16 окт. 1582 г.

ТИП ДАННЫХ *LONG*

19. Столбец, описанный как *LONG*, может содержать символьную строку переменной длины до двух гигабайт. Столбцы типа *LONG* имеют многие характеристики столбцов *VARCHAR*. Длина значений *LONG* лимитируется ОС и объемом памяти.

20. Тип данных *LONG* используется в ***словаре данных*** для хранения текста определений представлений.

ОГРАНИЧЕНИЯ НА ДАННЫЕ ТИПА *LONG* И *LONG RAW*

21. Хотя столбцы типа *LONG* (и *LONG RAW*; см. ниже) находят много различных применений, на их использование накладываются некоторые *ограничения*:

- только один столбец типа *LONG* допускается в таблице;
- столбцы *LONG* нельзя индексировать;

- столбцы *LONG* нельзя использовать в ограничениях целостности;
- столбцы *LONG* нельзя использовать в фразах *WHERE, GROUP BY, ORDER BY, CONNECT BY,* а также с оператором *DISTINCT* в предложениях *SELECT*;
- столбцы *LONG* нельзя использовать в функциях SQL (таких как *SUBSTR* или *INSTR*);
- столбцы *LONG* нельзя использовать в списке *SELECT* подзапроса или запросов, объединяемых операторами множеств (*UNION, UNION ALL, INTERSECT* или *MINUS*);
- столбцы *LONG* нельзя использовать в выражениях;
- нельзя ссылаться на столбцы *LONG* при создании таблицы с помощью запроса (*CREATE TABLE ... AS SELECT ...*) или при вставке в таблицу (представление) через запрос (*INSERT INTO ... SELECT ...*);
- переменная или аргумент программной единицы PL/SQL не могут быть объявлены с типом данных *LONG*.

22. Как правило, удобно при проектировании таблицы, содержащей данные *LONG* или *LONG RAW*, помещать каждый столбец *LONG* или *LONG RAW* в отдельную таблицу, отдельно от любых других связанных с ними данных, вместо того чтобы хранить столбец *LONG* или *LONG RAW* в общей таблице с другими данными. После этого можно связать обе таблицы ограничением ссылочной целостности. Такой подход позволит избежать *сканирования по данным LONG* или *LONG RAW* при выборке других данных.

ТИПЫ ДАННЫХ RAW И LONG RAW

23. Типы данных *RAW* и *LONG RAW* используются для данных, которые не должны ни интерпретироваться Oracle, ни преобразовываться при передаче данных между различными системами. Эти типы данных предназначены для двоичных данных или байтовых строк. Например, *LONG RAW* можно использовать для хранения графики, звука, документов или массивов двоичных данных; их интерпретация зависит от их использования.

24. *RAW* эквивалентен *VARCHAR*, а *LONG RAW* эквивалентен *LONG*, с тем исключением, что SQL*Net (который соединяет пользовательские сессии с экземпляром) и утилиты экспорта и импорта не выполняют преобразований при передаче данных *RAW* или *LONG RAW*. Напротив, SQL*Net и импорт/экспорт автоматически конвертируют данные *CHAR,*

VARCHAR и *LONG* между набором символов базы данных и набором символов сессии пользователя (установленным параметром *NLS_LANGUAGE* или командой *ALTER SESSION*), если эти наборы символов различны.

Данные *LONG RAW* не могут индексироваться, а *RAW* – могут.

ROWIDы И ТИП ДАННЫХ ROWID

25. Каждая таблица в базе данных Oracle внутренне имеет псевдополе с именем *ROWID*; это псевдополе не видно при выдаче структуры таблицы с помощью предложения *SELECT * FROM ...* или предложения *DESCRIBE* в SQL*Plus. Однако адрес каждой записи можно извлечь запросом SQL, используя ключевое слово *ROWID* как имя поля, например:

```
SELECT ROWID, ename FROM emp;
```

26. Нельзя устанавливать значение псевдополя *ROWID* в предложениях *INSERT* или *UPDATE*. Хотя к значениям псевдополя *ROWID* можно обращаться, как к другим полям таблицы (в списках полей *SELECT* и *WHERE*), эти значения не являются данными базы данных.

27. Некоторые характеристики *ROWIDов* могут также использоваться разработчиками приложений:

- *ROWIDы* дают самый быстрый доступ к конкретным записям;
- *ROWIDы* позволяют увидеть, как организована таблица;
- *ROWIDы* уникально идентифицируют записи в таблице.

28. Можно также создавать таблицы с записями, определенными с типом данных *ROWID*; например, если надо определить таблицу исключений с полем типа данных *ROWID*, чтобы запоминать *ROWIDы* тех записей базы данных, которые нарушают ограничения целостности. Поля, определенные с типом данных *ROWID*, ведут себя как обычные поля, их значения можно обновлять и т.п. Все значения в поле типа данных *ROWID* занимают шесть байт.

ЧИСЛОВЫЕ ФУНКЦИИ

Функция	*Возвращаемое значение*
ABS(n)	Абсолютное значение величины *n*.
CEIL(n)	Наименьшее целое, большее или равное *n*.
COS(n)	Косинус *n* (угла, выраженного в радианах).
COSH(n)	Гиперболический косинус *n*.

EXP(n)	*e* в степени *n.*
FLOOR(n)	Наибольшее целое, меньшее или равное *n.*
LN(n)	Натуральный логарифм *n*, где *n*>0.
LOG(m,n)	Логарифм *m* по основанию *n.*
MOD(m,n*)*	Остаток от деления *m* на *n.*
POWER(w,n)	*m в* степени *n.*
ROUND(n[,m])	*n,* округленное до *m* позиций после десятичной точки. По умолчанию *m* равно нулю.
SIGN(n)	Если *n*<0, то -1; *n*=0, то 0; *n*>0, то 1.
SIN(n)	Синус *n* (угла, выраженного в радианах).
SINHM	Гиперболический синус.
SQRT(n)	Квадратный корень от *n*. Если *n*<0, возвращает значение NULL.
TAN(n)	Тангенс *n* (угла, выраженного в радианах).
TANH(n) .	Гиперболический тангенс *n*.
TRUNC(n[,m])	*n,* усеченное до *m* позиций после десятичной точки. По умолчанию *m* равно нулю.

СИМВОЛЬНЫЕ ФУНКЦИИ, ВОЗВРАЩАЮЩИЕ СИМВОЛЬНЫЕ ЗНАЧЕНИЯ

Функция	*Возвращаемое значение*
CHR(n)	Символ с кодом *n*
CONCAT(char1,char2)	Конкатенация символьных строк *char1* и *char2*.
INITCAP(char)	Символьная строка *char*, первые буквы всех слов в которой преобразованы в прописные.
LOWER(char)	Символьная строка *char,* все буквы которой преобразованы в строчные.
LPAD(char1.n [,char2})	Символьная строка *char1,* которая дополняется слева последовательностью символов из *char2* так, чтобы общая длина строки стала равна *n.* Значение *char2* по умолчанию – один пробел. Если часть многобайтового символа не помещается в добавляемой строке, то конец строки заполняется пробелами.
LTRIM(char[,set])	Символьная строка *char,* в которой удалены все символы от начала вплоть до первого символа, которого нет в строке *set*. Значение set по умолчанию – один пробел.

NLS_INITCAP(char[,nls_sort])	Символьная строка *char,* в которой первые буквы всех слов преобразованы в прописные. Параметр *nls_sort* определяет последовательность сортировки.
NLS_LOWER(char[,nls_sort])	Символьная строка *char,* все буквы которой преобразованы в строчные. Параметр *nls-sort* определяет последовательность сортировки.
NLS_UPPER(char[,nls_sort])	Символьная строка *char,* все буквы которой преобразованы в прописные. Параметр *nls_sort* определяет последовательность сортировки.
REPLACE(char, search_string [,replacement_string])	Символьная строка *char,* в которой все фрагменты *search_string* заменены на *replacement_string*. Если параметр *replacement_string* не определен, все фрагменты *search-string* удаляются.
RPAD(char1.n[,char2])	Символьная строка *char1,* которая дополнена справа последовательностью символов из *char2* так, что общая длина строки равна *n*. Если часть многобайтового символа не помещается в добавляемой строке, то конец строки заполняется пробелами.
RTRIM(char[,set])	Символьная строка *char,* в которой удалены все символы справа вплоть до первого символа, которого нет в строке *set.* Значение параметра *set* по умолчанию – один пробел.
SOUNDEX(char)	Символьная строка, содержащая фонетическое представление для *char,* на английском языке.
SUBSTR(char,m[,n])	Фрагмент символьной строки *char,* начинающийся с символа *m,* длиной *n* символов (до конца строки, если параметр *n* не указан).
SUBSTRB(char,m[,n])	Фрагмент символьной строки *char,* начинающийся с символа *m,* длиной *n* байтов (до конца строки, если параметр *n* не указан).
TRANSLATE(char,from, to)	Символьная строка *char,* в которой все символы, встречающиеся в строке *from,* заменены на соответствующие символы из *to.*
UPPER(char)	Символьная строка *char,* в которой все буквы преобразованы в прописные.

СИМВОЛЬНЫЕ ФУНКЦИИ, ВОЗВРАЩАЮЩИЕ ЧИСЛОВЫЕ ЗНАЧЕНИЯ

Функция	*Возвращаемое значение*
ASCII(char)	Возвращает десятичный код первого символа строки *char* в кодировке, принятой в базе данных. (Код ASCII в системах, использующих кодировку ASCII). Возвращает значение первого байта многобайтового символа.
INSTR(char1.char2[,n[,m]])	Позиция первого символа *m*-го фрагмента строки *char1,* совпадающего со строкой *char2,* начиная с n-го символа. По умолчанию *n* и *m* равны 1. Номер символа отсчитывается от первого символа строки *char1,* даже когда *n*>1.
INSTRB(char1.char2[,n[,m]])	Позиция первого символа *m*-го фрагмента строки *char1,* совпадающего со строкой *char2,* начиная с *m*-го байта. По умолчанию *n* и *m* равны 1. Номер байта отсчитывается от первого символа строки *char1,* даже когда *n*>1.
LENGTH(char)	Длина строки *char* в символах.
LENGTHB(c/iar)	Длина строки *char* в байтах.
NLSSORT(char1,char2[,n[,m]])	Зависящее от национального языка значение, используемое при сортировке строки *char.*

ГРУППОВЫЕ ФУНКЦИИ

Функция	*Возвращаемое значение*
AVG([DISTINCT|ALL]n)	Среднее значение от *n*, нулевые значения опускаются.
COUNT([ALL]*)	Число строк, извлекаемых в запросе или подзапросе.
COUNT(IDISTINCT|ALL] expr)	Число строк, для которых e*xpr* принимает не пустое значение.

MAX([DISTINCT|ALL] e*xpr)*	Максимальное значение выражения e*xpr*.
MIN((DISTINCT|ALL] e*xpr)*	Минимальное значение выражения e*xpr*.
STDDEV([DISTINCT|ALL] n)	Стандартное отклонение величины *n*, нулевые значения опускаются.
SUM([DISTINCT|ALL] *n)*	Сумма значений *n*.
VARIANCE([DIST1NCT|ALL]n)	Дисперсия величины *n,* нулевые значения опускаются.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Каковы правила именования объектов БД?
2. Как создавать пользователей? Какие привилегии необходимы пользователям для работы?
3. Какие символьные типы данных вы знаете? Перечислите их характерные особенности.
4. Какие числовые типы данных вы знаете? Перечислите их характерные особенности.
5. Какие числовые встроенные функции вы знаете? Перечислите их характерные особенности.
6. Какие символьные встроенные функции вы знаете? Перечислите их характерные особенности.
7. Какие групповые функции вы знаете? Перечислите их характерные особенности.

Глава 6. СОЗДАНИЕ ТАБЛИЦ

1. Создавать таблицы можно с помощью команды SQL *CREATE TABLE*. Например, выдавая следующее предложение, пользователь AAA создает некластеризованную таблицу с именем TEST в своей схеме и сохраняет ее в табличном пространстве USERS:

```
CREATE TABLE aaa.test
  (id                  NUMBER PRIMARY KEY,
  name                 VARCHAR2(100) NOT NULL)
PCTFREE    10
PCTUSED    40
INITRANS   1
MAXTRANS   255
```

```
 TABLESPACE  users
 STORAGE   (
   INITIAL     57344
   NEXT         57344
   PCTINCREASE 25
   MINEXTENTS  1
   MAXEXTENTS  10
  )
/
COMMENT ON TABLE aaa.test1 IS 'Таблица про студентов'
/
COMMENT ON COLUMN aaa.test1.id IS 'Уникальный идентификатор'
/
COMMENT ON COLUMN aaa.test1.name IS 'Имя студента'
/
```

УПРАВЛЕНИЕ ИСПОЛЬЗОВАНИЕМ ПАМЯТИ ДЛЯ БЛОКОВ ДАННЫХ

2. Управлять блоком памяти можно путем использования параметров *PCTFREE* и *PCTUSED* для достижения следующих целей:

- повышения производительности записи и извлечения данных или индексов;
- уменьшения объема неиспользуемой памяти в блоках данных;
- уменьшения количества цепочек строк между блоками данных.

Параметр *PCTFREE*

3. По умолчанию значение параметра *PCTFREE* равно 10 %, можно задавать любое целое значение от 0 до 99 включительно, пока сумма *PCTFREE* и *PCTUSED* не превышает 100. (Если вы установите *PCTFREE* как 99, то Oracle будет помещать в каждый блок по меньшей мере одну запись, независимо от ее размера. Если записи очень малы, а блоки очень велики, может уместиться даже несколько записей).

4. *Низкое значение PCTFREE* даст следующие эффекты:

- меньше места будет зарезервировано для обновлений существующих записей таблицы;
- блок будет более полно заполнен вставками;

- может экономить память, так как все данные таблицы или индекса хранятся в меньшем количестве блоков (больше записей на один блок);
- увеличивает стоимость обработки, так как Oracle вынужден часто реорганизовывать блоки по мере заполнения их свободной памяти новыми или обновленными данными, что потенциально увеличивает стоимость обработки и требуемую память, если обновления записей приводят к росту их размера и, как следствие, к расщеплению их между блоками (поскольку в этом случае предложения *UPDATE*, *DELETE* и *SELECT* должны считывать больше блоков для данной записи, следуя по цепочке, связывающей ее куски).

5. *Высокое значение PCTFREE* даст следующие эффекты:
   - будет зарезервировано больше места для обновлений существующих записей таблицы;
   - может потребовать больше памяти для того же количества вставляемых данных (поскольку вставляет меньше записей на один блок);
   - уменьшает стоимость обработки, так как блоки редко требуют реорганизации своей свободной памяти;
   - может улучшить производительность обновлений, потому что Oracle не должен столь часто, как прежде, строить цепочки для кусков записей.

6. При установке *PCTFREE* необходимо понимать природу данных таблицы или индекса. Обновления могут приводить к росту строк. Новые значения могут иметь размер, отличный от размера заменяемых ими значений. Если имеют место много обновлений, при которых размер данных увеличивается, то *PCTFREE* следует увеличить; если обновления существенно не влияют на размеры строк, *PCTFREE* может быть низким.

7. Цель – найти удовлетворительный компромисс между плотной упаковкой данных (низкий *PCTFREE*, заполненные блоки) и хорошей производительностью обновлений (высокий *PCTFREE*, менее заполненные блоки).

8. *PCTFREE* также влияет на производительность запросов данного пользователя по таблицам, имеющим *неподтвержденные транзакции* других пользователей (т.е. по таблицам, одновременно обновляемым другими пользователями). Обеспечение *согласованности по чтению* может потребовать частой реорганизации свободной памяти в блоках, если свободные участки в этих блоках малы.

PCTFREE для индексов

9. Индексы редко требуют использования свободной памяти при обновлениях индексных данных. Поэтому для индекса обычно можно устанавливать весьма низкое значение *PCTFREE* (например 5 или ниже).

Параметр *PCTUSED*

10. Когда свободная память в блоке данных падает до *PCTFREE*, в этот блок не вставляются новые записи, пока процент занятой памяти не упадет ниже *PCTUSED*. Oracle старается удержать блок данных заполненным, по крайней мере, на *PCTUSED*. Это – процент памяти в блоке, свободной для данных после вычета накладных расходов из общей памяти блока.

11. Значение по умолчанию для *PCTUSED* равно 40 %; вы можете задавать любое целое значение от 0 до 99 включительно, пока сумма *PCTFREE* и *PCTUSED* не превышает 100.

12. Если принято решение явно задать значения *PCTUSED* и *PCTFREE*, необходимо принять во внимание следующие соображения:

- Сумма *PCTFREE* и *PCTUSED* не должна превышать 100.
- Если эта сумма меньше 100, то хорошим компромиссом между утилизацией памяти и производительностью ввода-вывода является случай, когда сумма *PCTFREE* и *PCTUSED* отличается от 100 на величину, равную проценту памяти в свободном блоке, занимаемому средней строкой. Например, предположим, что размер блока данных равен 2048 байтам; за минусом 100 байт накладных расходов это дает 1948 байт, доступных для данных. Если средняя строка требует 195 байт, или 10 % от 1948, то наилучший компромисс даст сумма *PCTFREE* и *PCTUSED*, равная 90 %.
- Если эта сумма равна 100, то Oracle пытается удерживать в блоке не больше чем *PCTFREE* свободной памяти и стоимость обработки будет максимальной.
- Фиксированные накладные расходы блока не включаются в вычисления *PCTUSED* и *PCTFREE*.
- Чем меньше разница между 100 и суммой *PCTUSED* и *PCTFREE* (скажем, при *PCTUSED*=75 и *PCTFREE*=20), тем выше *утилизация памяти* за счет некоторого повышения стоимости обработки.

Примеры выбора значений *PCTFREE* и *PCTUSED*

13. Следующие примеры иллюстрируют подбор значений *PCTFREE* и *PCTUSED* при заданных сценариях.

Пример 1

Сценарий: типичная работа включает предложения *UPDATE*, которые увеличивают размеры записей.

Установка: *PCTFREE* = 20, *PCTUSED* = 40.

Объяснение: *PCTFREE* установлен в 20, чтобы оставить достаточно места для записей, увеличивающихся в размере при обновлениях. *PCTUSED* установлен в 40, чтобы требовалось меньше обработки при высокой активности обновлений, т.е. для улучшения производительности.

Пример 2

Сценарий: типичная работа включает предложения *INSERT* и *DELETE*, а предложения *UPDATE* в среднем не увеличивают размеры записей.

Установка: *PCTFREE* = 5, *PCTUSED* = 60.

Объяснение: *PCTFREE* установлен в 5, так как большинство предложений *UPDATE* не увеличивают размеров записей. *PCTUSED* установлен в 60, так что память, освобождаемая предложениями *DELETE*, скоро начинает повторно использоваться, так что обработка минимизируется.

Пример 3

Сценарий: таблица очень велика, поэтому основной заботой является память. Типичная работа включает только читающие транзакции.

Установка: *PCTFREE* = 5, *PCTUSED* = 90.

Объяснение: *PCTFREE* установлен в 5, так как предложения *UPDATE* используются редко. *PCTUSED* установлен в 90, так что для хранения данных используется большая часть блока. Это значение *PCTUSED* уменьшает число блоков, требуемое для размещения всех данных таблицы, сокращает среднее число блоков, просматриваемых во время запросов, и тем самым увеличивает производительность запросов.

ПРИВИЛЕГИИ ДЛЯ СОЗДАНИЯ ТАБЛИЦ

14. Чтобы создать новую таблицу в своей схеме, пользователь должен иметь системную привилегию *CREATE TABLE*. Чтобы создать таблицу в *схеме другого пользователя*, нужно иметь системную привилегию *CREATE ANY TABLE*. Кроме того, владелец таблицы должен иметь достаточную квоту для табличного пространства, в котором содержится таблица, либо системную привилегию *UNLIMITED TABLESPACE*.

ИЗМЕНЕНИЕ ТАБЛИЦ. ПРИВИЛЕГИИ ДЛЯ ИХ ИЗМЕНЕНИЯ

15. Существуют следующие причины, которые могут потребовать изменения таблицы в базе данных Oracle:

- добавление или удаление одного или нескольких полей;
- добавление одного или нескольких ограничений целостности;
- модификация определения существующего поля (типа данных, длины, умалчиваемого значения или ограничения целостности *NOT NULL*);
- модификация параметров использования памяти блока данных таблицы (*PCTFREE, PCTUSED*);
- модификация характеристик записей транзакций (*INITRANS, MAXTRANS*);
- модификация параметров памяти (*NEXT, PCTINCREASE* и т.п.);
- включение или выключение ограничений целостности или триггеров, ассоциированных с таблицей;
- удаление ограничений целостности, ассоциированных с таблицей.

16. При изменении определений полей в таблице можно только *увеличить* длину существующего поля; уменьшить ее можно лишь в том случае, если таблица пуста. Более того, если администратор увеличивает длину поля с типом данных *CHAR*, он должен понимать, что эта операция может потребовать значительного времени и существенной дополнительной памяти, особенно если таблица содержит много записей. Причина в том, что значение *CHAR* в каждой записи должно быть дополнено пробелами до новой длины поля.

17. Чтобы изменить таблицу, используется команда *ALTER TABLE*. Нужно понимать следующие последствия изменения таблицы:

- Если к таблице добавляется новое поле, то изначально оно пусто.
- Можно добавить новое поле с ограничением *NOT NULL* лишь в том случае, если в таблице нет ни одной записи.
- Если представление или программная единица PL/SQL зависят от базовой таблицы, то её изменение может повлиять на зависимый объект и всегда делает его недействительным.

18. Чтобы изменить таблицу, либо она должна содержаться в схеме пользователя, либо он должен иметь объектную привилегию *ALTER* для этой таблицы или системную привилегию *ALTER ANY TABLE*.

УДАЛЕНИЕ ТАБЛИЦ. ПРИВИЛЕГИИ ДЛЯ ИХ УДАЛЕНИЯ

19. Чтобы ***удалить*** ненужную таблицу, используется команда *DROP TABLE*.

Если удаляемая таблица содержит первичный или уникальный ключ, на который ссылаются внешние ключи других таблиц, то можно одновременно с этой таблицей удалить ограничения *FOREIGN KEY* для порожденных таблиц, включив в команду *DROP TABLE* опцию *CASCADE CONSTRAINTS*.

20. Прежде чем удалять таблицу, необходимо помнить о следующих эффектах этого действия:

- Удаление таблицы приводит к удалению ее определения из словаря данных.
- Все записи таблицы необратимо теряются.
- Все индексы и триггеры, ассоциированные с таблицей, также удаляются.
- Все представления и программные единицы PL/SQL, зависимые от удаляемой таблицы, остаются, но становятся недействительными (непригодными для использования).
- Все синонимы удаленной таблицы остаются, но возвращают ошибку при обращении к ним.

21. Чтобы удалить таблицу, она либо должна содержаться в схеме пользователя, либо удаляющий должен иметь системную привилегию *DROP ANY TABLE*.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Как создавать таблицы?
2. Как правильно выбирать значение параметра *PCTFREE*?
3. Как правильно выбирать значение параметра *PCTUSED*?
4. Как изменять таблицы и какие привилегии для этого требуются?

Глава 7. ЦЕЛОСТНОСТЬ ДАННЫХ

1. Целостность данных связана с определением правил проверки достоверности данных (таких как «процентное значение должно находиться между 0 и 100»), гарантирующих, что недействительные данные не попадут в ваши таблицы. Этот контроль осуществляется в форме ограничений целостности и триггеров базы данных.

ОГРАНИЧЕНИЯ ЦЕЛОСТНОСТИ

2. **Ограничения целостности** устанавливают бизнес-правила *на уровне базы данных*, определяя набор проверок для таблиц системы. Эти проверки автоматически выполняются всякий раз, когда вызывается оператор вставки, модификации или удаления данных в таблице. Если какие-либо ограничения нарушены, операторы отменяются. Другие операторы транзакции остаются в отложенном состоянии и могут фиксироваться или отменяться согласно логике приложения.

Поскольку ограничения целостности проверяются на уровне базы данных, они выполняются независимо от того, откуда были инициированы операторы вставки, модификации или удаления. Определение проверок через эти ограничения также выполняется быстрее, чем при использовании SQL. Кроме того, информация, вводимая при объявлении ограничений, используется оптимизатором Oracle для принятия решений о том, как выполнить оператор для данной таблицы. Программный продукт Oracle Forms может также использовать ограничения для автоматической генерации кода в программах пользовательского интерфейса, чтобы обеспечить заблаговременный вывод предупреждений пользователю о каких-либо ошибках.

Для таблицы можно задавать следующие типы ограничений целостности: *NOT NULL*, *PRIMARY KEY*, *UNIQUE*, *FOREIGN KEY*, *CHECK* – и индексы.

NOT NULL (не пусто)

3. Ограничение *NOT NULL* устанавливается для поля, чтобы указать, что поле должно иметь значение в каждой записи, оно никогда не может быть *NULL* (пустым). По умолчанию все поля в таблице могут быть пустыми.

PRIMARY KEY (первичный ключ)

4. Ограничение *PRIMARY KEY* определяет поле или группу полей, которые можно использовать для уникальной идентификации записи. Никакие две записи в таблице не могут иметь одинаковые значения полей первичного ключа. Кроме того, поля первичного ключа должны всегда содержать значение. Другими словами, они – *NOT NULL*. Для таблицы может быть задан только один первичный ключ *PRIMARY KEY*.

UNIQUE (уникальный)

5. Ограничение *UNIQUE* определяет вторичный ключ для таблицы. Это поле или группа полей, которые можно использовать как другой способ уникальной идентификации записи. Никакие две записи не могут иметь одинаковые значения для поля или полей ключа *UNIQUE*. Хотя таблица не может иметь более одного первичного ключа, на нее можно наложить несколько ограничений *UNIQUE*.

Поля для ограничения *UNIQUE* не обязательно должны быть *NOT NULL* (хотя, как правило, так и есть). Если значения для каких-либо полей, которые формируют уникальное ограничение, пустые, ограничение не проверяется.

FOREIGN KEY (внешний ключ)

6. *FOREIGN KEY*, или ограничение ссылочной целостности, устанавливает отношение целостности между таблицами. Оно требует, чтобы поле или набор полей в одной таблице совпадали с первичным или вторичным ключом другой таблицы.

Ограничения *FOREIGN KEY* используются для установления родительско-дочерних отношений между таблицами. Можно даже использовать их, чтобы установить ограничения «ссылки на себя», обычно в ситуациях, когда иерархическая структура (например дерево) создается на записях, хранимых в одной таблице. Если какие-либо поля внешнего ключа пусты, ограничение вообще не проверяется. Поля внешнего ключа обычно объявляются как *NOT NULL*.

7. Можно указать серверу БД, что, когда родительская строка удаляется, удаление должно автоматически распространяться на дочерние строки. Подобное автоматическое удаление дочерних строк поддерживается, только если задана фраза *ON DELETE CASCADE* в конце оператора создания внешнего ключа. Однако, если изменяется значение ключа главной таблицы, записи потомка не обновляются автоматически, чтобы соответствовать новому ключу. Эту функциональность обычно реализуют с помощью триггеров.

Ограничение *CHECK*

8. Ограничение *CHECK* определяет логику дополнительной проверки, которая должна дать результат true (истина) для оператора вставки, модификации или удаления из таблицы. Дополнительная логика возвращает результат Boolean. Ограничение *CHECK* гарантирует, что значения в изменяемой записи удовлетворяют заданному набору проверок правильности. Синтаксис ограничения *CHECK* подобен синтаксису фразы *WHERE* оператора *SELECT*, однако здесь нельзя использовать подзапросы или поля, которые изменяются с течением времени (такие как *SYSDATE*). Можно также использовать триггеры базы данных, чтобы запрограммировать обработку, которую нельзя поместить в ограничения.

ИНДЕКСЫ

9. Ограничения *PRIMARY KEY* и *UNIQUE* автоматически создают индекс на полях, для которых они определены, если ограничение активизируется при создании таблицы. Если индекс уже существует на полях,

которые составляют ограничение *PRIMARY KEY* или *UNIQUE*, то используется именно этот индекс и Oracle не может создать новый. Oracle создает индексы, когда ограничение активизируется (что выполняется по умолчанию и при добавлении ограничения к таблице). Oracle удаляет индексы таблицы при отмене ограничения. Поэтому надо помнить, что активизация и отмена ограничения могут потребовать много времени и затрат ресурсов системы вследствие создания и удаления индексов.

Когда устанавливаются ограничения *FOREIGN KEY*, поля автоматически не индексируются. Поскольку поля внешнего ключа в операции соединения таблиц обычно участвуют вместе, на этих полях нужно создавать индексы вручную.

ТРИГГЕРЫ БАЗЫ ДАННЫХ

10. Триггер базы данных – это блок PL/SQL, который предназначен для автоматического выполнения при отработке оператора вставки, модификации и удаления из таблицы. Можно определить триггеры двух типов: запускающийся один раз для всего оператора и запускающийся для каждой вставляемой, изменяемой или удаляемой записи. С таблицей можно связать 12 событий, для которых можно определить триггеры базы данных, а для каждого из 12 событий – несколько триггеров базы данных.

Триггер базы данных может вызывать процедуры базы данных, которые также написаны на PL/SQL. В отличие от триггеров базы данных, процедуры в базе данных сохраняются в скомпилированной форме. По этой причине целесообразно поместить наиболее длинные сегменты кода в процедуру, а затем вызывать ее из триггера базы данных, нежели выполнять код из триггера.

11. В дополнение к реализации сложных бизнес-правил, проверок, ввода значений по умолчанию можно использовать триггеры базы данных, чтобы вставлять, модифицировать и удалять записи других таблиц. Примером такого использования является средство аудита, когда контрольный журнал автоматически создается в таблице аудита всякий раз, когда запись таблицы данных изменяется. Без триггеров базы данных эта функция могла быть реализована в программах внешнего интерфейса, которые проводят изменения в базе данных. Однако даже тот, кто сумеет обойти программы внешнего интерфейса (например, используя SQL*Plus), не сможет обойти установленные проверки и обработку.

В триггерах базы данных можно использовать операторы SQL, а в ограничениях нет.

По возможности ограничения лучше использовать для контроля: они более быстры, чем триггеры базы данных.

ПРИВИЛЕГИИ СИСТЕМНОГО УРОВНЯ

12. Каждый пользователь Oracle, определяемый в базе данных, может иметь одну или несколько из более чем 80 привилегий системного уровня. Эти привилегии очень тонко управляют правами выполнения команд SQL. Администратор базы данных назначает системные привилегии или непосредственно учетным записям пользователей Oracle, или ролям. Роли затем назначаются учетным записям Oracle.

Например, прежде чем создать триггер для таблицы (даже если пользователь является владельцем таблицы как пользователь Oracle), нужно иметь системную привилегию *CREATE TRIGGER*, назначенную учетной записи пользователя Oracle или роли, присвоенной учетной записи.

Привилегия *CREATE SESSION* – другая часто используемая привилегия системного уровня. Чтобы выполнить соединение с базой данных, учетная запись Oracle должна иметь привилегию системного уровня *CREATE SESSION*.

ПРИВИЛЕГИИ ОБЪЕКТНОГО УРОВНЯ

13. Привилегии объектного уровня обеспечивают возможность выполнить определенный тип действия (выбрать, вставить, модифицировать, удалить и т.д.) с указанным объектом. Владелец объекта имеет полный контроль над объектом и может выполнять любые действия с ним; он не обязан иметь привилегии объектного уровня. Фактически владелец объекта – пользователь Oracle, который может предоставлять привилегии объектного уровня другим пользователям.

Например, если пользователь, который владеет таблицей, желает, чтобы другой пользователь вставлял и выбирал строки из его таблицы (но не модифицировал или удалял), он предоставляет другому пользователю привилегии (объектного уровня) выборки и вставки для этой таблицы.

Вы можете предоставлять привилегии объектного уровня непосредственно пользователям или ролям, которые затем назначаются учетным записям пользователей Oracle.

ПОЛЬЗОВАТЕЛИ И РОЛИ

14. Роль – это тип объекта, который используется для упрощения управления привилегиями системного и объектного уровней. Вместо того, чтобы назначать привилегии непосредственно учетным записям пользователей, можно назначать привилегии ролям, которые затем назначаются пользователям.

Роли, по существу, – это группы привилегий системного и объектного уровня. Они делают управление привилегиями намного проще, так как можно один раз сконфигурировать привилегии для отдельных типов пользователей, а затем назначать эти привилегии ролям. Когда пользователь нуждается в какой-то группе привилегий, можно использовать единственную команду назначения роли, чтобы удовлетворить этого пользователя. Без ролей пришлось бы вводить по несколько команд для каждой из требуемых привилегий.

15. Кроме того, можно создавать различные роли с привилегиями, даже если еще нет учетных записей пользователей Oracle, которые нуждаются в этих ролях. Можно назначать роль другой роли, формируя их иерархию. При желании можно защитить роль паролем, который пользователь должен ввести для активизации роли.

Как уже говорилось, физическая база данных может содержать много учетных записей пользователей Oracle, которые защищены паролями. Вы должны ввести имя пользователя и пароль независимо от того, какой инструмент вы используете для получения доступа к базе данных. Роли – это не то же самое, что пользователи Oracle; вы не можете соединяться с базой данных, вводя имя роли и пароль.

АУДИТ

16. Механизм аудита Oracle поддерживает три типа контрольных журналов. Журнал привилегий прослеживает, какие системные привилегии используются. Журнал операторов отслеживает, какие операторы SQL используются для всех объектов. Журнал объектного уровня контролирует доступ к объектам.

УПРАВЛЕНИЕ ОГРАНИЧЕНИЯМИ ЦЕЛОСТНОСТИ *FOREIGN KEY*

17. Общая информация об определении, включении, выключении и удалении всех типов ограничений целостности дана выше. Ниже эта информация дополняется, фокусируясь на вопросах, касающихся непосредственно ограничений целостности *FOREIGN KEY*.

Определение ограничений целостности *FOREIGN KEY*

18. Следующие вопросы заслуживают внимания при определении ограничений целостности *FOREIGN KEY*.

Совпадение типов данных

19. При определении ограничений ссылочной целостности имена соответствующих полей дочерней и родительской таблиц не обязаны совпадать. Однако эти столбцы должны иметь одинаковые типы данных.

Составные внешние ключи

20. Так как внешние ключи ссылаются на первичные и уникальные ключи родительской таблицы, а соответствующие ограничения *PRIMARY KEY* и *UNIQUE* реализуются посредством индексов, составные внешние ключи ограничиваются 16 полями.

Подразумеваемая ссылка на первичный ключ

21. Если список полей не включен в опцию *REFERENCES* при определении внешнего ключа через ограничение *FOREIGN KEY*, то Oracle предполагает, что поле (или список полей) ссылается на первичный ключ указанной таблицы. Конечно, можно явно специфицировать в скобках поле или поля, адресуемые внешним ключом в родительской таблице. Oracle автоматически проверяет, что этот список столбцов ссылается на первичный или уникальный ключ родительской таблицы, а если это не так, возвращается ошибка.

Привилегии для ограничений целостности *FOREIGN KEY*

22. Чтобы создать ограничение *FOREIGN KEY*, его создатель должен иметь привилегированный доступ как к родительской, так и к дочерней таблице.

- *Родительская таблица*. Создатель ограничения ссылочной целостности должен владеть родительской таблицей либо иметь объектные привилегии *REFERENCES* по полям, составляющим родительский ключ родительской таблицы.
- *Дочерняя таблица*. Создатель ограничения ссылочной целостности должен иметь право создавать таблицы (т.е. системную привилегию *CREATE TABLE* или *CREATE ANY TABLE*) либо право изменять дочернюю таблицу (т.е. объектную привилегию *ALTER* для дочерней таблицы или системную привилегию *ALTER ANY TABLE*).

23. В обоих случаях необходимые привилегии *не могут* быть получены через роль, они должны быть явно назначены создателю ограничения.

Эти условия позволяют:

- владельцу порожденной таблицы – явно решать, какие ограничения включаются по его таблицам и кто может создавать ограничения по его таблицам;
- владельцу родительской таблицы – явно решать, разрешать ли определение внешних ключей, зависящих от первичных и уникальных ключей в его таблицах.

Задание ссылочных действий для внешних ключей Oracle

24. Задание ссылочных действий для внешних ключей Oracle позволяет вводить два различных типа действий ссылочной целостности, как специфицировано в определении ограничения *FOREIGN KEY*:

- *Отсутствие действия* запрещает обновление или удаление родительского ключа, если в дочерней таблице существует запись, которая ссылается на этот ключ. По умолчанию все ограничения *FOREIGN KEY* подразумевают отсутствие действия;

для специфицирования отсутствия действия не требуется ничего указывать в определении ограничения. Например:

```
CREATE TABLE emp ( . . . , FOREIGN KEY (deptno)
REFERENCES dept);
```

- *Действие ON DELETE CASCADE* разрешает удалять (но не обновлять) адресуемые данные в родительской таблице.

25. Когда удаляются адресуемые данные в родительской таблице, все строки в дочерней таблице, зависящие от удаляемого родительского ключа, также удаляются. Чтобы специфицировать это ссылочное действие, необходимо включить в определение ограничения *FOREIGN KEY* опцию *ON DELETE CASCADE*. Например:

```
CREATE TABLE emp ( . . . , FOREIGN KEY (deptno)
REFERENCES dept ON DELETE CASCADE);
```

Включение ограничений целостности *FOREIGN KEY*

26. Ограничение целостности *FOREIGN KEY* не может быть включено, если ограничение *UNIQUE* или *PRIMARY KEY*, определяющее адресуемый родительский ключ, не существует или не включено.

Вывод определений ограничений целостности

27. Словарь данных содержит следующие обзоры, представляющие интерес с точки зрения ограничений целостности:

ALL_CONSTRAINTS
ALL_CONS_COLUMNS
CONSTRAINT_COLUMNS
CONSTRAINT_DEFS
USER_CONSTRAINTS
USER_CONS_COLUMNS
USER_CROSS_REFS
DBA_CONSTRAINTS
DBA_CONS_COLUMNS
DBA_CROSS_REFS

ПРИМЕРЫ

28. Рассмотрим следующие предложения *CREATE TABLE*, которые определяют несколько ограничений целостности, и приведем примеры вывода определений этих ограничений:

```
CREATE TABLE dept (
   deptno   NUMBER(3) PRIMARY KEY,
   dname    VARCHAR2(15),
   loc    VARCHAR2(15)
          CONSTRAINT dname_ukey UNIQUE (dname, loc),
          CONSTRAINT loc_check1
           CHECK (loc IN ('NEW YORK', 'BOSTON', 'CHICAGO')));
CREATE TABLE emp (
   empno    NUMBER(5) PRIMARY KEY,
   ename    VARCHAR2(15) NOT NULL,
   job      VARCHAR2(10),
   mgr      NUMBER(5) CONSTRAINT mgr_fkey
            REFERENCES emp ON DELETE CASCADE,
   hiredate DATE,
   sal      NUMBER(7,2),
   comm     NUMBER(5,2),
   deptno   NUMBER(3) NOT NULL
          CONSTRAINT dept_fkey REFERENCES dept);
```

Пример 1. Вывод всех доступных ограничений

29. Следующий запрос выдает ограничения, которые определены по всем таблицам, доступным текущему пользователю:

```
SELECT constraint_name, constraint_type, table_name,      r_constraint_name
FROM user_constraints;
```

30. В общем случае имена некоторых ограничений могут быть как специфицированы пользователем, так и сгенерированы системой.

Каждый тип ограничения обозначается собственным символом в поле *CONSTRAINT_TYPE*. В приведенной ниже таблице даны обозначения для всех типов ограничений.

Тип ограничения	Символ
PRIMARY KEY	*P*
UNIQUE	*U*
FOREIGN KEY	*R*
CHECK, NOT NULL	*C*

31. Существует еще один тип ограничения, который обозначается в столбце *CONSTRAINT_TYPE* символом «V». Этот тип соответствует ограничениям, созданным опцией *WITH CHECK OPTION*.

Пример 2. Распознавание ограничений *NOT NULL* и *CHECK*

32. В предыдущем примере некоторые ограничения были выданы с типом ограничения «C». Чтобы выяснить, какие из этих ограничений по таблицам EMP и DEPT были определены как *NOT NULL*, а какие – как *CHECK*, можно выполнить следующий запрос:

```
SELECT constraint_name, search_condition
   FROM user_constraints
   WHERE (table_name = 'DEPT' OR table_name = 'EMP')
   AND constraint_type = 'C';
```

Пример 3. Вывод имен столбцов, составляющих ограничение целостности

33. Следующий запрос выдает все столбцы, составляющие ограничения, которые определены по всем таблицам, доступным текущему пользователю:

```
SELECT constraint_name, table_name, column_name
FROM user_cons_columns;
```

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Какие ограничения целостности можно применять в Oracle?
2. Роль индексов в поддержании целостности БД.
3. Роль триггеров в поддержании целостности БД.
4. Чем отличаются привилегии системного и объектного уровней?
5. Как работать с ролями пользователей?
6. Как использовать ограничения целостности и просматривать их?

Глава 8. ИМПОРТ И ЭКСПОРТ

1. Импорт и экспорт производятся в файлы специального внутреннего формата, которые невозможно отредактировать вручную. Импорт и экспорт применяются для переноса данных независимо от аппаратной и программной конфигураций. Файлы, сгенерированные более поздней версией экспорта, не могут быть импортированы. В то же самое время импорт может читать файлы экспорта, сгенерированные такой же или более старой версией экспорта.

ЭКСПОРТ

Режимы экспорта

2. Экспорт может производиться в нескольких режимах. Все пользователи могут производить экспорт в табличном и пользовательском режимах. Пользователь, у которого есть роль *EXP_FULL_DATABASE*, может производить экспорт во всех режимах. В зависимости от выбранного режима, экспортируются разные объекты.

3. Возможно, что перед использованием экспорта надо будет выполнить файл catexp.sql. Этот скрипт создает все необходимые для экспорта представления и предоставляет необходимые привилегии. Конечно, его достаточно выполнить только один раз для одной БД. Перед началом экспорта будет нелишне убедиться в наличии свободного места. Для того чтобы посчитать место, занимаемое таблицами, можно использовать следующее предложение:

```
SELECT SUM(Bytes) FROM User_Segments
WHERE Segment_Type = 'TABLE';
```

Экспорт производится с помощью следующей команды:

```
exp user/password PARFILE=file
```

4. При этом все параметры, которые можно использовать в файле параметров, можно использовать и в командной строке. Более того, если и в файле параметров, и в командной строке присутствуют одинаковые параметры, то действуют те, которые встречаются последними. С точки зрения безопасности предпочтительнее использование файла параметров, поскольку при этом реальная конфигурация не может быть узнана из истории командной строки. В файле параметров можно вставлять комментарии, для этого строку надо начинать со знака «#».

Параметры экспорта

BUFFER

Значение по умолчанию зависит от ОС. Указывает на размер буфера, используемого для извлечения записей. Если указать 0, то будет извлекаться по одной записи. Во многих случаях этот параметр увеличивают.

COMPRESS

Значение по умолчанию – Y. Указывает на то, как выделяется свободное место при импорте во время заполнения данных. При включенном параметре все данные по возможности пишутся подряд в один экстент.

CONSISTENT

Значение по умолчанию – N. Фактически во время экспорта делает *SET TRANSACTION READ ONLY*, чтобы гарантировать, что данные, видимые для утилиты экспорта, не будут изменены за время ее работы. Необходим этот параметр только тогда, когда экспортируемые данные действительно обновляются достаточно интенсивно.

CONSTRAINTS

Значение по умолчанию – Y. Указывает, экспортировать ли ограничения целостности вместе с таблицами.

DIRECT

Значение по умолчанию – N. Указывает, использовать ли традиционный или же прямой экспорт. Об их отличиях – ниже.

FEEDBACK

Значение по умолчанию – 0 (ноль). Указывает, что для данного числа экспортированных записей надо отображать точку. Удобно на больших объемах: позволяет увидеть, есть ли прогресс или нет.

FILE

Значение по умолчанию – expdat.dmp. Поскольку можно разбивать экспорт на несколько файлов, можно указывать несколько имен.

FILESIZE

Этот параметр указывает максимальный размер файла экспорта. Когда значение размера текущего файла достигает указанного в этом параметре, Экспорт берет следующее имя из списка *FILE*. Если имена кончились,

имя следующего файла будет запрошено в интерактивном режиме. Если этот параметр не указан (или его значение установлено в 0), Экспорт будет писать только в один файл, независимо от количества указанных в *FILE* файлов. Значение этого параметра можно устанавливать в других единицах, добавив к числу соответствующую букву (K, M, G).

FULL

Значение по умолчанию – N. Указывает, что экспорт должен производиться в режиме «Вся БД».

GRANTS

Значение по умолчанию – Y. Указывает, надо ли экспортировать права. В режиме «вся БД» для таблицы экспортируются все права, в режиме пользователя – только права, данные владельцем таблицы.

INCTYPE

Значения по умолчанию нет. Указывает тип инкрементального экспорта, и может принимать одно из следующих значений: *INCREMENTAL*, *COMPLETE*, *CUMULATIVE*.

INDEXES

Значение по умолчанию – Y. Указывает, нужно ли экспортировать индексы.

LOG

Значения по умолчанию нет. Указывает имя лог-файла, куда записываются информационные сообщения и сообщения об ошибках. Если имя файла указано, то эти сообщения не только записываются в файл, но так же, как раньше, показываются на экране.

OWNER

Значения по умолчанию нет. Указывает на список пользователей, чьи схемы будут полностью экспортированы. Если пользователь, производящий экспорт, является DBA, то список этот может состоять из нескольких пользователей.

QUERY

Значения по умолчанию нет. Параметр позволяет ограничивать экспорт в табличном режиме какой-то частью таблицы и представляет из себя

строку, содержащую предложение *WHERE*, которое будет применено ко всем таблицам из списка. При использовании этого параметра надо помнить, что применять его можно только в табличном режиме, получившееся предложение *WHERE* должно быть применимо ко всем перечисленным в списке таблицам, нельзя этот параметр использовать в режиме прямого экспорта; что из содержимого файла экспорта нет потом возможности узнать, использовался ли этот параметр при его создании.

ROWS

Значение по умолчанию – Y. Указывает, нужно ли экспортировать содержимое таблиц (т.е. данные).

TABLES

Значения по умолчанию нет. Перечисляет список таблиц, которые надо экспортировать.

Прямой и обычный экспорт

5. При обычном экспорте используется выборка с помощью оператора *SELECT*, и экспорт ведет себя точно так же, как и любой другой клиент. Прямой же экспорт во многих случаях оказывается гораздо быстрее, поскольку он минует SQL-слой и читает данные напрямую.

Инкрементальный экспорт

6. Система примерно такова. Сначала делается базовый, полный экспорт. Далее, с некоторой периодичностью, делаются достаточно частые инкрементальные экспорты и более редкие кумулятивные. Через какой-то промежуток времени все повторяется сначала, начиная с полного экспорта.

Инкрементальный экспорт экспортирует только таблицы, которые были изменены со времени последнего базового, кумулятивного или инкрементального экспорта, причем именно полные таблицы со всеми данными, а не только изменившиеся записи.

7. Кумулятивный экспорт экспортирует все таблицы, изменившиеся со времени последнего базового или кумулятивного экспорта. По сути, он собирает все инкрементальные экспорты в один файл, после чего они становятся ненужными.

8. Базовый экспорт эквивалентен полному с той лишь разницей, что он обновляет информацию в системных таблицах, в которых хранится информация о произведенных экспортах.

Кодировки при экспорте

9. При обычном экспорте утилита пишет данные в файл экспорта в кодировке, установленной для пользовательской сессии, перекодируя данные из БД, если это необходимо. Таким же образом утилита импорта конвертирует данные в другую кодировку, если это необходимо. В общем, надо следить за переменной окружения *NLS_LANG*.

Формат этой переменной таков:

NLS_LANG = язык_территория.кодировка

Например, для России это будет:

NLS_LANG = RUSSIAN_CIS.кодировка, где кодировка – одно из RU8PC866, CL8MSWIN1251, CL8KOI8R.

ИМПОРТ

10. Импорт производится командой imp. Специфика ее вызова и работы с ней совершенно такая же, как и у утилиты экспорта.

Есть возможность импортировать данные в уже существующие таблицы, с некоторыми ограничениями (нельзя добавлять *NOT NULL* поля, нельзя заменять тип поля на несовместимый). Однако утилита импорта имеет ряд дополнительных параметров, заслуживающих внимания:

COMMIT

Значение по умолчанию – N. Указывает, надо ли производить *COMMIT* после вставки каждого массива строк. По умолчанию импорт выполняет *COMMIT* после импортирования целой таблицы. Значение параметра лучше устанавливать в Y на больших импортах, поскольку это предотвращает сильный рост сегментов отката и улучшает производительность.

DESTROY

Значение по умолчанию – N. Указывает, использовать ли существующие файлы данных после удаления их содержимого. Пользоваться надо с осторожностью, поскольку файл экспорта содержит имена файлов данных и есть риск их перезаписать.

FROMUSER

Значения по умолчанию нет. Как правило, этот параметр используется совместно с параметром *TOUSER* утилиты экспорта.

IGNORE

Значение по умолчанию – N. Указывает, каким образом должны обрабатываться ошибки создания объектов. В случае, если указано значение Y, ошибки создания объектов будут игнорироваться. При этом надо помнить, что в случае ошибки существующий объект не замещается, а игнорируется. Следует также помнить, что игнорируются только ошибки *создания* объектов, а все остальные – нет.

INDEXFILE

Значение по умолчанию отсутствует. Параметр позволяет указать имя файла, в который будут записаны команды создания индексов. Если параметр *CONSTRAINTS* также установлен в Y, в этот файл будут записаны и ограничения целостности. Этот файл впоследствии можно отредактировать (например, чтобы изменить параметры распределения памяти и т.п.) и запустить.

TOUSER

Значения по умолчанию нет. Указывает список целевых пользователей. Для случая нескольких схем число входных (*FROMUSER*) и выходных (*TOUSER*) схем должно быть одинаковым.

ДЕФРАГМЕНТАЦИЯ БД

11. Бывает, что БД настолько фрагментирована, что работа с ней сильно затруднена. В этом случае поступают следующим образом: делают полный экспорт, останавливают БД, физически удаляют ее, затем создают новую пустую БД и делают полный импорт из файла экспорта.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Какие режимы экспорта вы знаете?
2. Перечислите основные параметры экспорта.
3. Какие виды экспорта вы знаете?
4. Назовите основные параметры импорта.

Глава 9. ФИЗИЧЕСКОЕ РЕЗЕРВНОЕ КОПИРОВАНИЕ И ВОССТАНОВЛЕНИЕ

1. В отличие от exp/imp, которые выполняют логическое резервное копирование, есть еще утилита для физического резервирования – rman (recovery manager). Режимов резервного копирования два – backup и image. В первом режиме данные сохраняются во внутреннем формате rman, доступном только ему. Во втором режиме создается файл, который может быть напрямую использован экземпляром и, следовательно, может быть подключен вручную. Преимуществом rman является то, что он, при возможности создания инкрементальных резервных копий, может записывать только измененные блоки данных.

2. В случае, если экземпляр упал в результате какой-либо ошибки, Oracle при старте пытается его восстановить автоматически. Однако, если повреждены файлы данных или управляющие файлы, автоматически он восстановить ничего не сможет.

ВОССТАНОВЛЕНИЕ ДАННЫХ

3. После того, как сбой носителя повредил базу данных, надо ответить на следующие два вопроса:

Какие операции восстановления возможны?

Ответ на первый вопрос определяется тем, архивирует ли база данных журналы. Если база данных работает в режиме *NOARCHIVELOG*, то обычно единственной возможностью восстановления после сбоя носителя является реставрация наиболее свежей полной копии базы данных и повторный ввод всех изменений, которые были выполнены после этой копии. (Если использовался экспорт в дополнение к регулярному копированию базы данных, то можно вместо этого реставрировать данные с помощью импорта.)

Если база данных работает в режиме *ARCHIVELOG*, то для восстановления поврежденной базы данных могут быть применены несколько различных операций восстановления.

Включить или выключить архивирование можно с помощью команды

```
ALTER DATABASE [NO]ARCHIVELOG;
```

Какие операции восстановления подходят для данного случая?

Если база данных работает в режиме *ARCHIVELOG*, то несколько различных операций восстановления могут быть применены для приведения поврежденной базы данных в согласованное состояние на момент, предшествующий сбою или ошибке пользователя.

СООБЩЕНИЯ, УКАЗЫВАЮЩИЕ НА СБОИ НОСИТЕЛЯ

4. Следующие сообщения указывают, что необходимо восстановление носителя:

```
ORA-00204 "error in reading control file 'ИМЯ'
(block НОМЕР,  #blocks ЧИСЛО)"

ORA-01113 "file ИМЯ needs recovery"

ORA-01168 "file ИМЯ: bad physical block size of
ЧИСЛО bytes expecting ЧИСЛО"

ORA-01178 "file ИМЯ created before last
CREATE CONTROLFILE,                cannot recreate"
```

ОПРЕДЕЛЕНИЕ ФАЙЛА, НУЖДАЮЩЕГОСЯ В ВОССТАНОВЛЕНИИ

5. Чтобы определить файл, нуждающийся в восстановлении, можно в большинстве случаев использовать таблицу *V$RECOVER_FILE*. Эта таблица показывает все файлы, для которых требуется восстановление, вместе с информацией о том, почему оно требуется.

Таблица *V$RECOVER_FILE* наиболее полезна, когда база данных закрыта, потому что при открытой базе данных она содержит информацию лишь об офлайновых файлах. Эта таблица бесполезна, если текущий используемый управляющий файл является реставрированной копией или заново создан после сбоя носителя. Реставрированный и пересозданный управляющий файл не содержит информации, необходимой Oracle для правильного заполнения таблицы *V$RECOVER_FILE*.

6. Например, следующий запрос выдает идентификационные номера файлов данных, требующих восстановления:

```
SELECT file#, online, error FROM v$recover_file;

FILE#      ONLINE     ERROR
-----------------------------------------------
  0014     ONLINE
  0018     ONLINE     FILE NOT FOUND
  0032     OFFLINE    OFFLINE NORMAL
...
```

7. Чтобы определить имя файла по его идентификационному номеру, можно использовать обзор словаря данных *V$DATA_FILE*, который содержит столбцы *FILE#* и *NAME*.

ВОССТАНОВЛЕНИЕ ФАЙЛА ДАННЫХ БЕЗ КОПИИ

8. Если файл данных поврежден, а резервной копии этого файла нет, то его все же можно восстановить, при условии, что доступны все файлы журнала, записанные с момента первоначального создания файла данных, и что управляющий файл содержит имя поврежденного файла (т.е. управляющий файл либо текущий, либо реставрирован из копии, взятой после того, как этот файл данных был добавлен к базе данных).

9. Для этого нужно использовать фразу *CREATE DATAFILE* команды *ALTER DATABASE*, чтобы создать новый, пустой файл данных, заменяющий поврежденный файл, для которого нет резервной копии. Например, предположим, что файл данных «disk:users» был поврежден, а резервной копии не существует. Следующее предложение пересоздает оригинальный файл данных (такого же размера) на диске 2:

```
ALTER DATABASE CREATE DATAFILE 'disk:user';
```

Замечание: старый файл данных неявно переименовывается в новый файл данных, когда выдается предложение *ALTER DATABASE CREATE DATAFILE*.

10. Это предложение заставляет Oracle создать пустой файл, совпадающий с потерянным файлом. Oracle получает информацию об этом файле, включая его размер, из управляющего файла и словаря данных. Затем нужно выполнить восстановление носителя на этом пустом файле данных. Все архивированные файлы журнала, записанные после создания оригинального файла данных, должны быть смонтированы

и применены к новой, пустой версии файла данных во время восстановления. Если база данных была создана в режиме *NOARCHIVELOG*, то первоначальные файлы данных табличного пространства *SYSTEM* не могут быть реставрированы с помощью команды *ALTER DATABASE CREATE DATAFILE*, так как необходимые архивные журналы недоступны.

РЕСТАВРАЦИЯ НЕОБХОДИМЫХ АРХИВНЫХ ФАЙЛОВ ЖУРНАЛА

11. Все архивные файлы журнала повторения, требуемые для восстановления носителя, которое вы собираетесь выполнять, должны быть заблаговременно сброшены на диск, чтобы они были доступны для Oracle.

12. Чтобы определить, какие архивные файлы журнала нужны, часто можно использовать таблицы *V$LOG_HISTORY* и *V$RECOVERY_LOG*.

V$LOG_HISTORY перечисляет все архивированные журналы, включая имена, которые они должны иметь согласно текущей схеме формирования имен файлов архива (как установлено параметром *LOG_ARCHIVE_FORMAT*). *V$RECOVERY_LOG* перечисляет лишь те архивированные журналы, которые, по мнению Oracle, требуются для выполнения восстановления; здесь также представлены имена архивных файлов в соответствии с форматом *LOG_ARCHIVE_FORMAT*.

13. Если свободной памяти достаточно, можно реставрировать все требуемые архивные файлы журнала в место, которое специфицировано текущим значением параметра *LOG_ARCHIVE_DEST*. Таким путем Oracle будет автоматически находить нужный архивный файл журнала во время восстановления носителя. Если в каталоге, указываемом параметром *LOG_ARCHIVE_DEST*, не хватает места, можно реставрировать некоторые или все архивные файлы журнала на любой диск, к которому Oracle имеет доступ. В этом случае нужно будет специфицировать местоположение архивных файлов журнала перед восстановлением носителя или во время него.

14. После того как архивный файл журнала применен, можно удалить его реставрированную копию с диска, чтобы освободить место. Однако надо убедиться, что копия каждой архивированной группы журнала по-прежнему существует на офлайновом носителе (например на ленте).

15. *CHECKPOINT* заставляет Oracle записать все изменения, сделанные подтвержденными транзакциями, в файлы данных на диске. Можно выполнять только на открытой БД. Управление не возвращается до тех пор, пока не будет все записано.

ARCHIVE LOG -> STOP | ALL | START | LOGFILE <fn> TO <fn>

Если *TO* не указано, используется параметр *LOG_ARCHIVE_DEST*.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Как определить, какие операции восстановления возможны для того или иного случая?
2. Как определить, какие операции восстановления наиболее подходят для данного случая?
3. Как восстановить файл данных без его копии?
4. Как работать с файлами архивного журнала?

Глава 10. SQL*LOADER

1. Одной из многих проблем, с которыми часто сталкиваются администраторы базы данных, является перемещение данных из внешних источников в базу данных Oracle. Сложность этой задачи возрастает с появлением хранилищ данных; приходится перемещать уже не мегабайты данных, а гигабайты, а в некоторых случаях даже терабайты.

2. Для утилиты SQL*Loader необходимы входные данные двух типов: внешние, которые могут находиться на диске или ленте, и управляющая информация (содержащаяся в управляющем файле), которая описывает характеристики входных данных, а также таблицы и столбцы, подлежащие загрузке. Выходные данные, часть которых является необязательной, включают таблицы Oracle, журналы, файлы некорректных записей и файлы отвергнутых записей.

ФАЙЛЫ SQL*LOADER

Входные данные

3. Утилита SQL*Loader может обрабатывать файлы данных практически любого типа и поддерживает собственные типы данных почти любой платформы. Данные обычно считываются из одного или нескольких файлов данных, однако они могут быть также внесены и непосредственно в управляющий файл после управляющей информации. Файл данных может находиться как в фиксированном, так и в переменном формате.

4. В фиксированном формате данные находятся в записях фиксированной длины, имеющих одинаковый формат. Поля файлов фиксированного формата определяются начальной и конечной позициями внутри записи (т.е. смещениями), и эти поля имеют один и тот же тип данных и длину в пределах всего файла. Двоичные данные всегда должны находиться в файле фиксированного формата, поскольку SQL*Loader не может обрабатывать их в переменном формате.

5. При использовании файлов с завершающими символами необходимо обязательно заключать любые поля, содержащие в своих данных завершающий символ, в ограничительные символы. Например, при использовании файла с разделением запятыми можно использовать двойные кавычки для ограничения любого поля, содержащего запятую. В общем случае всегда лучше, если данные хранятся в кавычках.

6. Для повышения производительности используйте записи фиксированной длины. Проведенные испытания показывают, что время обработки утилитой SQL*Loader файла с переменным форматом приблизительно на 50 % больше времени обработки файла фиксированного формата. При этом для обоих форматов использовался тот же объем данных и разделение запятыми для переменных данных.

7. SQL*Loader поддерживает работу с NLS (National Language Support – поддержка национального языка) и поэтому имеет возможность интерпретировать и преобразовывать данные со схемами кодировки символов других компьютерных платформ и стран.

Управляющий файл

8. Управляющий файл следует использовать для указания параметров файла данных, а также формата данных в файлах. Управляющий

файл – это файл произвольного формата, который также содержит дополнительные управляющие данные, указывающие SQL*Loader, как обрабатывать эти данные (см. ниже).

Журнал

9. После выполнения утилита SQL*Loader создает журнал, содержащий подробную информацию о загрузке, включая следующие сведения:

- Имена файлов входных данных, управляющего файла, файлов некорректных записей и файлов отвергнутых записей.
- Входные данные и связанные с ними определения таблиц.
- Ошибки SQL*Loader.
- Результаты работы SQL*Loader.
- Итоговую статистику.

Файлы отвергнутых и некорректных записей

10. Утилита SQL*Loader обеспечивает возможность на основе определений управляющего файла форматировать входные данные и включать или исключать входную запись на основе критериев выбора записи. Если SQL*Loader отклоняет запись из-за ошибки формата или по какой-либо причине ядро Oracle не может вставить запись в таблицы базы данных, входная запись записывается в файл *BAD* в том же формате, какой был у исходного файла входных данных. Если утилита SQL*Loader отклоняет запись из-за исключения на основе критериев выбора записи, входная запись записывается в файл *DISCARD* (при условии, что он был указан в управляющем файле) также в формате исходного файла входных данных. Поскольку файлы *BAD* и *DISCARD* записаны в том же формате, какой был у исходного файла входных данных, они могут быть в случае необходимости отредактированы и использованы как файлы входных данных для другого сеанса SQL*Loader.

ПУТИ SQL*LOADER

11. Утилита SQL*Loader предусматривает два пути загрузки данных: стандартный и непосредственный.

Стандартный путь

12. Стандартный путь, по сути, вырабатывает команду SQL *INSERT* с интерфейсом обработки массива для загрузки данных в таблицы. В связи с использованием этого интерфейса утилита SQL*Loader конкурирует со всеми другими процессами Oracle за ресурсы буфера кеша; Oracle ищет и старается заполнить частично занятые блоки при каждой вставке (в общем такое поведение практически аналогично вставке данных массивом из приложения). Обычно этот метод применим при загрузке небольшого объема данных, однако при больших объемах использование этого метода требует слишком много времени и ресурсов.

13. Загрузка данных с использованием стандартного пути может быть обусловлена следующими причинами:

- Одновременно с утилитой SQL*Loader другим пользователям и процессам Oracle необходимо иметь доступ к таблице.
- По мере вставки строк в таблицу обновляются индексы.
- По мере загрузки данных в таблицу проверяются все ограничения ссылочной целостности.
- Триггеры вставки базы данных активизируются по мере вставки строк в таблицу.
- К входным данным по мере их загрузки можно применять функции SQL.
- Данные загружаются в кластеризованную таблицу.

Непосредственный путь

14. В отличие от стандартного, непосредственный путь загрузки оптимизирован для больших объемов данных. Он создает форматируемые блоки базы данных и записывает их непосредственно в базу данных.

15. Непосредственный путь дает несколько значительных преимуществ:

- Можно загружать и индексировать большой объем данных в пустые или непустые таблицы за относительно короткий промежуток времени.
- При загрузке данных в пустые таблицы можно использовать предварительно отсортированные данные и устранять этапы сортировки и слияния в построении индекса, тем самым значительно повышая производительность.

- Можно загружать данные в параллельном режиме, что позволяет нескольким сеансам SQL*Loader одновременно выполнять загрузки с использованием непосредственного пути в одну и ту же таблицу.
- Можно указать, чтобы загрузка с использованием непосредственного пути выполнялась в режиме *UNRECOVERABLE*, который обходит ведение журналов обновлений Oracle и значительно повышает производительность.

Хотя непосредственный путь значительно повышает производительность, он имеет некоторые ограничения:

- Таблицы и индексы, в которые загружаются данные, подвергаются исключительной блокировке в начале загрузки и не освобождаются до конца загрузки; какие-либо активные транзакции не могут получать доступ к этим таблицам и индексам.
- Индексы переводятся в состояние непосредственной загрузки в начале загрузки и должны быть перестроены либо автоматически, либо вручную после завершения загрузки. Если сеанс SQL*Loader не завершается успешно, индексы остаются в состоянии непосредственной загрузки и должны быть перестроены вручную. Никакие ограничения *PRIMARY KEY* или *UNIQUE* не проверяются до завершения загрузки и переформирования индекса; могут появиться дубликаты ключей, и возникнет необходимость их исправить с использованием таблицы исключений перед формированием индекса.
- Во время вставки проверяется только одно ограничение – *NOT NULL*. Все другие ограничения целостности и ссылочные ограничения снова разрешаются и предписываются после завершения загрузки. При наличии каких-либо нарушений они помещаются в таблицу исключений, указанную при создании ограничения. Таблица исключений должна быть создана перед сеансом загрузки.
- Триггеры вставки базы данных не активизируются. Любые функциональные средства приложений, которые на них полагаются, должны быть реализованы каким-либо иным методом.
- Нельзя применять функции SQL к входным данным по мере их загрузки.
- Данные не могут быть загружены в кластеризованную таблицу.
- Данные могут быть загружены с использованием SQL*Net,

только если обе компьютерные системы принадлежат к одному и тому же семейству и используют одинаковый набор символов. Нельзя использовать SQL*Net для загрузок непосредственным путем, поскольку этот путь должен использоваться только для больших объемов данных. Непроизводительные затраты сети сводят на нет любой выигрыш в производительности, связанный с применением непосредственного пути.

- Определения столбца *DEFAULT* не применимы с использованием непосредственного пути.
- Синонимы, которые существуют для загружаемых таблиц, должны указывать непосредственно на таблицу; они не могут указывать на другой синоним или представление.

Загрузка одиночных разделов с использованием непосредственного пути связана со следующими ограничениями:

- На загружаемой таблице не могут быть определены какие-либо глобальные индексы.
- На загружаемом объекте должны быть отменены ссылочные ограничения и ограничения проверки.
- На загружаемом объекте должны быть отменены триггеры.

СИНТАКСИС УПРАВЛЯЮЩЕГО ФАЙЛА

16. Управляющий файл имеет произвольный формат, однако он подчиняется некоторым правилам:

- Пробелы, символы возврата каретки, символы табуляции и т.д. разрешены в любой позиции файла.
- Как и в Oracle, содержимое файла является нечувствительным к регистру, за исключением строк, взятых в кавычки (одинарные или двойные).
- Комментарии можно включать в любом месте, но не внутри данных (при условии, что данные находятся в управляющем файле), поставив перед ними два дефиса; SQL*Loader игнорирует все находящееся за двойным дефисом и до конца строки.
- Можно использовать зарезервированные слова для имен таблиц или столбцов, однако эти слова должны быть заключены в двойные кавычки.

17. Управляющий файл логически подразделяется на семь разделов:

- Фраза *OPTIONS*.
- Фраза *UNRECOVERABLE/RECOVERABLE*.
- Фраза *LOAD DATA*.
- Фраза *INFILE*.
- Метод загрузки таблицы.
- Фраза *CONCATENATION*.
- Фраза *INTO TABLE*.

18. Кроме управляющего файла, SQL*Loader требует еще и собственно файла с загружаемыми данными. Этот файл определяется с помощью фразы *INFILE*. С точки зрения SQL*Loader, данные во входном файле хранятся в виде записей (в фиксированном или в произвольном формате).

Фиксированный формат

19. В случае, когда все записи в файле данных имеют одинаковую длину, этот формат файла данных называется фиксированным. Хотя этот формат гораздо менее гибок, чем произвольный, в результате его применения можно достичь гораздо более высокой скорости загрузки. К тому же фиксированный формат проще описать, например:

```
INFILE <datafile_name> "fix n"
```

означает, что SQL*Loader должен интерпретировать файл данных как файл с фиксированной длиной поля, а именно длиной *n* байт.

20. Ниже приведен пример управляющего файла и соответствующего файла данных. Файл данных состоит из пяти записей. Первая запись – [001, od,], ее длина ровно 11 байт. Вторая запись – [0002,fghi,] за ней следует символ перевода строки, который и является одиннадцатым байтом, и т.д.

Управляющий файл:

```
load data
infile 'example.dat'  "fix 11"
into table example
fields terminated by ',' optionally enclosed by '"'
(col1 char(5),
 col2 char(7))

example.dat:
001,   od, 0002,fghi, 00003,lmn,1, "pqrs",0005,uvwx,
```

Произвольный формат

21. Когда указывается, что файл данных является файлом в произвольном формате, SQL*Loader ожидает, что в начале каждой строки в файле данных стоит длина записи, в ней содержащаяся. Этот формат предоставляет некоторую дополнительную гибкость. Можно указать, что файл находится в произвольном формате, так

```
INFILE "datafile_name" "var n"
```

где *n* указывает число байт в поле, хранящем длину поля. Необходимо учитывать, что если это число не указано, значением по умолчанию является 5.

22. Приведенный ниже пример управляющего файла иллюстрирует сказанное выше.

```
load data
infile 'example.dat'  "var 3"
into table example
fields terminated by ',' optionally enclosed by ""
(col1 char(5),
 col2 char(7))

example.dat:
009hello,cd,010world,im,
012my,name is,
```

Фраза *OPTIONS*

23. Фраза *OPTIONS*, позволяющая указывать некоторые из аргументов времени выполнения в управляющем файле, а не в командной строке, является необязательной. Эта фраза особенно полезна, если длина командной строки превышает максимальную длину командной строки в операционной системе. Аргументы, которые можно включить в фразу *OPTIONS*, описаны ниже. Даже при указании аргументов в фразе *OPTIONS* их отменяют определения командной строки.

SKIP = n	Число логических записей, которые следует пропустить.
LOAD = n	Число логических записей для загрузки (по умолчанию – все).
ERRORS = n	Число допустимых ошибок до завершения.

SKIP = n	Число логических записей, которые следует пропустить.
ROWS = n	Число строк в массиве связывания (стандартный путь); число строк, после которого производится запись в файл (непосредственный путь).
BINDSIZE = n	Размер массива связывания в байтах.
SILENT = {HEADERFEEDBACK ERROR | DISCARDS | ALL}	Сообщения, которые необходимо подавить.
DIRECT = {TRUE | FALSE}	Метод пути загрузки.
PARALLEL = {TRUE |FALSE}	Несколько одновременно работающих сеансов.

Фраза *UNRECOVERABLE/RECOVERABLE*

24. Опции *UNRECOVERABLE/RECOVERABLE* применяются только к загрузкам по непосредственному пути; все стандартные загрузки восстановимы по определению. Когда база данных находится в режиме archivelog, то при указании опции *RECOVERABLE* загружаемые данные записываются в журналы обновлений. Указав *UNRECOVERABLE*, можно обойти запись в журналы обновлений, что повышает производительность (примерно на 100 %), но вынуждает удалять и вновь создавать загруженные данные, если потребуется восстановление носителя.

25. Если не происходит добавления данных к существующим данным в таблице, следует указывать *UNRECOVERABLE* для повышения производительности. Если по некоторым причинам экземпляр потребует восстановления, следует восстановить его и удалить все записи в таблицах, в которые были загружены данные, затем снова запустить сеанс SQL*Loader.

Фраза *LOAD DATA*

26. Фраза *LOAD DATA* – это основной оператор в управляющем файле. В управляющем файле ей могут предшествовать только комментарии, фразы *OPTIONS* и *RECOVERABLE*. За фразой *LOAD DATA* следуют фразы и опции, которые ее дополнительно уточняют. Полное описание синтаксиса управляющего файла приведено в руководстве Oracle Server Utilities Guide.

27. Фраза *LOAD DATA* начинается с ключевого слова *LOAD*, за которым может следовать ключевое слово *DATA*. За ключевым словом *CHARACTERSET* должно следовать имя набора символов, которое необходимо, если файл входных данных подготовлен в другом наборе символов. В управляющем файле может быть только одна фраза *LOAD DATA*.

Фраза *INFILE*

28. Чтобы указать входной файл, содержащий загружаемые данные, надо применять ключевое слово *INFILE* или *INDDN* с именем файла и необязательной строкой определения обработки файла в зависимости от операционной системы. Например:

```
INFILE myfile.dat
INFILE  'c:\loader\\input\\march\\sales.dat'
INFILE  '/clinical/a0004/data/clin0056.dat'
"recsize 80 buffers 10"
BADFILE '/clinical/a0004/logs/clin0056.bad1
DISCARDFILE '/clinical/a004/logs/clin0056.dsc'
DISCARDMAX 50 INFILE  'clin_a4:[data]clin0056.dat'
DISCARDFILE 'clin_a4:[log]clin0056.dsc'
DISCARDMAX 50
```

29. В одиночные кавычки необходимо заключать любое определение файла, которое содержит знаки препинания.

Если в операционной системе для указания символа «escape» используется обратный слэш, то следует использовать два обратных слэша.

Определения имен файлов и строки указания способа обработки файла обычно зависят от платформы и при переходе на другую платформу могут потребовать корректировки.

30. После оператора *INFILE* следует необязательное определение файла некорректных записей, состоящее из ключевого слова *BADFILE* или *BDDN*, за которым следует имя файла. Если имя файла некорректных записей не указано, по умолчанию берется имя файла данных с расширением «.BAD». Файл некорректных записей создается, только если записи были отклонены из-за ошибок форматирования или ядро Oracle возвратило ошибку при попытке вставить записи в базу данных.

31. После оператора *BADFILE* следует необязательное определение файла отвергнутых записей, состоящее из ключевого слова *DISCARDFILE* или *DISCARDDN*, за которым следует имя файла. Затем стоит ключевое

слово *DISCARDS* или *DISCARDMAX* с указанием целого числа. Утилита SQL*Loader может создавать файл отвергнутых записей для записей, которые не соответствуют критериям загрузки, указанным в фразах *WHEN* управляющего файла. Если файл отвергнутых записей не назван и не указаны ключевое слово *DISCARDS* и *DISCARDMAX*, файл отвергнутых записей не создается даже при наличии отвергнутых записей. Однако, если ключевые слова *DISCARDS* или *DISCARDMAX* указано в командной строке или в управляющем файле, а имя файла отвергнутых записей не указано, создается файл по умолчанию с именем файла данных и расширением «.DSC».

32. Фраза *DISCARDS* или *DISCARDMAX* ограничивает возможное число отброшенных записей для каждого файла данных. Если указано ограничение, обработка соответствующего файла данных прекращается при достижении этого предела.

И для файлов некорректных записей, и для файлов отвергнутых записей действуют следующие правила:

- Если этот файл создается, он перезаписывает существующий файл с тем же именем.
- Если этот файл не создается, а файл с тем же именем уже существует, он остается нетронутым.

Методы загрузки таблицы

33. Ключевое слово метода загрузки таблицы указывает применяемый по умолчанию глобальный метод загрузки таблиц. Можно использовать один из четырех методов:

- *INSERT*
- *APPEND*
- *REPLACE*
- *TRUNCATE*

34. *INSERT* – это метод по умолчанию, который требует, чтобы таблица перед загрузкой файла данных была пуста. Утилита SQL*Loader завершает работу с ошибкой, если таблица не пуста.

35. *APPEND* добавляет новые строки в таблицу, даже если таблица пуста.

36. *REPLACE* использует команду SQL *DELETE* для удаления всех строк из таблицы, выполняет фиксацию, а затем загружает новые данные в таблицу.

37. *TRUNCATE* использует команду SQL *TRUNCATE* для уничтожения строк таблицы, выполняет фиксацию, а затем загружает новые данные в таблицу. Все ссылочные ограничения должны быть отменены до начала сеанса SQL*Loader, в противном случае SQL*Loader завершает работу с ошибкой.

38. Если при использовании метода *REPLACE* для этой таблицы указано *DELETE CASCADE*, то выполняются также каскадные удаления. По мере удаления строк активизируются также все триггеры удаления, определенные для этой таблицы.

39. При использовании ключевого слова *REPLACE* или *TRUNCATE* перед началом загрузки данных в таблицу из этой таблицы удаляются все записи.

При использовании опции параллельной загрузки можно только добавлять строки по *APPEND*. Нельзя использовать методы *REPLACE*, *TRUNCATE* и *INSERT*. Если необходимо усечь таблицу перед параллельной загрузкой, необходимо сделать это вручную.

40. Указав глобальный метод загрузки таблицы, можно дополнительно указать метод для каждой таблицы в фразе *INTO TABLE*.

Фраза *CONCATENATION*

41. Можно создать одну логическую запись из нескольких физических записей с использованием ключевого слова *CONCATENATE* или *CONTINUEIF*.

42. Если число объединяемых записей остается постоянным на протяжении всего файла данных, можно использовать ключевое слово *CONCATENATE*, а за ним целое число, обозначающее число записей, подлежащих объединению.

43. Если число физических записей меняется, необходимо использовать ключевое слово *CONTINUEIF*, сопровождаемое условием, которое проверяется для каждой записи по мере ее обработки. При использовании ключевых слов *THIS*, *NEXT* и *LAST* условие проверяется в текущей физической записи. Если условие true, SQL*Loader считывает следующую физическую запись, соединяет ее с текущей физической записью и продолжает действовать таким образом, пока условие не станет false. Если условие примет значение false в текущей физической записи, она станет последней физической записью текущей логической записи.

Фраза *INTO TABLE*

44. Фраза *INTO TABLE* – это раздел оператора *LOAD DATA*, который содержит основную часть синтаксиса управляющего файла. Фраза *INTO TABLE* состоит из следующих пунктов:

- Имя таблицы, в которую должны быть загружены данные.
- Метод загрузки конкретной таблицы.
- Фраза *OPTIONS* конкретной таблицы.
- Фраза *WHEN*.
- Фраза *FIELDS*.
- Фраза *TRAILING NULLCOLS*.
- Опции индекса.
- Условия поля.
- Взаимосвязь между полями файла данных и столбцами базы данных.

45. Фраза *INTO TABLE* начинается с ключевых слов *INTO TABLE*, за которыми следует имя таблицы, подлежащей загрузке (эта таблица должна быть создана заранее).

Фраза *OPTIONS* конкретной таблицы

46. В фразу *INTO TABLE* можно включить фразу *OPTIONS* конкретной таблицы. Фраза *OPTIONS* действительна только для параллельных загрузок, и она отменяет определение *FILE* (единственная опция, которую можно указать на уровне таблицы) в глобальной фразе *OPTIONS* в начале управляющего файла.

Фраза *WHEN*

47. С помощью фразы *WHEN*, за которой следуют условия поля, можно указать критерии выбора записи. Фраза *WHEN* может содержать несколько сравнений; они должны быть отделены словом *AND*. Утилита SQL*Loader определяет значения полей в логической записи, а затем проверяет фразу *WHEN*. Строка вставляется, только если условие фразы *WHEN* принимает значение true. Примеры использования фразы *WHEN* показаны ниже.

```
INTO TABLE scott.emp
WHEN (12:13) ! = '50'
```

Загружает записи, которые не содержат «50» в позициях с 12-й по 13-ую.

```
INTO TABLE project
WHEN (date = 'Ol-JAN-94') AND (deptno = '20')   4
```

Использование ссылки на столбец.
Загружает записи, отмеченные датой «1 января 1994 года», для отдела «20».

Фраза *FIELDS*

48. Поля файла входных данных фиксированного формата обычно определяются явной позиционной нотацией с указанием позиции байта (startend), тогда как поля файла входных данных переменного формата обычно определяются по своему положению относительно друг друга и отделяются завершающими символами поля. Для файла во фразе *FIELDS* можно определить заданный по умолчанию завершающий символ поля. Можно отменить значение по умолчанию на уровне поля, указав завершающий символ поля после имени столбца. Например:

```
INTO TABLE emp
WHEN empno > 1000 FIELDS TERMINATED
BY WHITESPACE TRAILING NULLCOLS (
emp position(1) integer external terminated by '*,
ename char terminated by whitespace, deptno integer external terminated by ' ' )
```

Фраза *TRAILING NULLCOLS*

49. Когда в управляющем файле указано больше полей, чем в физической записи, необходимо сообщить утилите SQL*Loader, чтобы она либо обработала отсутствующие поля как пустые столбцы, либо выработала ошибку. При использовании относительного позиционирования запись может закончиться до того, как будут найдены все поля. Фраза *TRAILING NULLCOLS* сообщает SQL*Loader, чтобы она рассматривала любые столбцы с относительным позиционированием, отсутствующие в записи, как пустые столбцы.

Опции индекса

50. При загрузке данных с использованием непосредственного пути с предварительной сортировкой файла данных по индексированным столбцам

можно указать фразу *SORTED INDEXES*, за которой следуют имена индексов в круглых скобках. Индексы, перечисленные в фразе *SORTED INDEXES*, должны быть созданы до начала непосредственной загрузки, в противном случае утилита SQL*Loader вернет ошибку. Если индекс указан в фразе *SORTED INDEXES* и данные для него не были правильно отсортированы, индекс останется в состоянии непосредственной загрузки в конце загрузки. Нужно будет удалить и вновь создать индекс, прежде чем его использовать.

Условия поля

51. Условие поля представляет собой логическое выражение, которое принимает значение true или false. Условия полей могут использоваться с ключевыми словами *NULLIF* и *DEFAULTIF*, а также в фразе *WHEN*. *NULLIF* устанавливает столбец в пустое значение, если выражение принимает значение true, тогда как *DEFAULTIF* заполняет столбец нулями или пробелами. Ключевое слово *BLANKS* позволяет легко сравнивать поле любой длины и определять, не заполнено ли оно только пробелами. Например:

```
dept no POSITION (1:2) integer external
NULLIF (dept = BLANKS) comm POSITION (50:57) integer external
DEFAULTIF (hiredate > 'Ol-jan-94')
```

Определения поля

52. Определения типа данных в определении поля управляющего файла сообщают SQL*Loader, как интерпретировать данные в файле данных. Определение поля в ***словаре данных*** задает тип данных для каждого поля базы данных. Связью между данными в файле данных и в поле базы данных служит имя поля, указанное в управляющем файле.

53. Может быть загружено любое число полей таблицы при условии, что незагруженные поля не были созданы с ограничениями *NOT NULL*. Полям, определенным для таблицы, но не указанным в управляющем файле при вставке записи, присваиваются пустые значения.

54. Определением поля является его имя, за которым следует определение значения, помещаемое в поле. Перечень полей должен быть заключен в круглые скобки и разделен запятыми. Каждое имя поля должно соответствовать полю в таблице, которая была названа в фразе *INTO TABLE*.

Установка значений полей

55. Можно установить значение поля одним из двух способов. Значение можно считать из файла данных, указав либо явную позиционную нотацию (start:end) для файлов фиксированного формата, либо относительную позиционную нотацию для файлов переменного формата. Второй способ состоит в выработке этого значения с использованием функции SQL*Loader *CONSTANT*, *RECNUM*, *SYSDATE* или *SEQUENCE*. Можно использовать эти функции SQL*Loader и для стандартного пути, и для непосредственного пути.

56. Функция *CONSTANT*, за которой следует значение, вырабатывает постоянное значение для каждой записи, вставляемой в таблицу. SQL*Loader интерпретирует значение как символьное, но в случае необходимости преобразовывает его в тип данных поля базы данных. Значение может быть заключено в кавычки. Числовые значения, превышающие 232-1 (4294967295), должны быть заключены в кавычки.

57. Функция *RECNUM*, для которой не нужно указывать значение, устанавливает значение поля на номер логической записи, из которой была загружена строка. Записи отсчитываются последовательно с начала файла данных, начиная с записи 1. Значение наращивается с каждой логической записью, даже если эта запись была отвергнута, пропущена или отклонена. При использовании параллельной опции для загрузки нескольких файлов данных одновременно каждый сеанс SQL*Loader вырабатывает дубликаты значений, поскольку каждый сеанс начинается с 1.

58. Функция *SYSDATE* получает текущую системную дату для каждого массива записей при использовании стандартного пути и для каждого блока записей при использовании непосредственного пути. Формат даты совпадает с форматом функции *SYSDATE* SQL. Столбец базы данных должен иметь тип данных *VARCHAR*, *CHAR* или *DATE*.

59. Функция *SEQUENCE* утилиты SQL*Loader (которая не совпадает с функцией *SEQUENCE* объекта базы данных) увеличивается на указанное значение приращения, определенное для каждой загружаемой или отклоняемой логической записи. Эта функция не наращивается для отброшенных или пропущенных записей. Начальным значением для функции *SEQUENCE* может быть:

- Указанное целое число.
- *COUNT*, которое представляет собой число записей в таблице плюс приращение.
- ***MAX***, которое представляет собой текущее максимальное значение поля плюс приращение.

При использовании параллельной опции функция *SEQUENCE* является единственной опцией, применимой в SQL*Loader для создания уникальных чисел; для всех других опций требуются функции SQL, которые нельзя использовать при загрузке по непосредственному пути.

ИСПОЛЬЗОВАНИЕ ФУНКЦИЙ SQL

60. Можно применять функции SQL к данным поля, используя строку SQL. Эта строка может содержать любую комбинацию выражений SQL, которые возвращают единственное значение. Она должна находиться после всех прочих определений данного поля и должна быть заключена в двойные кавычки. Если строка содержит синтаксическую ошибку, загрузка завершается с ошибкой. Если синтаксис в строке правилен, но вызывает ошибку базы данных, строка отклоняется и обработка продолжается. Можно ссылаться на имена полей в строке SQL, ставя перед ними двоеточие. Ниже приведены некоторые примеры использования функций SQL:

```
my_field position(1:18) integer external
"TRANSLATE (RTRIM(my_field),'N/A','0')" my_string CHAR terminated by
","
"SUBSTR (:my_string,l,10)" my_number position(1:9)
DECIMAL EXTERNAL(8)
"my_field/1000"
```

МНОГОТАБЛИЧНЫЕ ОПЕРАТОРЫ

61. SQL*Loader позволяет указывать несколько операторов *INTO TABLE,* использование которых дает возможность выполнять следующие задачи:

- Извлекать несколько логических записей из одной физической записи.
- Различать форматы записи.
- Загружать данные в различные таблицы.

62. Ниже приведен пример, позволяющий использовать значения различных полей для заполнения поля в зависимости от условий, содержащихся в данных.

```
INTO TABLE proj
WHEN projno ! = '   '
(empo position(1:4) integer external, projno        position(29:31) integer external)
INTO TABLE proj
WHEN PROJNO ! = '   '
(empno  position(1:4) integer external projno position(33:35) integer external)
```

63. Пример загрузки одних и тех же данных в физической записи в несколько таблиц:

```
INTO TABLE dept
WHEN deptno < 100
(deptno POSITION(1:4) INTEGER EXTERNAL (dname POSITION(6:19)
CHAR) INTO TABLE emp
WHEN empno > 1000
(empno POSITION(1:4) INTEGER EXTERNAL, ename POSITION(6:15)
CHAR deptno POSITION(17:18) INTEGER EXTERNAL)
```

Пример заполнения нескольких таблиц на основе условий *WHEN*:

```
INTO TABLE current_studies
WHEN (status = current) AND (country_code = 'US')
INTO TABLE Study_345
WHEN (11:13) = '345'
INTO TABLE Study_456
WHEN (11:13) = '456'
```

ОПЦИИ КОМАНДНОЙ СТРОКИ И ФАЙЛЫ ПАРАМЕТРОВ

64. SQLLOAD parfile=weekly.par bindsize silent direct userid scott/tiger control = weekly.ctl log = weekly.log errors = 25 direct = true.

Опции командной строки, указанные в управляющем файле или файле parfile, могут быть переопределены в командной строке.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Для чего нужен SQL*Loader?
2. Как загружать данные в фиксированном формате?
3. Как загружать данные в произвольном формате?
4. Какие основные ключевые слова позволяют управлять процессом загрузки?
5. Какие средства можно использовать для автоматизации процесса загрузки данных?

Глава 11. УПРАВЛЕНИЕ ТРАНЗАКЦИЯМИ

1. Транзакция – элементарная единица работы, состоящая из одного или нескольких операторов SQL.

ФИКСАЦИЯ И ОТКАТ

2. Изменения в базе данных не сохраняются, пока пользователь явно не укажет, что результаты вставки, модификации и удаления должны быть зафиксированы окончательно. Вплоть до этого момента изменения находятся в отложенном состоянии, и какие-либо сбои, подобные аварийному отказу компьютера, аннулируют изменения.

3. Транзакция начинается, когда пользователь подключается к базе данных, и завершается, когда вводится оператор *COMMIT* или *ROLLBACK*. После выполнения операторов *COMMIT* или *ROLLBACK* автоматически начинается следующая транзакция. Все результаты транзакции или полностью сохраняются (фиксируются), или полностью отменяются (выполняется откат).

4. Фиксация транзакции делает изменения окончательными, занося их в базу данных, и, после того как транзакция фиксируется, изменения не могут быть отменены. Откат отменяет все вставки, модификации и удаления, сделанные в транзакции; после отката транзакции ее изменения не могут быть зафиксированы. Процесс фиксации транзакции подразумевает запись изменений, занесенных в кеш обновлений SGA, в оперативные журналы обновлений на диске. Если этот дисковый ввод/вывод успешен, приложение получает сообщение об успешной фиксации транзакции. (Текст сообщения изменяется в зависимости от инструментального средства.) Фоновый процесс *DBWR* может записывать блоки актуальных данных Oracle в буферный кеш SGA базы данных позже. В случае сбоя системы Oracle может автоматически повторить изменения из журналов обновлений, даже если блоки данных Oracle не были перед сбоем записаны в файлы базы данных.

5. Оператор *DDL*, такой как *CREATE TABLE*, автоматически выдает команду *COMMIT*, которая будет выполнена, даже если сам оператор *DDL* выполнится неудачно.

6. Oracle также реализует идею отката на уровне оператора. Если произойдет единственный сбой при выполнении оператора, весь оператор завершится неудачей. Другими словами, оператор *INSERT* для 1000 строк

вставит или все 1000 строк или не вставит вообще ни одной; весь оператор или выполнится, или нет. Если оператор терпит неудачу в пределах транзакции, остальные операторы транзакции будут находиться в отложенном состоянии и должны либо фиксироваться, либо откатываться.

7. Если пользовательский процесс завершается ненормально (например, процесс уничтожен), фоновый процесс *PMON* автоматически выполняет откат изменений. Какие-либо изменения, которые процесс зафиксировал до момента сбоя, остаются фиксированными, и только для текущей транзакции выполняется откат изменений.

8. Все блокировки, захваченные транзакцией, автоматически освобождаются, когда транзакция фиксируется или когда выполняется откат, или когда фоновый процесс PMON отменяет транзакцию. Кроме того, другие ресурсы системы (такие как сегменты отката) освобождаются для использования другими транзакциями.

9. Точки сохранения позволяют устанавливать маркеры внутри транзакции таким образом, чтобы имелась возможность отмены только части работы, проделанной в транзакции. Целесообразно использовать точки сохранения в длинных и сложных транзакциях, чтобы обеспечить возможность отмены изменения для определенных операторов. Однако это обусловливает дополнительные затраты ресурсов системы: оператор выполняет работу, а изменения затем отменяются; обычно усовершенствования в логике обработки могут оказаться более оптимальным решением. Oracle освобождает блокировки, захваченные отмененными операторами.

Явное назначение транзакции сегменту отката

10. Каждая транзакция требует ассоциированного сегмента отката. Обычно сегмент отката назначается транзакции автоматически. Вы можете, однако, явно назначить транзакцию подходящему сегменту отката, используя команду *SET TRANSACTION* с параметром *USE ROLLBACK SEGMENT*. Транзакции явно назначаются сегментам отката со следующими целями:

- чтобы предполагаемое количество информации отката, генерируемой транзакцией, могло уместиться в текущих экстентах назначаемого сегмента отката;

- чтобы не пришлось динамически распределять (а потом освобождать) дополнительные экстенты для сегмента отката, что снизило бы общую производительность системы.

Подобные ситуации могут возникнуть, когда в одной транзакции идет работа с очень большим массивом данных.

11. Для явного назначения транзакции сегменту отката необходимо, чтобы этот сегмент отката был доступен и чтобы предложение *SET TRANSACTION USE ROLLBACK SEGMENT* было первым в транзакции. Если указанный сегмент отката не доступен или предложение *SET TRANSACTION USE ROLLBACK SEGMENT* – не первое в транзакции, то возвращается ошибка.

Подтверждение транзакции

12. Для подтверждения транзакции необходимо использовать команду *COMMIT*. Следующие предложения эквивалентны и подтверждают текущую транзакцию:

```
COMMIT WORK;
COMMIT;
```

13. Команда *COMMIT* позволяет включать опцию *COMMENT* для возможности задать комментарий (длиной до 50 символов), содержащий информацию о подтверждаемой транзакции. В частности, эта возможность может быть полезна для того, чтобы указывать источник транзакции при подтверждении распределенных транзакций, например:

```
COMMIT COMMENT 'Dallas/Accts_pay/Trans_type 10B';
```

Откат транзакции

14. Чтобы отменить всю транзакцию или ее часть, необходимо использовать команду *ROLLBACK*. Например, каждое из следующих предложений откатывает всю текущую транзакцию:

```
ROLLBACK WORK;
ROLLBACK;
```

15. Для отката к точке сохранения, определенной в текущей транзакции, должна использоваться опция *TO* команды *ROLLBACK*. Например,

каждое из следующих предложений откатывает текущую транзакцию к точке сохранения с именем *POINT1*:

```
ROLLBACK TO SAVEPOINT point1;
ROLLBACK TO point1;
```

ОПРЕДЕЛЕНИЕ ТОЧКИ СОХРАНЕНИЯ ТРАНЗАКЦИИ

16. Для определения точки сохранения транзакции необходимо использовать команду *SAVEPOINT*. Следующее предложение создает точку сохранения с именем ADD_EMP1 в текущей транзакции:

```
SAVEPOINT add_emp1;
```

17. Если создается вторая точка сохранения с тем же именем, что у более ранней точки сохранения, то более ранняя точка сохранения становится неопределенной. После того как точка сохранения создана, можно выполнять откат к этой точке.

18. Не существует ограничения на количество точек сохранения на сессию.

Пример использования *COMMIT, SAVEPOINT и ROLLBACK*

19. Следующая серия предложений SQL иллюстрирует применение предложений *COMMIT*, *SAVEPOINT* и *ROLLBACK* внутри транзакции:

Предложение SQL	*Результаты*
SAVEPOINT a;	Первая точка сохранения в транзакции.
DELETE ... ;	Первое предложение *DML* в транзакции.
SAVEPOINT b;	Вторая точка сохранения в транзакции.
INSERT INTO ... ;	Второе предложение *DML* в транзакции.
SAVEPOINT c;	Третья точка сохранения в транзакции.
UPDATE ... ;	Третье предложение *DML* в транзакции.
ROLLBACK TO c;	Предложение *UPDATE* откатывается, точка сохранения *c* остается определенной.
ROLLBACK TO b;	Предложение *INSERT* откатывается, точка сохранения *c* теряется, точка сохранения *b* остается определенной.

ROLLBACK TO c;	Ошибка – точка сохранения C больше не определена.
INSERT INTO ... ;	Новое предложение *DML* в транзакции.
COMMIT;	Подтверждает все действия, выполненные первым предложением *DML* в транзакции (*DELETE*) и последним предложением *DML* (вторым предложением *INSERT*). Все прочие предложения (второе и третье предложения *DML*) в этой транзакции были подвергнуты откату перед этим *COMMIT*.

20. Никаких привилегий для управления своими транзакциями пользователю не требуется.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое транзакция, фиксация, откат?
2. Какова связь транзакций с сегментами отката?
3. Как подтвердить, откатить транзакцию?
4. Что такое точка сохранения транзакции?

ЗАКЛЮЧЕНИЕ

В данном учебном пособии рассмотрены далеко не все возможности современных СУБД и различные аспекты их применения. Также не рассмотрены вопросы теории реляционных СУБД. В пособии, прежде всего, внимание сконцентрировано на системных аспектах функционирования СУБД, на примере СУБД Oracle показано внутреннее устройство и взаимодействие входящих в СУБД подсистем. Объектно-ориентированные стороны Oracle также оставлены за кадром, поскольку это тема для отдельной книги.

С помощью этой книги читатель, уже знакомый с какой-либо реляционной СУБД, сможет значительно повысить свои теоретические знания по администрированию современных СУБД, а также научиться максимально эффективно использовать Oracle для хранения, обработки и защиты информации практически любого объёма.

ПРИЛОЖЕНИЯ

1. ЗАРЕЗЕРВИРОВАННЫЕ СЛОВА ORACLE

Зарезервированные слова Oracle и PL/SQL

ACCESS* DEFAULT* INTEGER OPTION* START*
ADD* DELETE* INTERSECT* OR* SUCCESSFUL
ALL* DESC* INTO* ORDER* SYNONYM
ALTER* DISTINCT* IS* PCTFREE* SYSDATE
AND* DROP* LEVEL* PRIOR* TABLE*
ANY* ELSE* LIKE* PRIVILEGES THEN*
AS* EXCLUSIVE LOCK PUBLIC* TO*
ASC* EXISTS* LONG RAW TRIGGER
AUDIT FILE MAXEXTENTS RENAME* UID
BETWEEN* FLOAT MINUS* RESOURCE* UNION*
BY* FOR* MODE REVOKE UNIQUE*
CHAR* FROM* MODIFY ROW UPDATE*
CHECK* GRANT* NOAUDIT ROWID USER
CLUSTER* GROUP* NOCOMPRESS* ROWLABEL VALIDATE
COLUMN HAVING* NOT* ROWNUM* VALUES*
COMMENT IDENTIFIED* NOWAIT ROWS VARCHAR*
COMPRESS* IMMEDIATE NULL* SELECT* VARCHAR2*
CONNECT* IN* NUMBER* SESSION VIEW*
CREATE* INCREMENT OF* SET* WHENEVER
CURRENT* INDEX* OFFLINE SHARE WHERE*
DATE* INITIAL ON* SIZE* WITH*
DECIMAL INSERT ONLINE SMALLINT

Зарезервированные слова PL/SQL

ABORT CREATE GOTO PACKAGE SUM
ACCEPT CURRENT GRANT PARTITION TABAUTH
ACCESS CURSOR GROUP PCTFREE TABLE
ADD DATABASE HAVING PRAGMA TABLES
ALL DATA_BASE IDENTIFIED PRIOR TASK
ALTER DATE IF PRIVATE TERMINATE
AND DBA IN PROCEDURE THEN
ANY DEBUGOFF INDEX PUBLIC TO
ARRAY DEBUGON INDEXES RAISE TRUE
AS DECLARE INDICATOR RANGE TYPE
ASC DEFAULT INSERT RECORD UNION
ASSERT DEFINITION INTERSECT RELEASE UNIQUE
ASSIGN DELAY INTO REM UPDATE
AT DELETE IS RENAME USE
AUTHORIZA DELTA LEVEL RESOURCE VALUES
TION DESC LIKE RETURN VARCHAR
AVG DIGITS LIMITED REVERSE VARCHAR2
BEGIN DISPOSE LOOP REVOKE VARIANCE
BETWEEN DISTINCT MAX ROLLBACK VIEW
BODY DO MIN ROWNUM VIEWS
BOOLEAN DROP MINUS ROWTYPE WHEN
BY ELSE MOD RUN WHERE
CASE ELSIF NEW SAVEPOINT WHILE
CHAR END NOCOMPRESS WSCHEMA WITH
CHAR_BASE ENTRY NOT SELECT WORK
CHECK EXCEPTION NULL SEPARATE XOR
CLOSE EXCEP NUMBER SET
CLUSTER TION_INIT NUMBER_BASE SIZE
CLUSTERS EXISTS OR SPACE
COLAUTH EXIT ON SQL
COLUMNS FALSE OPEN SQLCODE
COMMIT FETCH OPTION SQLERRM
COMPRESS FOR OR START

CONNECT FORM ORDER STATEMENT
CONSTANT FROM OTHERS STDDEV
COUNT FUNCTION OUT SUBTYPE
CRASH GENERIC

2. ПРИОРИТЕТ ОПЕРАТОРОВ

Оператор	*Описание*
( )	Подавляет обычные правила старшинства операций.
** NOT	Возведение в степень и логическое отрицание.
+ –	Знак, предшествующий числовому выражению.
* /	Умножение и деление.
+ –	Сложение и вычитание.
||	Сочленение текстовых выражений и (или) констант.
:=	Присвоение значения переменной пользователя.
=, <>, <, >, <=, >=, IS NULL, LIKE, IN, BETWEEN	Операторы сравнения, используемые при построении условий.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. *Бобровски С.* Oracle8: Архитектура. – М.: Лори, 1998. – 210 с.
2. *Грабер М.* SQL: Справ. руководство. – М., Лори, 1997. – 291 с.
3. Использование Oracle8: Спец. изд. / В. Пэйдж, Н. Хьюз, и др.– Киев; М.; СПб.: Вильямс, 1998. – 752 с.
4. Настройка Oracle8 / М. Кори, М. Эбби, Д. Дечичьо (мл.) и др. – М.: Лори, 1999. – 396 с.
5. *Юринский В., Бачин А., Абрамов В.* Oracle7. Практ. руководство. – М.: Софтсервис, 1997. – 420 с.

ОГЛАВЛЕНИЕ

Учебное издание

Комплексная защита объектов информатизации. Книга 3

АЛЕКСАНДРОВ Александр Александрович

А Д М И Н И С Т Р И Р О В А Н И Е Б А З Д А Н Н Ы Х

Учебное пособие

Редактор И.А. Арефьева
Корректор Е.В. Афансьева
Компьютерная верстка Д.Н. Ях

ЛР №020275. Подписано в печать 08 .12.03.
Формат 60x84/16. Бумага для множит. техники. Гарнитура Таймс.
Печать на ризографе. Усл. печ. л. 5,11. Уч.- изд. л. 5,31. Тираж 100 экз.
Заказ
Редакционно-издательский комплекс
Владимирского государственного университета.
600000, Владимир, ул. Горького, 87.