МИНИСТЕРСТВО ПО РАЗВИТИЮ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И КОММУНИКАЦИЙ РЕСПУБЛИКИ УЗБЕКИСТАН

ТАШКЕНТСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ ФЕРГАНСКИЙ ФИЛИАЛ

ФАКУЛЬТЕТ “КОМПЬЮТЕР ИНЖИНИРИНГ”
КАФЕДРА “КОМПЬЮТЕРНЫЕ СИСТЕМЫ”

КОНСПЕКТ ЛЕКЦИЙ

по предмету:

«БАЗЫ И БАНКИ ДАННЫХ»

для студентов направлений

5330200 – “Информатика и информационные технологии” (по отраслям)

Фергана – 2015 г.

Лекция№ 1. Введение. Цели и задачи. Основные понятия и определения СУБД ORACLE. Краткая история развития СУБД ORACLE. РСУБД ORACLE. Серверные системы. Модель реляционной базы данных. Архитектура ORACLE.

План

1.История СУБД ORACLE
2. Основные понятия и определения СУБД ORACLE
3. Архитектура ORACLE

История СУБД ORACLE

История Oracle началась в легендарной Силиконовой долине, штат Калифорния, США. В 1977 году молодой программист Ларри Эллисон бросил учебу в Йельском университете, чтобы начать собственный бизнес. Ларри Эллисон, в распоряжении которого тогда было всего 1200 долларов, уговорил Боба Майнера и Эда Оутса, своих бывших коллег, создать собственную компанию. До этого все трое разрабатывали по заказу ЦРУ проект под названием... Oracle. Так в 1977 году появилась Software Development Lab., вскоре переименованная сначала с Relational Software Inc., а затем - в Oracle. Молодые программисты, чьи общие вложения в бизнес составили $2 тыс., начали разработку системы управления базами данных (СУБД), построенной на принципах реляционной алгебры.

В июне 1979 года появилась первая система компании Эллисона - Oracle 2. В маркетинговом отношении Эллисон оказался талантливее Билла Гейтса. В отличие от основателя Microsoft, который первую версию Windows так и назвал - Windows 1.0, он проявил маркетинговую смекалку, поэтому свет увидела сразу вторая версия продукта. Это должно было как бы дать понять заказчикам, что система надежна и даже прошла проверку временем. Эллисона можно понять: у него не было одобрения со стороны IBM, которое было у Гейтса. Даже напротив: Эллисон создавал продукт, который мог стать конкурентом решениям IBM. Главным конкурентным преимуществом СУБД Oracle была высокая скорость обработки огромных массивов информации, которую тогда отметили все эксперты. В отличие от System R, для работы которой был необходим мощный суперкомпьютер - мейнфрейм, Oracle 2 справлялась с обработкой информации на гораздо более скромных машинах. Это и поспособствовало невероятно широкому распространению детища Эллисона в начале 80-х годов.

Краткая история Oracle:

- 1977 - Ларри Эллисон, Боб Майнер и Эд Оутс основали компанию Software Development Laboratories (SDL), предшественницу Oracle.
- 1979 - SDL сменила имя на Relational Software, Inc. (RSI) и выпустила Oracle v2. Эта версия не поддерживала транзакции, но реализовывала основную функциональность SQL. Это была первая коммерческая система управления реляционными базами данных (СУРБД) на основе языка запросов SQL. Первая версия была написана на ассемблере, работала на системе PDP-11 под управлением операционной системы RSX-11, используя 128 кб оперативной памяти.
- 1982 - RSI вновь сменила своё имя и стала называться Oracle Systems.
- 1983 - выпущена версия Oracle 3, переписанная на C и поддерживающая функции COMMIT и ROLLBACK для реализации транзакций. В этой версии поддержка платформ была расширена: помимо реализации на DEC VAX/VMS появилась реализация на Unix. Oracle v3 являлась первой СУРБД, работающей на мейнфреймах, миникомпьютерах и ПК.
- 1984 - выпущена версия Oracle 4.

- 1985 - выпущена версия Oracle 5, одна из первых СУРБД, работающих в клиент-серверных средах. Появляется поддержка распределённых запросов, Oracle Link, кластерных технологий (реализация для DEC VAX), многоверсионного согласованного чтения.
- 1986 - выпущена версия Oracle 5.1.
- 15 марта 1986 - Oracle Corporation выходит на биржу.
- 1988 - выпущена версия Oracle 6, с поддержкой блокировок на уровне строк и средств «горячего» резервирования. Появляется поддержка встроенного языка PL/SQL в средстве разработки приложений Oracle Forms v3 (в 6-й версии СУБД еще нет поддержки PL/SQL).
- 1989 - выпущена версия Oracle 6.2, с поддержкой средств оперативной обработки транзакций (OLTP). Oracle переносит свою штаб-квартиру в комплекс зданий в Рэдвуд Шорз, штат Калифорния.
- 1992 - выпущена версия Oracle 7, с поддержкой ссылочной целостности, хранимых процедур и триггеров.
- 1994 - выпущена версия Oracle 7.1, в том числе для IBM PC - до этого времени компания Oracle не рассматривала данную платформу как серверную, ограничиваясь лишь созданием для нее клиентских частей своей СУБД.
- 1996 - выпущена версия Oracle 7.3, включающая Universal Server, позволяющий управлять данными любых типов - текстами, видеоматериалами, картами, аудиозаписями или графическими изображениями.
- 1997 - выпущена версия Oracle 8 (8.0), основными особенностями которой стали более высокая надежность по сравнению с предыдущей версией, а также поддержка большего числа пользователей и больших объемов данных. Появляется поддержка средств объектно-ориентированной разработки и мультимедийных приложений; партиционирование таблиц. Oracle становится объектно-реляционной СУБД.
- 1998 - выпущена версия Oracle 8i Release 1 (8.1.5), «i» в названии обозначает «Internet», символизируя поддержку интернета. Начиная с Oracle 8.1.5 - появляется встроенная в СУБД виртуальная машина Java (JVM). На Java написаны клиентские утилиты, инсталлятор, средства администрирования.
- 1998 - выпущена версия Oracle 8i Release 2 (8.1.6), поддерживающая XML и содержащая некоторые новшества, связанные с созданием хранилищ данных.
- 2000 - выпущена версия Oracle 8i Release 3 (8.1.7), содержащая Java Virtual Machine Accelerator и Internet File System. Последний Patch Set, выпущенный для данной версии — 8.1.7.4.1. Последний Patch (для платформы Win32) - 8.1.7.4.29.
- 2001 - выпущена версия Oracle 9i Release 1 (9.0.1). В версии 9i появляются: средства обработки XML-документов; технология Oracle RAC (Real Application Clusters), как замена Oracle Parallel Server (OPS); механизм создания репликаций Oracle Streams; скроллируемый курсор для программ на C и C++; встроенная в СУБД поддержка OLAP и Data Mining; переименование столбцов и ограничений целостности; поддержка Java 1.3.1 и Unicode 3.1.
- 2004 - выпущена версия Oracle 10g Release 1 (10.1.0); «g» в названии обозначает «Grid» («сеть»), символизируя поддержку распределенных вычислений (Grid-вычислений).
- 2005 - выпущена версия Oracle 10g Release 2 (10.2.0.1).
- 2007 - выпущена версия Oracle 11 g Release **1** (11.1.0.6).

Основные понятия и определения СУБД ORACLE

База данных - это набор данных. Oracle позволяет сохранять данные и получать к ним доступ в соответствии с моделью, называемой реляционной. В связи с этим Oracle называют системой управления реляционными базами данных (РСУБД). Чаще всего под базой данных подразумевают не только физические данные, но также и комбинацию физических объектов, объектов памяти и процессов.

Экземпляром (или сервером) БД называется набор структур памяти и фоновых процессов, обращающихся к группе файлов базы данных. К одной базе данных могут обращаться несколько экземпляров (свойство Real Application Clusters). Экземпляр может

смонтировать и открыть только одну базу данных в каждый момент времени Отношения между экземплярами и базами данных показаны на рис. 1.1.

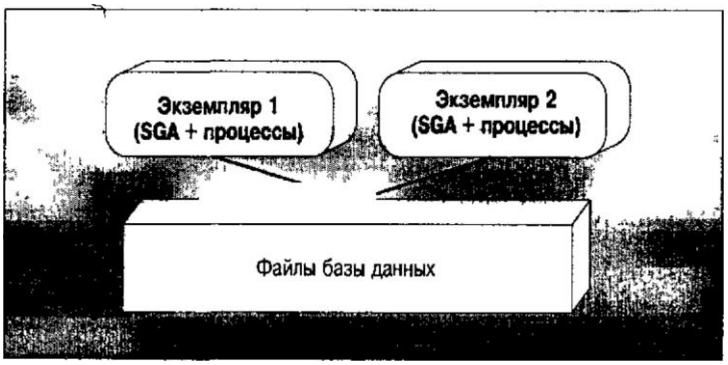


Рис 1.1 Экземпляры и файлы данных в Oracle.

Startup *(запуск)* - это процесс выполнения команды, необходимой для того, чтобы сделать базу данных ORACLE 11g доступной для приложений. После завершения процесса запуска, база данных становится в режим OPEN. Затем база данных готова к использованию.

Shutdown (остановка) - это процесс остановки базы данных ORACLE llg. Когда база данных ORACLE llg остановлена никто не может получить доступ к информации и к файлам Oracle.

Background process (фоновые процессы) - эти процессы поддерживают доступ к запущенной базе данных ORACLE llg, играют жизненно важную роль в реализации работы базы данных ORACLE llg. Различные фоновые процессы рождаются, когда база данных запущена **и** каждый из них производит небольшое количество заданий до момента остановки базы данных.

Архитектура ORACLE

Архитектура Oracle включает в себя три основных компонента (рис 1.2).

- **Файлы.** Будут рассмотрены пять видов файлов, образующих базу данных и поддерживающих экземпляр. Это файлы параметров, сообщений, данных, временных данных и журналов повторного выполнения.
- **Структуры памяти,** в частности системная глобальная область (System Global Area - SGA). Мы рассмотрим взаимодействие SGA, PGA и UGA. Будут также рассмотрены входящие в SGA Java-пул, разделяемый пул и большой пул.
- **Физические процессы или потоки.** Будут описаны три типа процессов, образующих экземпляр: серверные процессы, фоновые процессы и подчиненные процессы.

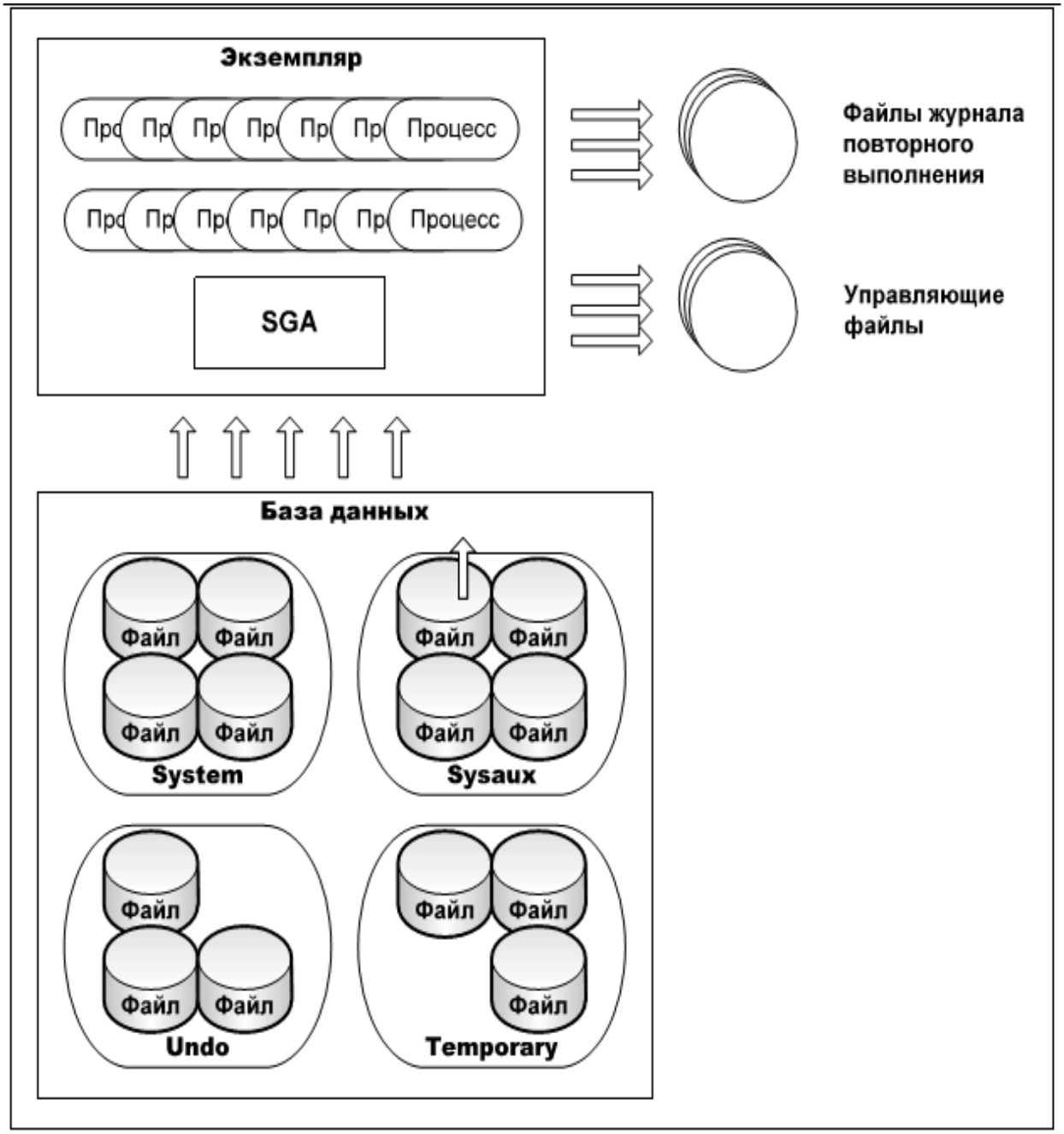


Рис. 1.2 Упрощенный вид архитектуры ORACLE.

Файлы

Базу данных образуют следующие файлы:

- **Файлы данных.** Собственно данные (в этих файлах хранятся таблицы, индексы и все остальные сегменты).
- **Файлы журнала повторного выполнения.** Журналы транзакций.
- **Управляющие файлы.** Определяют местонахождение файлов данных и содержат другую необходимую информацию о состоянии базы данных.
- **Временные файлы.** Используются при сортировке больших объемов данных и для хранения временных объектов.
- **Файлы паролей.** Используются для аутентификации пользователей, выполняющих администрирование удаленно, по сети. Мы не будем их подробно рассматривать.

Табличные пространства. Табличное пространство представляет собой логический раздел базы данных. В каждой базе данных имеется хотя бы одно табличное пространство, называемое системным (SYSTEM). Можно создавать другие табличные пространства, чтобы объединить вместе пользователей или приложения, что позволит упростить управление базой данных и повысить ее производительность. В качестве

примеров можно привести табличные пространства USERS для общего использования и UNDO для сегментов отмены (см. ниже). Табличное пространство может принадлежать только одной базе данных.

Структуры памяти

Теперь пришло время рассмотреть основные структуры памяти сервера Oracle. Их три.

• SGA, System Global Area — глобальная область системы. Это большой совместно используемый сегмент памяти, к которому обращаются все процессы Oracle.

• PGA, Process Global Area — глобальная область процесса. Это приватная область памяти процесса или потока, недоступная другим процессам/потокам.

• UGA, User Global Area — глобальная область пользователя. Это область памяти, связанная с сеансом. Глобальная область памяти может находиться в SGA либо в PGA. Если сервер работает в режиме MTS, она располагается в области SGA, если в режиме выделенного сервера, — в области PGA.

Физические процессы

В экземпляре Oracle есть три класса процессов.

• Серверные процессы. Они выполняют запросы клиентов. Мы уже затрагивали тему выделенных и разделяемых серверов. И те, и другие относятся к серверным процессам.

• Фоновые процессы. Это процессы, которые начинают выполняться при запуске экземпляра и решают различные задачи поддержки базы данных, такие как запись блоков на диск, поддержка активного журнала повторного выполнения, удаление прекративших работу процессов и т.д.

• Подчиненные процессы. Они подобны фоновым процессам, но выполняют, корме того, действия от имени фонового или серверного процесса. Мы рассмотрим все эти процессы и постараемся выяснить, какую роль они играют в экземпляре.

Вопросы к лекции

1. Что такое база данных?
2. Что является экземпляром (или сервером) БД?
3. Что включает в себя архитектура Oracle?
4. Что значит SGA, PGA, UGA?

Список используемой литературы

Кевин Луни, Марлен Терьо «Oracle 9i. Настольная книга администратора» изд. «ORACLE Press» перев. «Лори» 2004г. 750 стр.

Ian Abramson, Michael Abbey, Michael J. Corey, Michelle Malcher «Oracle Database 1 lg. A Beginner's Guide» изд. «McGrowHill» 2008г. 415 стр.

Том Кайт «Oracle для профессионалов. Книга 1. 4.Архитектура и основные особенности.» изд. «APress» перев. «ДиаСофтЮП» 2003г 672 стр.

Лекция №2. **Основные объекты ORACLE. Средства манипуляции с данными ORACLE. Структура запросов. Создание простых запросов. Создание выборки данных. Сортировка.**

План

1. Основные объекты ORACLE
2. Объекты схемы и общие объекты ORACLE
3. Средства манипулирования данными языка SQL
4. Использование SQL операторов при формировании запросов.
5. Сортировка данных.

Основные объекты ORACLE

Существуют следующие объекты в ORACLE:

- табличные пространства;
- сегменты;
- экстенты;
- блоки данных;
- каталоги (directory);
- пользователи;
- роли;
- профили;
- таблицы;
- столбцы: таблиц и представлений;
- временные таблицы;
- ограничения целостности;
- кластеры;
- последовательности;
- индексы;
- секции (разделы и подразделы)
- снимки;
- представления;
- материализованные представления;
- хранимые функции и процедуры;
- библиотеки внешних процедур;
- пакеты;
- триггеры;
- объектные таблицы;
- объектные типы;
- объектные представления;
- синонимы;
- связи баз данных;

Табличные пространства - верхний уровень абстракции. Обладает следующими свойствами:

• База данных ORACLE может быть логически сгруппирована в более мелкие области, называемые табличными пространствами.
• Табличное пространство может принадлежать только одной базе данных.
• Каждое табличное пространство состоит из одного или более файлов данных.
• Табличное пространство может содержать один или несколько сегментов данных.
• Табличные пространства могут быть переведены в активный (online) режим, в котором данные доступны пользователям, или в неактивный (автономный, offline) режим, когда

файлы закрыты и данные недоступны пользователям.
- Табличные пространства можно переводить в режим «только для чтения» (Read only) или в режим записи (read write).
- Существует табличное пространство SYSTEM, которое содержит словарь данных.

Сегмент данных - пространство, выделенное для логического объекта в табличном пространстве. Он располагается только в одном табличном пространстве, но может находится в любом файле этого табличного пространства. Сегмент состоит из одного или более экстентов.

Экстент — это последовательность физически прилегающих друг к другу блоков данных. Табличное пространство для сегментов выделяется путем добавления экстентов. При создании сегмента, ему выделяется по крайней мере один экстент. Сегмент увеличивает размер на экстент. При этом администратор базы данных вручную может добавлять экстент сегменту. Экстент может принадлежать только одному файлу данных.

Блок данных — наименьшая логическая единица, которую ORACLE выделяет в файле данных. Блок данных ORACLE состоит из одного или более блоков операционной системы. По умолчанию, размер блока в ORACLE 1 lg равен 8 Кб. Размер блока данных в ORACLE может варьироваться в пределах от 2 до 16 Кб на Linux или Windows. И на некоторых других операционных системах, до 32 Кб. Размер определяется параметром DBBLOCKSIZE. Но уже не может быть изменен после создания базы данных, потому что используется для формата файлов данных, которые образуют табличное пространство SYSTEM. Блок может быть идентифицирован по номеру внутри файла данных.

Объект каталог (directory) является логической ссылкой в базе данных на каталог файловой системы сервера, где установлена БД ORACLE. Владельцем всех объектов directory в базе данных является пользователь SYS, даже если объект directory создан другим пользователем. Имена объектов directory уникальны внутри всей БД. Все объекты Directory хранятся в табличном пространстве SYS.

Поскольку файлы и каталоги в действительности располагаются ВНЕ базы данных, процессы СУБД ORACLE должны иметь необходимые права доступа к указанным каталогам файловой системы.

Пользователи.

В базе данных учетная запись пользователя не является физической структурой, но она связана важными взаимоотношениями с объектами базы данных: пользователям принадлежат объекты. Пользователь SYS владеет таблицами словаря данных, содержащими информацию об остальных структурах базы данных. Пользователю SYSTEM принадлежат представления, обращающиеся к этим таблицам словаря данных, что позволяет остальным пользователям базы данных использовать их.

Объекты в базе данных создаются с учетными записями пользователей. Для каждой учетной записи можно задать конкретное табличное пространство в качестве табличного пространства по умолчанию.

Учетные записи данных можно связать с учетными записями системы, что дает возможность пользователям получать доступ к базе данных из операционной системы, не вводя при этом пароли ни для доступа к системе, ни для доступа к базе данных. В этом случае пользователи получат доступ к объектам, которыми они владеют или к которым имеют право доступа.

Схемы.

Набор объектов, принадлежащих учетной записи пользователя, называется схемой. Можно создать пользователей, не имеющих права входа в базу данных. Такие учетные записи предлагают схему, которую можно использовать для хранения наборов объектов базы данных отдельно от схем других пользователей.

В ORACLE схема привязывается только к одному пользователю (USER) и является логическим набором объектов базы данных. Схема создается при создании пользователем

первого объекта, и все последующие объекты, созданные этим пользователем, становятся частью этой схемы.

Она может включать другие объекты, принадлежащие этому пользователю:

- таблицы последовательности, хранимые программы, кластеры, связи баз данных, триггеры,
  библиотеки внешних процедур,
  индексы, пакеты,
  хранимые функции и процедуры,
  синонимы, представления, снимки,
  объектные таблицы,
  объектные типы,
  объектные представления.

Существуют и подобъекты схемы, такие как:

- столбцы: таблиц и представлений,
- секции таблиц,
- ограничения целостности,
- триггеры, пакетные процедуры и функции и другие элементы, хранимые в пакетах (курсоры, типы и т. п).

Существуют объекты не зависимые от схемы

- каталоги,
- профили,
- роли,
- сегменты,
- табличные пространства
- пользователи.

Роли.

Для сокращения объема информации по управлению доступом и для обеспечения более гибких возможностей управления обе СУБД применяют группирование привилегий - возможность одним действием администратора предоставить разным пользователям одинаковый набор привилегий. Однако представления концепций группирования различны в наших двух СУБД.

ORACLE использует для этих целей роли. ***Роль*** — это объект базы данных, представляющий собой именованный набор привилегий, который может предоставляться пользователю или другой роли. Администратор может создавать новые роли и изменять состав ролей. Любая привилегия может быть выдана как пользователю, так и роли. Роль может быть назначена пользователям или другим ролям. Таким образом, пользователь наследует привилегии данные назначенной ему роли.

Профили.

Профили имеют двойную функцию, это реализация парольной политики и распределение ресурсов. Парольная политика исполняется всегда, контроль за использованием ресурсов осуществляется, если значение параметра RESOURCELIMIT равно TRUE, по умолчанию оно равно FALSE. Профили используются автоматически, но профиль, назначаемый всем пользователям по умолчанию, а именно пользователям SYS, SYSTEM и др., - DEFAULT очень простой.

Объекты схемы и общие объекты ORACLE.

Таблицы представляют собой механизм сохранения информации в базе данных ORACLE. Они содержат фиксированный набор столбцов, в которых описываются атрибуты объекта, с которым эта таблица работает. У каждого столбца есть имя и уникальные характеристики.

Столбец характеризуется типом данных и длиной. Для столбцов типа NUMBER можно задать дополнительные характеристики точности и масштаба. Точность определяет число значащих цифр. Масштаб показывает место десятичной точки. Определение NUMBER (9.2) показывает, что в поле столбца хранится всего 9 цифр, две из которых располагаются справа от десятичной точки. Точность по умолчанию - 38 цифр; кроме того, это максимально возможная точность.

Таблицы, принадлежащие SYS, называются таблицами словаря данных. Они содержат системный каталог, с помощью которого система управляет свой работой. Словарь данных создается набором сценариев каталогов, предоставляемых системой ORACLE. При каждой установке или модернизации базы данных необходимо запускать сценарии, создающие или модифицирующие таблицы словаря данных. При установке новой возможности в базу данных может потребоваться запуск дополнительных сценариев каталога.

Таблицы связываются друг с другом через общие столбцы. База данных реализует эти отношения с помощью так называемой ссылочной целостности. При использовании объектно-ориентированных возможностей ORACLE строки можно связывать друг с другом при помощи внутренних ссылок, называемых идентификаторами объекто. На уровне базы данных ссылочная целостность реализуется путем использования ограничений.

Временные таблицы.

Подобно регулярной таблице временная таблица является механизмом хранения данных в базе данных ORACLE. Временная таблица состоит из столбцов, имеющих типы данных и длину. В отличие от регулярной таблицы описание временной таблицы сохраняется, но данные, внесенные в таблицу, остаются в ней во время сеанса или во время транзакции. Создание временной таблицы в качестве глобальной временной таблицы обеспечивает для всех сеансов, поддерживающих соединение с базой данных, возможность видеть данную таблицу и пользоваться ею. Во время коллективных сеансов во временные таблицы можно вставлять строки данных. Однако каждая строка данных в таблице видна только для того сеанса, который вставил эту строку.

Данные, содержащиеся во временной таблице, относятся либо к сеансу, либо к транзакции, в зависимости от ключевых слов, которые используются в конструкции on commit по отношению к данной временной таблице. Можно указать on commit delete rows или on commit preserve rows.

Временные таблицы доступны начиная с ORACLE8i, и обеспечивают средство помещения результатов в буфер или набор результатов, когда во время транзакции или сеанса приложение должно выполнить несколько операторов DML. Ограничения

Вложенные таблицы и изменяемые массивы.

Вложенная таблица - это столбец (столбцы) таблицы, содержащий несколько значений в одной строке. Например, если человек имеет несколько адресов, в таблице можно создать строку, содержащую несколько значений для столбца Address и только по одному значению для остальных столбцов. Вложенные таблицы могут содержать несколько столбцов и неограниченное число строк. Другим типом сборных конструкций является изменяемый массив, в котором число строк ограничено.

Использование как вложенных таблиц, так и изменяемых массивов требует модификации синтаксиса SQL, применяемого для доступа к данным. В целом отношения между данными вложенных таблиц можно симулировать с помощью реляционных таблиц.

Кластеры.

Таблицы, с которыми часто работают совместно, можно физически хранить совместно. Для этого создается кластер, который будет их содержать. Данные таких

таблиц сохраняются вместе в кластере, что уменьшает число операций ввода/вывода и повышает производительность.

Связанные столбцы таблиц называются кластерным ключом. Кластерный ключ индексируется с помощью кластерного индекса, причем его значение сохраняется только один раз для нескольких таблиц кластера. Кластерный индекс необходимо создать до введения (insert) новых строк в таблицы кластера.

Кластеры эффективны для таблиц, с которыми часто работают совместно. В кластерах строки из отдельных таблиц сохраняются в одних и тех же блоках, так что запросы, объединяющие эти таблицы, выполняются за счет меньшего числа операций ввода/вывода, чем при сохранении таблиц по отдельности. Однако производительность операций вставки, обновления и удаления для кластерных таблиц может быть значительно ниже, чем в случае некластерных таблиц. Прежде чем объединять таблицы в кластеры, определите частоту совместного обращения к ним. Если с ними всегда работают совместно, стоит рассмотреть возможность их объединения в одной таблице, а не группировать в одном кластере.

Ограничения.

На столбцы таблицы можно налагать ограничения; при этом каждая ее строка должна удовлетворять указанному в описании ограничению. Ниже с помощью команды create table создается таблица EMPLOYEE с несколькими налагаемыми на нее ограничениями:

```
Create table EMPLOYEE(
EmpNo NUMBER (10) PRIMARY KEY,
Name VARCHAR2(40)NOT NULL,
DeptNo NUMBER(2) DEFAULT 10,
Salary NUMBER(7,2)CHECK (salary<1000000)
Birth_Date DATE,
Soc_Sec_Num CHAR(9) UNIQUE
FOREIGN KEY (DeptNo) references DEPT(DeptNo))
Tablespace USERS;
```

Прежде всего обратите внимание, что таблице дано имя EMPLOYEE (служащий). Все ее столбцы также имеют имена (EmpNo, Name и т.д.). Для каждого столбца указаны его типы данных и длина. Столбец EmpNo определен как имеющий тип NUMBER без масштаба, что эквивалентно целочисленным значениям. Столбец Name относится к типу VARCHAR2(40); поля этот столбца имеют переменную длину до 40 символов.

Первичным ключом таблицы называется столбец или набор столбцов, делающих все строки таблицы уникальными. Столбец первичного ключа определяется в базе данных с ограничением NOT NULL. Это значит, что каждая строка таблицы должна содержать значение для данного столбца; его нельзя оставить равным NULL. Ограничение NOT NULL можно применять и к другим строкам таблицы, как показано на примере столбца Name.

На столбец может налагаться ограничение DEFAULT. Оно генерирует значение столбца при включении (insert) строки в таблицу, но для него не указывается никакого значения.

Ограничение CHECK позволяет удостовериться, что значения в указанном столбце соответствуют определенному критерию (в данном случае значения в столбце Salary меньше 1000000). Ограничение NOT NULL воспринимается базой данных как частный случай ограничения CHECK.

Ограничение UNIQUE гарантирует уникальность столбца, который должен быть уникальным, но не является частью первичного ключа. В данном примере это ограничение имеет столбец SocSecNum, поскольку значение этого столбца для каждой записи таблицы должно быть уникальным.

Ограничение внешнего ключа определяет природу взаимоотношений между таблицами. Внешний ключ одной таблицы ссылается на первичный ключ, который был ранее определен где-то в другом месте базы данных. Например, если первичным ключом таблицы DEPT является поле DeptNo, то записи этой таблицы должны содержать все допустимые значения DeptNo. Столбец DeptNo таблицы EMPLOYEE из нашего примера ссылается на столбец DEPT.DeptNo. Определив столбец EMPLOYEE.DeptNo как внешний ключ для столбца DEPT.DeptNo, вы гарантируете, что в таблицу EMPLOYEE нельзя будет ввести ни одного значения DeptNo, если оно уже существует в таблице DEPT.

Ограничения в базе данных позволяют гарантировать ссылочную целостность информации. Это дает уверенность в том, что все ссылки в базе данных достоверны и все ограничения соблюдены.

Последовательности.

Определение последовательностей (sequences) содержится в словаре данных. Последовательности позволяют упростить процесс программирования, поскольку предоставляют последовательный список уникальных номеров.

При первом обращении к последовательности в запросе она возвращает предопределенное значение. Каждый следующий запрос возвращает значение, которое больше предыдущего на указанное приращение. Последовательности могут быть циклическими, а могут увеличиваться до достижения заданного максимального значения.

При работе с последовательностями нельзя дать гарантии, что вы получите непрерывную строку значений. Например, при запросе следующего значения из последовательности для использования в команде insert вам принадлежит единственный сеанс, и только он может использовать это значение. Если вы не сможете завершить транзакцию, это значение не будет включено в таблицу, а дальнейшие вставки будут использовать следующие значения из последовательности.

Индексы.

Для того чтобы ORACLE мог найти данные, каждая строка в каждой таблице помечается с помощью идентификатора RowED. Этот идентификатор содержит информацию о том, где конкретно расположена строка (файл, блок внутри этого файла и строка внутри этого блока).

ВНИМАНИЕ: Таблица, организованная по индексу, не содержит традиционных для ORACLE идентификаторов RowlD. Вместо этого в качестве логических идентификаторов используется первичный ключ.

Индекс - это структура базы данных, используемая сервером для быстрого поиска строки в таблице. Существуют три типа индексов: кластерные, табличные и индексы битовой карты, или битовые индексы. Кластерные индексы содержат значения ключей кластеров в кластерах. Табличный индекс содержит значения строк таблиц вместе с физическим расположением строки **(RowlD).** Битовый индекс является особым типом табличного индекса, предназначенным для поддержки запросов больших таблиц со столбцами, содержащими несколько отдельных значений.

Каждое поле индекса состоит из ключевого значения и идентификатора RowlD. Можно проиндексировать один или несколько столбцов. ORACLE сохраняет поля индекса с помощью механизма **B*tree,** который обеспечивает короткий путь доступа к ключевому значению: когда запрос обращается к индексу, он находит поля индекса, соответствующие критерию запроса. Значение RowlD соответствующего поля указывает на физическое расположение связанной с ним строки - все это уменьшает объем операций ввода/вывода, требуемых для локализации данных.

Индексы повышают производительность, а также могут (при желании) обеспечивать уникальность столбца. СУБД ORACLE автоматически создает индекс, если в команде create table указывается ограничение UNIQUE или PRIMARY KEY. С помощью команды create index можно вручную создать и свои собственные индексы.

Индексы можно создавать на основе одного или нескольких столбцов таблицы. В приведенном выше примере таблицы EMPLOYEE автоматически будут созданы уникальные индексы для столбцов EmpNo и Soc_Sec_Num, поскольку они были определены с ограничениями PRIMARY KEY и UNIQUE. Удаление индекса не оказывает влияния на данные ранее проиндексированной таблицы.

Секции (разделы и подразделы).

По мере увеличения размера таблиц процесс управления ими все более усложняется. Администрирование большими базами данных можно значительно упростить, распределив данные одной большой таблицы по нескольким меньшим, например по времени, отделам или по наименованиям продукции.

Разделы доступны только для компаний, которые установили средство Partitioning.

В базе данных можно определять диапазоны, используемые при разбиении большой таблицы на меньшие. Как правило, меньшими таблицами (разделами) легче управлять, чем большими. Например, можно усечь данные одного раздела, не затрагивая остальных. ORACLE рассматривает разбитую на разделы таблицу как одну большую таблицу, но разделами можно управлять как отдельными объектами.

Разделы могут также повысить производительность приложения. Поскольку оптимизатор будет знать значения диапазона, используемые в качестве базиса для разделов при работе с таблицей, он сможет направлять запросы сразу в конкретные разделы. Поэтому при обработке каждого запроса будет считываться меньшее количество данных и производительность обработки запроса повысится.

Индексы можно также разбить на разделы. Диапазоны значений разделов такого разделенного индекса могут соответствовать диапазонам, используемым для проиндексированной таблицы, — в этом случае индекс называется локальным. Если разделы индекса не соответствуют диапазонам значений разделов таблицы, индекс называется глобальным.

В версии ORACLE8i разделы можно также разбивать, создавая подразделы. Например, можно разбить таблицу на разделы, основываясь на одном наборе значений, а затем разбить разделы с помощью другого метода разделения. Начиная с версии ORACLE9i, помимо разделения диапазона и хеширования, можно создать разделение списка.

Снимок - это только-читаемая копия таблицы или данных из нескольких таблиц. Снимок периодически освежается, чтобы отразить последнее согласованное состояние таблиц, которые он отображает. Снимки содержатся в схеме пользователя. Имя снимка должно быть уникальным по отношению к другим объектам в этой схеме.

Поддерживайте имена снимков не длиннее 23 символов. Если имя снимка содержит более 23 символов, ORACLE автоматически (и уникально) усекает префиксованные имена таблиц и обзоров, реализующих снимок, чтобы они удовлетворяли правилам именования объектов схемы.

Представление - это таблица, содержащая столбцы, и обращение к нему осуществляется точно так же, как и к таблице. Однако оно не содержит данных. Концептуально представление можно считать маской, перекрывающей одну или несколько таблиц, так как столбцы представления содержатся в одной или нескольких таблицах. Но физически представления не содержат данных. Определение представления (включающее запрос, на котором оно основано, расположение его столбцов и назначенные привилегии) содержится в словаре данных.

При обращении к представлению оно обращается к таблицам, на которых основано, и возвращает значения в формате и порядке, указанном в его определении. Поскольку с представлением не связано непосредственно никаких физических данных, оно не может быть проиндексировано.

Представления часто используются для обеспечения безопасности данных уровня строки или столбца. Например, можно предоставить пользователю доступ только **к** такому представлению, которое показывает из таблицы лишь строки этого пользователя, не открывая при этом доступа ко всем строкам таблицы. Таким же способом можно ограничить видимые пользователем столбцы.

Материализованные представления позволяют предоставлять пользователям локальные копии удаленных данных. Материализованное представление основывается на запросе, использующем связь базы данных, которая позволят выбирать данные из удаленной БД. Затем пользователи могут обращаться к этому материализованному представлению, или же оптимизатор может динамически перенаправлять запросы, чтобы использовать материализованные представления вместо исходной таблицы. Эта функциональная возможность называется повторным построением запроса; ее применение разрешается с помощью параметра инициализации. Материализованные представления можно реализовывать как "только для чтения" или как "обновляемые". Для повышения производительности можно проиндексировать локальную таблицу, используемую материализованным представлением.

В зависимости от сложности базового запроса материализованного представления можно с помощью журнала материализованных представлений повысить производительность операций репликации. Эти операции выполняются автоматически в зависимости от схемы, определенной для каждого материализованного представления.

Хранимые процедуры и функции.

Процедура - это блок операторов PL/SQL, сохраняемый в словаре данных и вызываемый приложениями. Процедуры позволяют сохранять в базе данных часто используемую логику приложений. При выполнении процедуры все ее операторы выполняются как единое целое. Процедуры не возвращают никаких значений вызвавшей их программе.

Сохраняемые процедуры позволяют поддерживать безопасность данных. Вместо предоставления пользователю доступа к таблицам в приложении можно разрепшть ему выполнить только процедуру, осуществляющую доступ к этим таблицам. При выполнении процедура использует привилегии ее владельца. В результате пользователи не смогут получить доступ к таблице, иначе как посредством процедуры.

Функции, как и процедуры, представляют собой блоки кода, сохраняемые в базе данных. Однако в отличие от процедур функции могут возвращать значения вызвавшей их программе. Можно создавать свои собственные функции и обращаться к ним в операторах SQL, а можно использовать только те функции, которые предоставляются средой ORACLE.

Например, ORACLE предоставляет функцию SUBSTR, выбирающую подстроку из строки символов. Если создать функцию MYSUBSTR, выполняющую специальные операции подстроки, то обратиться к ней можно с помощью команды SQL:

```
select MY_SUBSTR ( text ) from DUAL,
```

Если вы не владеете функцией MYSUBSTR, следует получить на нее разрешение EXECUTE. Определенную пользователем функцию можно применять в операторе SQL, только если она не изменяет строки базы данных.

Пакеты

С помощью пакетов можно упорядочить процедуры и функции и объединять их в логические группы. Спецификации и тела пакетов сохраняются в словаре данных. Пакеты бывают очень полезны при решении задач администрирования по управлению процедурами и функциями.

Элементы пакета бывают публичными, или общими (public), и частными (private). Публичные элементы доступны пользователю пакета, а частные скрыты от него. Частные

элементы могут включать, например, процедуры, вызываемые другими процедурами пакета.

Исходный код функции, пакетов и процедур сохраняется в таблицах словаря данных. Если приложение активно использует пакеты, может потребоваться существенно увеличить размер табличного пространства SYSTEM, чтобы оно соответствовало увеличившемуся словарю данных.

Число и сложность используемых пакетов непосредственно влияют на размер фрагмента Shared SQL Pool системной глобальной области.

Триггеры - это процедуры, выполняемые при наступлении указанного события базы данных. С помощью триггеров можно укрепить ссылочную целостность, обеспечить дополнительную безопасность или повысить доступные возможности аудита.

Существуют два типа триггеров:

1. Операторные триггеры. Срабатывают один раз для каждого активизирующего оператора.
2. Строковые триггеры. Срабатывают один раз для каждой строки таблицы, на которую влияют данные операторы.

Скажем, триггер уровня оператора срабатывает один раз для команды delete, удаляющей 10000 строк. В той же самой транзакции триггер уровня строки должен был бы сработать 10000 раз.

Для любого типа триггера можно создать триггеры BEFORE (до) и AFTER (после), относящиеся к каждому типу активизирующих событий. К числу таких событий относятся команды insert, update и delete.

Операторные триггеры полезны, если код запускаемого триггера не использует затрагиваемые данные. Например, в таблице можно создать операторный триггер BEFORE INSERT, который не позволит осуществить вставку в таблицу, кроме как в специальные периоды времени.

Строковые триггеры полезны, если код действия применяет данные, затрагиваемые транзакцией. Например, можно создать строковый триггер AFTER INSERT, чтобы вставить новые строки в таблицу аудита или в базовую таблицу триггера.

В версии ORACLE8 можно создавать триггер INSTEAD OF. Он выполняется вместо действия, вызвавшего его запуск. Например, если в таблице задан триггер INSTEAD OF INSERT, будет выполняться его код, а вставка, которая вызывает работу этого триггера, никогда не произойдет. Триггеры INSTEAD OF можно применять к представлениям. Если представление объединяет в своем запросе несколько таблиц, триггер INSTEAD OF позволит управлять действиями ORACLE при попытке пользователей обновить строки через это представление.

Новый триггер под названием on logon дает возможность получить значения, которые задаются пользователем с момента установки его соединения с базой данных.

Таблицы объектов

Таблицей объектов называется таблица, все строки которой являются объектами, и для них определены значения идентификаторов объектов. Для создания таблицы объектов можно использовать команду create table. Например, в следующем листинге команда create table используется для создания таблицы NAME на основе типа данных NAMETY:

```
create table NAME of NAME_TY,
```

Затем можно создать ссылки из других таблиц на объекты строк таблицы объектов NAME. Это позволит выбирать строки данной таблицы по ссылкам, не обращаясь к таблице NAME непосредственно.

Объектные представления

Используя абстрактные типы данных, можно столкнуться с проблемами непротиворечивости при их реализации. Доступ к атрибутам абстрактных типов данных результате для поддержки абстрактных типов данных может потребоваться изменить стандарты кодирования SQL вашего предприятия. Кроме того, нужно помнить, какие таблицы используют абстрактные типы данных при выполнении транзакций и запросов в них.

Объектные представления облегчают работу с абстрактными типами данных. С помощью объектных представлений можно дать объектно-реляционную презентацию реляционным данным. Хотя подлежащая таблица не изменяется, представление будет поддерживать определения абстрактных типов данных. С точки зрения администратора базы данных, изменения здесь невелики - управлять базой данных вы будете так же, как и любой другой таблицей в базе данных. С точки зрения разработчика, объектные представления предоставляют объектно-реляционный доступ к данным таблиц.

Синонимы.

Для того чтобы полностью идентифицировать объект в распределенной базе данных (например, таблицу или представление), нужно указать имя хост-компьютера, имя (экземпляр) сервера, владельца объекта и имя этого объекта. В зависимости от расположения объекта может потребоваться от одного до всех четырех этих параметров. Чтобы скрыть весь этот процесс от пользователя, можно создать указывающие на объект синонимы; в результате пользователю нужно будет знать только имя синонима. Публичные синонимы используются совместно всеми пользователями базы данных. Частные синонимы принадлежат индивидуальным собственникам учетных записей базы данных.

Например, у рассмотренной выше таблицы EMPLOYEE должен быть владелец - назовем его HR. Из учетной записи другого пользователя той же базы данных на данную таблицу можно сослаться как на HR.EMPLOYEE. Однако такой синтаксис не предполагает, что вторая учетная запись знает о том, что собственникам таблицы EMPLOYEE является учетная запись HR. Чтобы преодолеть это, необходимо создать синоним EMPLOYEE, указывающий на учетную запись HR.EMPLOYEE. Ссылка на этот синоним означает указание на соответствующую таблицу. Синоним EMPLOYEE создает следующий оператор SQL:

```
create public synonym EMPLOYEE for HR EMPLOYEE
```

Синонимы применяются в качестве указателей на таблицы, процедуры, представления, функции, модули и последовательности. Они могут указывать на объекты в локальной или в удаленной базе данных. Указание на удаленную базу данных требует использования ссылок базы данных .

Синонимы для абстрактных типов создавать нельзя. Более того, ORACLE не проверяет допустимость синонима при его создании. Создав синонимы, следует их проверить и убедиться, что их можно использовать.

Связи баз данных

В базах данных ORACLE можно ссылаться на данные, находящиеся вне локальной базы данных. При этом необходимо указать полное имя удаленного объекта. В описанном ранее примере синонима было определено только две части полного имени — имя владельца и имя таблицы. Однако что делать, если таблица содержится в удаленной базе данных?

Для того чтобы определить путь доступа к объекту из удаленной базы данных, необходимо создать связь базы данных. Такая связь может быть либо публичной (доступной всем учетным записям из этой базы данных), либо частной (созданной пользователем только для своей записи). При создании связи базы данных необходимо определить имя и пароль учетной записи для соединения с ней, а также название службы, связанной с удаленной базой данных. Если не указать имя учетной записи, то для

соединения с удаленной базой данных ORACLE воспользуется вашим локальным именем учетной записи и паролем. В следующем примере показано создание публичной связи MY_LINK:

```
create public database link MY_LINK connect to HR identified
by PUFFINSTUFF using  DBl
```

В этом примере связь откроет сеанс в базе данных, которая идентифицирована службой с именем DB1. Открыв сеанс в этом экземпляре DB1, она зарегистрируется как пользователь с учетной записью HR и паролем puffinstuff. Имена служб для экземпляров сохраняются в файлах конфигураций, используемых ORACLE NET. Файл конфигураций для имен служб называется tnsnames.ora и указывает хост, порт и экземпляр, связанный с каждым именем службы.

Для использования этой связи в таблице ее необходимо указать в конструкции from:

```
select * from EMPLOYEE@MY_LINK;
```

Данный запрос получает доступ к таблице EMPLOYEE через связь базы данных MYLINK. Для этой таблицы можно создать синоним, как показано в следующей команде SQL:

```
create synonym EMPLOYEE for EMPLOYEE@MYLINK,
```

Обратите внимание, что для объекта базы данных было указано полное название: хост и экземпляр — через имя службы, владелец (HR) — через связь базы данных, а также имя (EMPLOYEE).

Таким образом, для конечного пользователя локализация таблицы EMPLOYEE будет полностью прозрачна. Эту таблицу можно перенаправить на другую схему или в другую базу данных; изменение описания связи базы данных переадресует синоним в новое место.

Если сохраняемая процедура, модуль или триггер содержат ссылку на базу данных, существование связи необходимо для компиляции кода PL/SQL.

Средства манипулирования данными языка SQL

ORACLE поддерживает 4 стандартных оператора манипулирования данными:

1. **INSERT** - используется для ввода данных;
2. **SELECT** - используется для выборки данных;
3. **UPDATE** - используется для обновления данных;
4. **DELETE** - используется для удаления данных.

Ввод данных.

Оператор INSERT используется для добавления строк в таблицу. Вы можете указать следующую информацию при использовании оператора INSERT:

- Таблица, в которую необходимо добавить строку.
- Список столбцов, для которых будут заданы значения.
- Список значений, которые будут храниться в указанных столбцах.

Во время добавления строки необходимо указать значения для первичного ключа и всех других столбцов, которые определены как NOT NULL. Нет необходимости указывать значения для остальных столбцов; им автоматически будет присвоено значение NULL. Выполните следующую инструкцию INSERT, которая добавляет строки в таблицу CUSTOMERS, обратите внимание на то, чтобы порядок вводимых значений совпадал с порядком списка столбцов:

SQL> **INSERT INTO** customers (customer_id, first_name, last_name, dob, phone) **VALUES** (6, 'Fred', 'Brown', '01-JAN-1970', '800-555-1215');
1 row created.

Внимание: SQL*Plus автоматически нумерует строки после каждого нажатия клавиши ENTER.

Каждый раз после выполнения инструкции INSERT выводится сообщение «1 row created». Таким образом ORACLE информирует о том, что данная таблица заполняется вводимыми значениями.

Простые манипуляции по выборке данных из одной таблицы.

Оператор SELECT используется для выборки данных из таблиц базы данных. В самом простом примере вы указываете таблицу и столбцы, которые вам необходимо выбрать из базы данных Следующий пример выбирает столбцы CUSTOMER_ID, FIRST_NAME, LAST_NAME, DOB и PHONE из таблицы CUSTOMERS.

```
SELECT customer_id, first_name, last_name, dob, phone
FROM customers;
```

Сразу за ключевым словом SELECT указываются имена столбцов, которые вам необходимо получить, а после ключевого слова FROM указывается имя таблицы. Оператор языка SQL заканчивается точкой с запятой (;). Операторы SELECT часто называют запросами.

```
SELECT <список столбцов>
FROM <список таблиц>;
```

Нет необходимости указывать программному обеспечению базы данных, каким образом извлечь информацию, которая вам необходимо. Необходимо указать, что вам нужно, и позволить программному обеспечению самому решить, как это выполнить. Части, которые следуют сразу же после ключевого слова SELECT, не обязательно всегда должны быть именами столбцов из таблицы: Они также могут быть любыми допустимыми SQL выражениями.

CUSTOMER_ID	*FIRST_NAME*	*LAST_NAME*	*DOB*	*PHONE*
1	John	Brown	01-JAN-65	800-555-1211
2	Cynthia	Green	05-FEB-68	800-555-1212
3	Steve	White	16-MAR-71	800-555-1213
4	Gail	Black	800-555-1215	
5	Doreen	Blue	20-MAY-70	

Таблица 2.1 Результат выполнения запроса.

Строки, которые были возвращены базой данных, также известны как результирующая выборка. Как вы видите, База данных ORACLE преобразует имена столбцов в их эквиваленты с заглавными буквами. Столбцы строковых значений и дат выровнены по левому краю; столбцы чисел - по правому краю. База данных ORACLE отображает даты в формате DD-MON-YY, где DD - день недели, MON - три первые буквы месяца заглавными буквами, YY - последние две цифры года. На самом деле база данных хранит все четыре цифры для обозначения года, но по умолчанию отображаются последние две цифры.

Определение выводимых строк с помощью ключевого слова WHERE.

Если необходимо вывести только определенные строки, необходимо добавить к оператору SELECT ключевое слово WHERE. Это очень важно, поскольку ORACLE вмещает огромное количество строк в таблице, можно получить небольшое подмножество этих строк. Для этого необходимо после ключевого слова FROM и имени таблицы поместить ключевое слово WHERE и указать условие отбора:

```
SELECT <список столбцов>
FROM <список таблиц>
WHERE <условие отбора>;
```

В следующем запросе выражение с ключевым словом WHERE используется для получения строк из таблицы CUSTOMERS со значением столбца CUSTOMER_ID, равным2:

```
SELECT *
FROM customers
WHERE customer id = 2;
```

CUSTOMER_ID	*FIRST_NAME*	*LAST_NAME*	*DOB*	*PHONE*
2	Cynthia	Green	05-FEB-68	800-555-1212

Таблица 2.2 Результат выборки по условию.

Как базы данных представляет значения, которые неизвестны? Для этого используется специальное значение, которое называется NULL значение. NULL значение - это не пустая строка, а особое значение, определенное для столбцов, значения которых не известны.

```
SELECT *
FROM customers;
```

CUSTOMER_ID	*FIRST_NAME*	*LAST_NAME*	*DOB*	*PHONE*
1	John	Brown	01-JAN-65	800-555-1211
2	Cynthia	Green	05-FEB-68	800-555-1212
3	Steve	White	16-MAR-71	800-555-1213
4	Gail	Black	800-555-1215	null
5	Doreen	Blue	20-MAY-70	null

Таблица** 2.3 NULL **значения в результате выборки.

Для того чтобы выбрать строки, у которых значения столбцов не определены, используется IS NULL в запросах, как в следующем примере:

```
                        first name, last name, phone
SELECT customer_id,
FROM customers WHERE
phone IS NULL;
```

	FIRST_NAME	*LAST_NAME*	*PHONE*
5	Doreen	Blue	null

Таблица 2.4 Выбор по** NULL **значению

Поскольку часто не NULL значение не отображается, как различать NULL значение от пустой строки? Для этого можно использовать встроенную функцию ORACLE NVL () .NVL () принимает два параметра: столбец и значение, возвращаемое вместо NULL.

В следующем примере **NVL** () возвращает строку 'Unknown phone number', когда значение столбца PHONE содержит NULL значение:

```
SELECT customer_id, first_name, last_name,
NVL(phone, 'Unknown phone number') AS PHONE_NUMBER FROM customers;
```

CUSTOMER_ID	*FIRST_NAME*	*LAST_NAME*	*DOB*	*PHONE*

1	John	Brown	01-JAN-65	800-555-1211
2	Cynthia	Green	05-FEB-68	800-555-1212
3	Steve	White	16-MAR-71	800-555-1213
4	Gail	Black	800-555-1215	800-555-1214
5	Doreen	Blue	20-MAY-70	Unknown phone number

***Таблица 2.5 Результат работы функции** NVLQ.*

Следующая таблица операторов может быть использована для сравнения значений в условии отбора:

Оператор	**Описание**
=	Равно
<> или !=	Не равно
<	Меньше
>	Больше
<=	Меньше или равно
>=	Больше или равно
ANY	Сравнивает значение с любыми значениями из списка
SOME	Идентично оператору ANY; используется реже, чем ANY
ALL	Сравнивает значение со всеми значениями в списке.

Таблица 2.6 Операторы сравнения.

В следующем примере оператор (о) используется для получения строк из таблицы CUSTMERS, у которых значение столбца CUSTOMER_ID не равно 2:

SELECT * FROM customers **WHERE** customer id <> 2;

CUSTOMER_ID	***FIRST_NAME***	***LAST_NAME***	***DOB***	***PHONE***
1	John	Brown	01-JAN-65	800-555-1211
3	Steve	White	16-MAR-71	800-555-1213
4	Gail	Black	800-555-1215	
5	Doreen	Blue	20-MAY-70	

Таблица 2.7 Результат выполнения запроса.

В следующем примере используется оператор > для получения значений столбцов PRODUCT_ID и NAME из таблицы PRODUCTS, у которых значение столбца PRODUCT_ID больше 8:

```
SELECT product_id, name FROM products
WHERE product id > 8;
```

PRODUCT ID	**NAME**
9	Classical Music
10	Pop3
11	Creative Yell
12	My Front Line

***Рис. 2.8 Результат выполнения запроса с оператором** >.*

Как уже было сказано, ключевое слово WHERE указывает ORACLE произвести поиск в таблице и вернуть значения только тех строк, которые удовлетворяют указанным критериям (условиям). Часто необходимо получить строки, которые удовлетворяют нескольким критериям (условиям). Например если необходимо получить список CUSTOMERS из города UTAH, кто также имеет CREDIT_LIMIT более 10,000$. SQL запрос будет выглядеть следующим образом:

```
SELECT cust_id, cust_state_province, cust_credit_limit
FROM customers
WHERE cust_state_province = "UTAH1
AND cust credit limit > 10000;
```

CUST ID	CUST_STATE_PROVTNCE	CUST CREDIT LIMIT
50601	**UT**	11000
24830	**UT**	15000
28983	**UT**	15000
100267	**UT**	11000
100207	**UT**	11000
103007	**UT**	15000

Таблица 2.9 Результат выполнения запроса, удовлетворяющего обоим условиям.

В предыдущем примере были выбраны строки, которые удовлетворяют обоим условиям. Ну а если необходимо выбрать строки, которые удовлетворяют любому из

C

ортировка. условий, например: выбрать строки, где категория продукта WEIGHT CLASS равен 4, то необходимо выполнить следующий запрос:

```
SELECT prod_id, prod_category, prod_weight_class WGT
FROM products
WHERE prod_category = 'Hardware'
OR prod_weight_class = 4;
```

PROD ID	PROD CATEGORY	WGT
15	Hardware	1
18	Hardware	1
139	Electronics	4

Таблица 2.10 Результат выполнения запроса, удовлетворяющего любому из условий.

В ORACLE существует также возможность получить строки, не удовлетворяющие условию. Конечно в некоторых случаях возможно использовать оператор != или <>, однако лучше использовать оператор NOT. Например: вывести всю продукцию, чья WEIGHT_CATEGORY не равна 1:

```
SELECT prod_id, prod_category, prod_weight_class WGT
FROM products
WHERE prod_weight_class != 1;
```

Сортировка данных.

Для сортировки выбираемых строк используется ключевое слово ORDER BY. При использовании ORDER BY можно указать один и более столбцов по которым необходимо отсортировать полученные строки. Выражение ORDER BY должно следовать за выражением FROM или WHERE (если с помощью WHERE указывается условие отбора). Следующий пример использует ORDER BY для сортировки строк из таблицы CUSTOMERS по столбцу LAST_NAME:

```
SELECT * FROM customers ORDER BY last name;
```

CUSTOMER ID	FIRST NAME	LAST NAME	DOB	PHONE
4	Gail	Black	800-555-1214	
5	Doreen	Blue	20-MAY-70	
1	John	Brown	01-JAN-65	800-555-1211
2	Cynthia	Green	05-FEB-68	800-555-1212
3	Steve	White	16-MAR-71	800-555-1213

Таблица 2.15 Сортировка данных с помощью ORDER BY/

По умолчанию ORDER BY сортирует значения столбца в порядке возрастания (от меньшего - к большему). Однако можно с помощью ключевого слова DESC отсортировать значения столбца в порядке убывания (от большего - к меньшему). Ключевое слово AS C может быть использовано для явного указания сортировки по возрастания. Следующий запрос использует ORDER BY для сортировки строк из таблицы CUSTOMERS для столбца FIRST_NAME по возрастанию и для столбца LAST_NAME - по убыванию:

```
SELECT *
FROM customers
ORDER BY first name ASC, last name DESC;
```

CUSTOMER ID	FIRST NAME	LAST NAME	DOB	PHONE
2	Cynthia	Green	05-FEB-68	800-555-1212
5	Doreen	Blue	20-MAY-70	
4	Gail	Black	800-555-1214	
1	John	Brown	01-JAN-65	800-555-1211
3	Steve	White	16-MAR-71	800-555-1213

Таблица 2.16 Сортировка с использованием ASC и DESC.

Внимание: Указывать столбцы, по которым необходимо сортировать данные можно указывать использую, порядковые номера.

Обновление данных.

Для изменения данных в таблице используется оператор UPDATE. При использовании оператора UPDATE обычно указывается следующая информация:

- Имя таблицы
- Выражение WHERE, определяющее какие строки будут изменены.
- Список столбцов и их значений, определенных с помощью ключевого слова SET.

С помощью одного и того же запроса UPDATE можно изменить одну и несколько строк. При изменении нескольких строк нужно помнить, что новое значение будет применено во всех строках. Например, следующий запрос UPDATE устанавливает для столбца las t_name значение Orange в строке, чей customer_id равен 2.

```
UPDATE customers SET last_name = 'Orange' WHERE customer_id = 2;

1 row updated.
```

SQL*Plus подтверждает, что строка была успешно обновлена. В случае, если выражение WHERE будет опущено, то будут обновлены все строки.

CUSTOMER ID	FIRST NAME	LAST NAME	DOB	PHONE
2	Cynthia	Green	05-FEB-68	800-555-1212

Таблица 2.17 Измененная строка.

Удаление данных.

Для удаления строк используется оператор DELETE. Обычно необходимо указать с помощью выражения WHERE строки, которые необходимо удалить; в противном случае будут удалены все строки.

Следующий запрос DELETE удаляет из таблицы покупателей строку, в которой customerid равен 10:

```
DELETE FROM customers
WHERE customer_id = 10; 1
row deleted.
```

SQL*Plus подтверждает, что одна строка была удалена.

Внимание: При использовании операторов INSERT, UPDATE и DELETE необходимо фиксировать транзакции с помощью оператора COMMIT, т.к. до момента выхода или завершении сессии измененные значения хранятся только во временном пространстве, а не в постоянной базе данных. Если после подтверждения изменений необходимо вернуть данные в исходное состояние можно выполнить оператор ROLLBACK.

Вопросы к лекции

1. Какие виды основных объектов вы знаете?
2. Какие средства манипулирования данными применяются в языке SQL?
3. Использование SQL операторов при формировании запросов.
4. Каким образом происходит сортировка данных?

Список используемой литературы

1. Кевин Луни, Марлен Терьо «ORACLE 9i. Настольная книга администратора» изд. «ORACLE Press» перев. «Лори» 2004г. 750 стр.
2. Ian Abramson, Michael Abbey, Michael J. Corey, Michelle Malcher «ORACLE Database 1 lg. A Beginner's Guide» изд. «McGrowHill» 2008r. 415 стр.
3. Том Кайт «ORACLE для профессионалов. Книга 1. Архитектура и основные особенности.» изд. «APress» перев. «ДиаСофтЮП» 2003г 672 стр.
4. Jason Price «Oracle Database 1 lg SQL» «McGrowHill» 2007r. 656 стр.

Лекция 3. **Язык описания данных ORACLE. Типы и объекты данных в ORACLE. Символьные строки. Числовые типы. Тип ROWID. Битовые строки. Тип дата и время.**

План

1. Язык описания данных ORACLE
2. Создание таблиц с помощью оператора CREATE.
3. Типы данных в ORACLE

Язык описания данных ORACLE

Data Definition Language (DDL) (язык описания данных) - это семейство компьютерных языков, используемых в компьютерных программах для описания структуры баз данных. В базах данных **DDL** является подмножеством SQL, используемым для определения и модификации различных структур данных.

К данной группе относятся команды, предназначенные для создания, изменения и удаления различных объектов базы данных. Команды CREATE (создание), ALTER (модификация) и DROP (удаление) работают с большинством типов объектов баз данных (таблиц, представлений, процедур, триггеров, табличных областей, пользователей и др.). Также к этой группе относятся операторы RENAME и TRUNCATE.

Создание таблиц с помощью оператора CREATE.

Базы данных предназначены для хранения информации, а она содержится в таблицах. Таблицы, создаваемые в ORACLE обычно:

- хранят данные различных типов, например, текст, числа и даты;
- ограничивают длину вводимых данных;
- запрещают ввод записей, в которых не заполнены определенные столбцы;
- гарантируют, что значения, введенные в определенные столбцы, находятся в допустимом диапазоне;
- имеют логическую связь с другими таблицами.

Именование таблиц и столбцов

Присваивая имена таблицам и столбцам, вы должны следовать определенным правилам. Некоторые из них носят обязательный характер, тогда как другие представляют собой рекомендации, помогающие придать таблицам профессиональный вид.

Перечисленные ниже правила обязательны для любой таблицы или столбца:

1. Максимальная длина имени таблицы или столбца равна 30 символам.
2. Имена таблиц и столбцов могут содержать буквы, цифры и символ подчеркивания (__). (Есть еще пара специальных символов, которые можно использовать в случае острой необходимости, но в обычной работе это не принесет ничего, кроме проблем, поэтому лучше ограничиться буквами, цифрами и символом подчеркивания.)
3. Имена таблиц и столбцов должны начинаться с буквенного символа.
4. Имя может содержать цифры или символы подчеркивания, но в любом случае должно начинаться с буквы.
5. Символы верхнего и нижнего регистров в именах таблиц и столбцов считаются одинаковыми.
6. Имя таблицы или столбца не должно содержать пробелы.

7. В ORACLE таблицы присваиваются пользователям; по умолчанию они присваиваются тому пользователю, который их создал. Каждая из таблиц должна иметь имя, отличное от имен других таблиц этого пользователя. Иными словами, у пользователя не может быть двух таблиц с одним и тем же именем. (Однако разные пользователи могут без проблем создавать таблицы с одинаковыми именами.) Все столбцы в пределах таблицы должны иметь уникальные имена.

8. Некоторые слова представляют собой команды и параметры ORACLE, а следовательно, не могут использоваться в качестве имен таблиц или столбцов.

Для создания таблиц используется оператор CREATE TABLE, имеющий следующий общий формат:

```
CREATE TABLE [schema.] table
 ( column datatype [DEFAULT expr] [column_constraint] [, {
column datatype [DEFAULT expr] [column_constraint] ... |
CONSTRAINT constraint }]...) [TABLESPACE tab_space];
```

Описание column_constraint:

```
[NOT NULL]
[PRIMARY KEY]
[FOREIGN KEY REFERENCES table (columns) [ON {UPDATE | DELETE} { CASCADE | SET DEFAULT | SET NULL |NO ACTION }]]
[UNIQUE] [CHECK (condition)]
```

Эта версия оператора CREATE TABLE включает средства определения ограничений ссылочной целостности и других ограничений. В результате выполнения этого оператора будет создана таблица, имя которой определяется параметром table, состоящая из одного или нескольких столбцов column типа datatype.

Схема schema это набор таблиц и других объектов, которым владеет один пользователь, чье имя совпадает с именем схемы. Если при выполнении данного запроса не будет указана схема, то таблица будет создана в схеме пользователя, выполняющего запрос. В дополнение к этому пользователь выполняющий запрос CREATE TABLE должен иметь соответствующие привилегии для создания таблиц в чужих схемах.

Таблица создается в табличном пространстве tab_space, но если в запросе опущено упоминание о табличном пространстве, то по умолчанию таблица создается табличном пространстве владельца таблицы.

Для задания значения, применяемого по умолчанию при вставке данных в конкретный столбец, предусмотрена необязательная конструкция DEFAULT. В базе данных это значение применяется по умолчанию в тех случаях, если в операторе INSERT не задано значение для такого столбца. Кроме прочих значений, опция определения применяемого по умолчанию значения DEFAULT может включать литералы. Остальные конструкции известны под названием ограничений таблицы и могут быть дополнительно обозначены с помощью следующей конструкции:

```
[PRIMARY KEY (column)
| FOREIGN KEY (columns) REFERENCES table (columns) [ON {UPDATE | DELETE} {CASCADE | SET DEFAULT | SET NULL | NO ACTION }]
| UNIQUE (column) j
CHECK (condition)
```

Конструкция NOT NULL не позволяет оставлять столбец пустым, UNIQUE проверяет уникальность значения внутри столбца, а CHECK проверяет значение столбца на соответствие определенному условию.

Конструкция PRIMARY KEY определяет один или несколько столбцов, которые образуют первичный ключ таблицы. По умолчанию для всех столбцов, представляющих первичный ключ, предусмотрено применение ограничения NOT NULL и UNIQUE. При создании таблицы разрешено использование только одной конструкции PRIMARY KEY. База данных отвергает все попытки выполнения операций INSERT или UPDATE, которые влекут за собой создание строки с повторяющимся значением в столбце (столбцах) PRIMARY KEY. Таким образом, в базе данных гарантируется уникальность значений первичного ключа.

В конструкции FOREIGN KEY определяется внешний ключ (дочерней) таблицы и ее связь с другой (родительской) таблицей. Эта конструкция позволяет реализовать ограничения ссылочной целостности и состоит из следующих частей:

- Список столбцов columns, содержащий имена одного или нескольких столбцов создаваемой таблицы, которые образуют внешний ключ.
- Вспомогательная конструкция REFERENCES, указывающая на родительскую таблицу (т.е. таблицу, в которой определен соответствующий потенциальный ключ).
- Необязательное правило обновления (ON UPDATE) для определения взаимосвязи

между таблицами, которое указывает, какое действие должно выполняться при обновлении в родительской таблице потенциального ключа, соответствующего внешнему ключу дочерней таблицы. В качестве параметра можно указать CASCADE, SET NULL, SET DEFAULT или NO ACTION. Если конструкция ON UPDATE опущена, то по умолчанию подразумевается, что никакие действия не выполняются, в соответствии со значением NO ACTION.

- Необязательное правило удаления (ON DELETE) для определения взаимосвязи между таблицами, которое указывает, какое действие должно выполняться при удалении строки из родительской таблицы, которая содержит потенциальный ключ, соответствующий внешнему ключу дочерней таблицы. Определение параметра совпадает с определением такого же параметра для правила ON UPDATE.

По умолчанию ограничение ссылочной целостности удовлетворяется, если любой компонент внешнего ключа имеет значение NULL или в родительской таблице есть соответствующая строка. В операторе создания таблицы может быть задано любое количество конструкций FOREIGN KEY.

Например: Компания Скотта собирается отделить сотрудников с почасовой оплатой в новую таблицу. Новая таблица будет выглядеть следующим образом:

```
CREATE TABLE EMP_HOURLY (
EMPNO NUMBER (4) NOT NULL,ENAME VARCHAR2 (10),JOB
VARCHAR2(9),MGR NUMBER (4),
HIREDATE DATE,HOURRATE NUMBER (5,2) NOT NULL DEFAULT 6.50,
 DEPTNO NUMBER (2),
CONSTRAINT PK_EMP PRIMARY KEY ( EMPNO ) );
```

Для просмотра структуры таблицы очень удобно использовать команду DESCRIBE.

Изменение таблиц с помощью оператора ALTER TABLE.

Команда ALTER TABLE позволяет пользователю производить изменение объектов в базе данных. Возможности команды ALTER TABLE можно разделить три категории.

1. Добавление изменение удаление столбца.
2. Добавление и удаление ограничений.
3. Включение и выключение ограничений.

Для изменения таблицы можно упростить синтаксис до одного из трех выражений:

ALTER TABLE tablename ADD (columnl datatypel [DEFAULT expression] [, . . .]) ;

ALTER TABLE tablename MODIFY (columnl datatypel [DEFAULT expression] [,...]);

ALTER TABLE tablename DROP COLUMN columnl;

Например: политика, которая применяется в компании Скотта, имеет новое значение часовой оплаты в 7,25$. Таблица EMP_HOURLY должна быть изменена таким образом, чтобы отражать данную политику. Мы можем использовать вторую форму запроса ALTER TABLE, рассмотренную выше, чтобы выполнить поставленную задачу. Также выясняется, что один из менеджеров управляет всеми сотрудниками с почасовой оплатой, следовательно, нам не нужен столбец MGR в таблице EMP_HOURLY. Мы можем использовать третью форму для выполнения дополнительной задачи:

```
ALTER TABLE EMP_HOURLY MODIFY (HOURRATE NUMBER(5,2) DEFAULT 7.25);
ALTER TABLE EMP_HOURLY DROP COLUMN MGR;
```

Если столбец в таблице удален или изменен, значения остальных столбцов также как и общее количество записей в таблице остается тем же. Если в таблицу с существующими записями добавляется столбец, то значение для этого столбца во всех

записях устанавливается в NULL, за исключением моментов, когда значение столбца обязательно. Если значение нового столбца обязательно, то должно быть определено значение по умолчанию DEFAULT.

Для добавления или удаления ограничений необходимо знать имя этого ограничения. Синтаксис команды ALTER TABLE выглядит следующим образом:

ALTER TABLE table_name ADD CONSTRAINT constraint_name PRIMARY KEY (column [, columnl, . .]) ;

ALTER TABLE table_name ADD CONSTRAINT constraint_name UNIQUE (column [, columnl, . .]) ;

ALTER TABLE table_name ADD CONSTRAINT constraint_name FOREIGN KEY (column [, columnl, . .]) REFERENCES new_table_name (new_column [, new_columnl, …]);

В вышеприведенных запросах создается ограничение в таблице tablename с именем ограничения constraintname на столбец column. В случае со вторичным ключом new_table_name означает таблицу, на которую ссылается вторичный ключ, а new_column столбец в этой таблице.

Можно также установить ограничение CHECK.

ALTER TABLE table_name ADD CONSTRAINT constraint_name CHECK (column condition);

Где condition -условие, на которое проверятся значение столбца.

Чтобы удалить ограничение используется выражение DROP CONSTRAINT:

```
ALTER TABLE table_name DROP CONSTRAINT constraint_name;
```

Ограничение на столбец, возможно отключить при помощи выражения DISABLE CONSTRAINT:

```
ALTER TABLE table_name DISABLE CONSTRAINT constraint_name;
```

Можно использовать слово CASCADE после DISABLE CONSTRAINT чтобы отключить все ограничения, которые зависят от ограничения constraint name. Можно использовать CASCADE при отключении ограничения первичного ключа, которое является частью вторичного ключа в другой таблице.

Удаление таблиц с помощью оператора DROP TABLE.

Когда таблица больше не нужна, ее можно удалить. Удаляются и сама таблица и все записи, хранящиеся в ней, а пространство, выделенное для таблицы, становится доступным для других объектов базы данных. Синтаксис DROP TABLE очень прост:

```
DROP TABLE tablename;
```

Как и большинство DDL операторов, чтобы таблица была удалена, пользователь, выполняющий данную операцию, должен иметь соответствующие привилегии.

Изменение имени таблиц с помощью оператора RENAME.

Оператор RENAME очень прост. Имя таблицы должно быть изменено на новое имя при этом связи с данной таблицей других объектов, таких как индексы, автоматически корректируются. Синтаксис выглядит следующим образом:

```
RENAME old_tablename TO new_tablename;
```

Например, компания Скотта производит разделение сотрудников на категории: постоянные и с почасовой оплатой. По этой причине должна быть создана новая таблица EMP_HOURLY_TEMP, а старая EMP_HOURLY должна быть переименована на EMP_HOURLY_PERM:

```
RENAME EMP_HOURLY TO EMP_HOURLY_PERM;
```

Внимание любые ссылки на переименованную таблицу в программном коде (таком как C++) или в хранимых SQL скриптах должны быть вручную изменены.

Удаление всех строк таблицы с помощью оператора TRUNCATE.

С точки зрения пользователей оператор TRUNCATE очень похож на оператор DELETE. Оба оператора удаляют записи из таблицы. Основная разница состоит в том, что при использовании DELETE можно удалять строки выборочно. Также оператор TRUNCATE работает намного быстрее. Однако с точки зрения администраторов операторы TRUNCATE и DELETE очень отличаются. Оператор TRUNCATE очищает все пространство удаленных строк. Пространство записей, удаленных оператором DELETE, останется распределенным за таблицей, и в будущем может быть повторно использовано операторами INSERT. Также оператор TRUNCATE нельзя отменить, в то время как DELETE можно откатить оператором ROLLBACK. Синтаксис оператора TRUNCATE очень прост.

```
TRUNCATE TABLE tablename;
```

Например, в базе данных корпорации Скотта один из разработчиков невнимательно заполнил таблицу EMP_HOURLY 50000 записей из неправильной таблицы. Разработчик понял, что использование оператора DELETE решит эту проблему, однако администратор заинтересован в пространстве, которое будет восстановлено. Поэтому использование TRUNCATE является лучшим решением.

```
TRUNCATE TABLE EMP_HOURLY;
```

Типы данных в ORACLE

Типы данных в ORACLE делятся на:

- простые
- комплексные
- объектные

Помимо этого в ORACLE возможно создавать собственные типы данных.

Тип данных	Описание типа
VARCHAR2(size [BYTE | CHAR])	Строка переменной длины с максимальным размером bsize байтов или символов. Максимальный размер составляет4000 байтов или символов, минимальный– 1 байт или символ. Для этого типа указание размера обязательно. СловоBYTE указывает, что размер указан в байтах, аCHAR – в символах.
NVARCHAR2(size)	Строка переменной блины поддерживающаяUnicode размером bsize символов. Количество байтов для хранения одного символа может возрастать в два раза для кодировкиAL16UTF16, и в3 раза дляUTF8. Максимальный размер составляет4000 байтов.
NUMBER [(p [, s])]	Число имеетprecision p (точность) iscale s (масштаб) Размер точности колеблется от1 до38. Масштаб может колебаться от-84 до127. Оба числа являются десятичными числами. ТипNUMBER имеет размер от 1 до22 байт.
FLOAT [(p)]	Подтип типаNUMBER имеющий толькоprecision p (точность) ТипFLOAT внутри базы данных представляется какNUMBER. Точность может колебаться от1 до123 цифр. ТипFLOAT имеет размер от1 до22 байт.
LONG	Символьные данные переменной длины размером до2 ГБ, или2

	-1 байт. Обеспечивает обратную совместимость.
DATE	Дата, диапазон которой начинается от1 января4712 года до н.э. и заканчивается31 декабрем9999 года н.э. Формат даты по умолчанию задается параметром NLS_DATE_FORMAT или неявно параметром NLS_TERRITORY. Имеет фиксированный размер7 байтов. ТипDATE содержит как дату, так и время, т.е. поля: год, месяц, день, час, минуты и секунды. Не имеет дробных значений для временных зон
BINARY_FLOAT	32-битное число с плавающей точкой. Занимает4 байта.
BINARY_DOUBLE	64-битное число с плавающей точкой. Занимает8 байтов.
TIMESTAMP [(fractional_seconds_precision)]	Тип данных, содержащий как год, месяц и день в качестве значения даты, так и час, минуты и секунды в качестве значения времени, где число fractional_seconds_precision указывает сколько долей секунды в формате времени. fractional_seconds_precision может принимать значение от0 до9, по умолчанию имеет значение6. Формат времени и даты по умолчанию зависит от значения параметраNLS_TIMESTAMP_FORMAT или неявно от параметраNLS_TERRITORY. Имеет размер от7 до 11 байт, в зависимости от размера fractional_seconds_precision. ТипDATE содержит как дату, так и время, т.е. поля: год, месяц, день, час, минуты и секунды. Не имеет временных зон
TIMESTAMP [(fractional_ seconds)] WITH TIME ZONE	Данный тип такой же, как иTIMESTAMP, значение даты задается с временными зонами. Размер фиксированный– 13 байт.
TIMESTAMP [(fractional_ seconds)] WITH LOCAL TIME ZONE	Расширенный форматTIMESTAMP, имеющий следующие исключения: ■ Формат даты нормализован в соответствии с значением временных зон сервера. ■ Когда значение даты получено из базы данных, пользователи видят дату во время се сессии в соответствии с их временной зоной. Формат времени и даты по умолчанию зависит от значения параметраNLS_TIMESTAMP_FORMAT или неявно от параметраNLS_TERRITORY. Размер от7 до11 байт.
NTERVAL YEAR [(year_ precision)] TO MONTH	Хранит период времени в годах, месяцах, гдеyear_ precision – количество цифр содержащихся в полеdate. Принимает значения от0 до9, по умолчанию2. Размер фиксированный– 5 байт
INTERVAL DAY [(day_precision)] TO SECOND [(fractional_ seconds)]	Хранит период времени в днях, часах, минутах и секундах, где: ■day_precision – максимальное количество цифр в полеday типаdatetime. Принимает значения от0 до9. ■fractional_seconds_precision – число, состоящее из2 цифр, показывающее количество долей секунд поля second. Принимает значения от0 до9, по умолчанию 6. Размер фиксированный– 11 байт.
RAW(size)	Необработанные бинарные данные длинойsize байт. Максимальный размер составляет2000 байт. Параметрsize должен быть указан явням образом.
LONG RAW	Необработанные бинарные данные переменной длины размером до2 Гбайт.
ROWID	64-битная строка, представляющая уникальный адрес

	строки в ее таблице. Этот тип данных в основном предназначен для значений, возвращаемых псевдо столбцомROWID.
UROWID [(size)]	64-битная строка, представляющая уникальный адрес строки в ее индексированной таблице. Параметрsize указывает размер столбца типаUROWID. Максимальная длина4000 байт.
CHAR [(size [BYTE | CHAR])]	Символьные данные фиксированной длины bsize байт или символов. Максимальный размер2000 байт или символов. Минимальный размер и размер по умолчанию составляют1 байт. BYTE иCHAR имеют такое же предназначение, как и в типе данных VARCHAR2.
NCHAR[(size)]	Символьные данные фиксированной длины bsize символов. Количество байт может быть в два раза больше для кодировкиAL16UTF16 и в три раза– для UTF8. Максимальная длина определяется национальной кодировкой, с максимальным размером в2000 байт. Минимальный размер и размер по умолчанию составляют1 байт.
CLOB	Большие объекты, содержащие одно или многобайтные символы. Имеет как фиксированную, так и переменную длину. Максимальный размер (4Гбайт-1)*(Размер блока базы данных).
NCLOB	Большие объекты символов, содержащие символы Unicode. Имеет как фиксированную, так и переменную длину. Максимальный размер(4Гбайт-1)*(Размер блока базы данных). Хранит данные в национальной кодировке.
BLOB	Большие объекты бинарных данных. Максимальный размер(4Гбайт-1)*(Размер блока базы данных).
BFILE	Содержит ссылку на большой бинарный файл, хранящийся вне базы данных. Подключает внешний потоковый доступ для внешнего постоянного хранения файлов на сервере базы данных. Максимальный размер4 Гбайт

Вопросы к лекции

1. Что значит аббревиатура *DDL?*
2. Как создается таблица с помощью оператора CREATE?
3. Какие типы данных в ORACLE вы знаете?
4. Какие типы данных входят в объектные типы?

Список используемой литературы

Кевин Луни, Марлен Терьо «ORACLE 9i. Настольная книга администратора» изд. «ORACLE Press» перев. «Лори» 2004г. 750 стр.

Ian Abramson, Michael Abbey, Michael J. Corey, Michelle Malcher «ORACLE Database 1 lg. A Beginner's Guide» изд. «McGrowHill» 2008г. 415 стр.

Том Кайт «ORACLE для профессионалов. Книга 1. Архитектура и основные особенности.» изд. «APress» перев. «ДиаСофтЮП» 2003г 672 стр.

Jason Price «Oracle Database 1 lg SQL» изд. «McGrowHill» 2007г. 656 стр.

Bob Bryla «ORACLE Database Foundations lOg» изд. «SYBEX» 2004г. 349 стр.

Лекция 4 **Объекты LOB. Представления. Таблицы. Специальные предикаты SQL(IN, BEETWEEN , LIKE и др.)**

План

1. Представления в ORACLE
2. Таблицы в ORACLE
3. Специальные предикаты SQL

Представления в ORACLE

Представление - это таблица, содержащая столбцы, и обращение к нему осуществляется точно так же, как и к таблице. Однако оно не содержит данных. Концептуально представление можно считать маской, перекрывающей одну или несколько таблиц, так как столбцы представления содержатся в одной или нескольких таблицах. Но физически представления не содержат данных. Определение представления (включающее запрос, на котором оно основано, расположение его столбцов и назначенные привилегии) содержится в словаре данных.

При обращении к представлению оно обращается к таблицам, на которых основано, и возвращает значения в формате и порядке, указанном в его определении. Поскольку с представлением не связано непосредственно никаких физических данных, оно не может быть проиндексировано.

Представления часто используются для обеспечения безопасности данных уровня строки или столбца. Например, можно предоставить пользователю доступ только **к** такому представлению, которое показывает из таблицы лишь строки этого пользователя, не открывая при этом доступа ко всем строкам таблицы. Таким же способом можно ограничить видимые пользователем столбцы.

Идея представления (view) проста: определить запрос, который предполагается часто использовать, сохранить его в базе данных Oracle и разрешить пользователям обращаться к нему по имени, как к обычной таблице. Когда пользователь выбирает данные из представления, Oracle выполняет соответствующий запрос, организует результаты так, как определено в представлении, и выдает их пользователю. Для пользователя представление выглядит как таблица, из которой поступают данные. Однако на самом деле данные поступают через представление, из одного или нескольких других источников.

Зачем нужны представления? По целому ряду причин. В частности, представления широко применяются для соединения данных из двух и более таблиц и выдачи их пользователям в виде одного легко читаемого списка.

Представления часто используют:

- Чтобы поддерживать безопасность, поскольку ни позволяют ограничивать диапазон строк и столбцов, возвращаемых пользователям. Если вы не хотите, чтобы пользователи видели столбец с зарплатой из таблицы личных данных, просто не включайте его в определение представления. Для пользователей представления этот столбец не будет существовать. То же самое справедливо и для строк: включите в представление конструкцию WHERE, и возвращаемые записи будут отфильтрованы любым нужным вам образом.
- Чтобы скрывать сложность данных. Например, представление может быть определено соединением - количеством столбцов или строк в различных таблицах. Однако представление скрывает тот факт, что информация на самом деле происходит из нескольких таблиц.
- Для упрощения использования для пользователей. Например, представления позволяют пользователям выбирать информацию из нескольких таблиц без

необходимости знания как производить соединение.

- Для представления данных в проекции, не так как они расположены в основной базе данных. Например, столбцы или представление могут быть переименованы, не оказывая влияния на таблицы, на основе которых создано представление.
- Для изолирования приложения от возможных изменений в базовых таблицах. Например, представление определено запросом выбирающим три столбца из четырех столбцов таблицы, и в таблицу добавляется пятый столбец, на представление это не оказывает влияния и все приложения, работающие с представлением, также не меняются.
- Для представления запросов, которые просто не могут быть сделаны без использования представлений. Например, представления которые соединяют представление основанное на предикате GROUPBY с таблицей, или представление которое объединяет с помощью оператора UNION представление и таблицу.
- Позволяет хранить сложнее запросы. Например, запрос может производить пространственные вычисления с информацией из таблиц. Сохраняя такой запрос в качестве представления, вы производите вычисления каждый раз при запросе к представлению.

ORACLE хранит представления в словаре определения данных в виде текста. Когда ссылаетесь на представление в SQL выражении, ORACLE:

1. Объединяет выражение, которое ссылается на представление, с запросом определенным в представлении.
2. Разбирает объединенное выражение в общей области SQL.
3. Выполняет выражение.

Создание и удаление представлений.

Метод создания представления - это сама простота. Нужно указать только имя представления и оператор SELECT, который будет выполняться при обращении к представлению. Вотсоответствующийсинтаксис:

CREATE [OR REPLACE] VIEW view_name AS оператор SELECT;

Обратите внимание, что здесь присутствует новый элемент: ORREPLACE. Он позволяет создавать новое представление даже тогда, когда представление с указанным именем уже существует. (Разумеется, существующее представление при этом перезаписывается.) Удалить представление так же легко, как и таблицу (но это действие менее разрушительно, поскольку представление не содержит никаких данных; самое худшее, к чему может привести случайное удаление представления - это к необходимости создавать его заново). Команда, удаляющая представление, имеет следующий синтаксис:

```
DROP VIEW view_name;
```

Таблицы в ORACLE

Таблицы представляют собой механизм сохранения информации в базе данных ORACLE. Они содержат фиксированный набор столбцов, в которых описываются атрибуты объекта, с которым эта таблица работает. У каждого столбца есть имя и уникальные характеристики.

Столбец характеризуется типом данных и длиной. Для столбцов типа NUMBER можно задать дополнительные характеристики точности и масштаба. Точность определяет число значащих цифр. Масштаб показывает место десятичной точки. Определение NUMBER (9.2) показывает, что в поле столбца хранится всего 9 цифр, две из которых располагаются справа от десятичной точки. Точность по умолчанию - 38 цифр; кроме того, это максимально возможная точность.

Таблицы, принадлежащие SYS, называются таблицами словаря данных. Они содержат системный каталог, с помощью которого система управляет свой работой. Словарь данных создается набором сценариев каталогов, предоставляемых системой ORACLE. При каждой установке или модернизации базы данных необходимо запускать сценарии, создающие или модифицирующие таблицы словаря данных. При установке новой возможности в базу данных может потребоваться запуск дополнительных сценариев каталога.

Таблицы связываются друг с другом через общие столбцы. База данных реализует эти отношения с помощью так называемой ссылочной целостности. При использовании объектно-ориентированных возможностей ORACLE строки можно связывать друг с другом при помощи внутренних ссылок, называемых идентификаторами объекте На уровне базы данных ссылочная целостность реализуется путем использования ограничений.

Временные таблицы.

Подобно регулярной таблице временная таблица является механизмом хранения данных в базе данных ORACLE. Временная таблица состоит из столбцов, имеющих типы данных и длину. В отличие от регулярной таблицы описание временной таблицы сохраняется, но данные, внесенные в таблицу, остаются в ней во время сеанса или во время транзакции. Создание временной таблицы в качестве глобальной временной таблицы обеспечивает для всех сеансов, поддерживающих соединение с базой данных, возможность видеть данную таблицу и пользоваться ею. Во время коллективных сеансов во временные таблицы можно вставлять строки данных. Однако каждая строка данных в таблице видна только для того сеанса, который вставил эту строку.Данные, содержащиеся во временной таблице, относятся либо к сеансу, либо к транзакции, в зависимости от ключевых слов, которые используются в конструкции on commit по отношению к данной временной таблице. Можно указать on commit delete rows или on commit preserve rows.

Временные таблицы доступны начиная с ORACLE8i, и обеспечивают средство помещения результатов в буфер или набор результатов, когда во время транзакции или сеанса приложение должно выполнить несколько операторов DML. Ограничения

Вложенные таблицы и изменяемые массивы.

Вложенная таблица - это столбец (столбцы) таблицы, содержащий несколько значений в одной строке. Например, если человек имеет несколько адресов, в таблице можно создать строку, содержащую несколько значений для столбца Address и только по одному значению для остальных столбцов. Вложенные таблицы могут содержать несколько столбцов и неограниченное число строк. Другим типом сборных конструкций является изменяемый массив, в котором число строк ограничено.

Использование как вложенных таблиц, так и изменяемых массивов требует модификации синтаксиса SQL, применяемого для доступа к данным. В целом отношения между данными вложенных таблиц можно симулировать с помощью реляционных таблиц.

SQL операторы позволяют ограничивать выбираемые строки, которые соответствуют шаблону, списки значений, соответствующие определенному диапазону или пустые значения. Операторы представлены в таблице 4.1

Оператор	**Описание**
LIKE	Проверяет соответствие строки заданному шаблону
IN	Проверяет значение на присутствие в списке
BETWEEN	Проверяет значение на вхождение в диапазон
IS NULL	Проверяет, является ли значение пустым

IS NAN	Проверяет, является ли значение не числовым значением
IS INFINITE	Проверяет, является ли значение бесконечным BINARY_FLOAT или BINARY_DOUBLE

Таблица 4.1 Операторы SQL

Можно также использовать NOT для того чтобы изменить значение оператора на обратное, например:

NOT LIKE
NOT IN
NOT BETWEEN
IS NOT NULL
IS NOT NAN
IS NOT INFINITE

Оператор LIKE может быть использован в выражении с WHERE для поиска строк, соответствующих шаблону. Шаблон задавать можно, используя комбинацию символов алфавита и двух специальных символов:

- (_) Нижнее подчеркивание указывает на символ в указанной позиции
- (%) Процент указывает на любое количество символов в позиции.

Например, рассмотрим следующий шаблон: ' _o%'

Нижнее подчеркивание (_) соответствует любому символу в первой позиции, (o) соответствует букве «o» во второй позиции, и процент (%) соответствует любым символам после «o». Следующий запрос использует оператор LIKE для поиска в таблице CUSTOMERS строк в столбце FIRST_NAME, соответствующих шаблону ' _o% ':

```
SELECT *
FROM customers
WHERE first name LIKE ' o%';
```

CUSTOMER ID	FIRST_NAME	LASTNAME	**DOB**	**PHONE**
1	John	Brown	01-JAN-65	800-555-1211
5	Doreen	Blue	20-MAY-70	

Таблица 4.2 Результат выполнения запроса с LIKE.

Оператор IN используется в выражениях с WHERE для получения строк, значения которых встречаются в списке хотя бы раз. Следующий запрос выбирает строки из таблицы CUSTOMERS, где значения столбца CUSTOMER_ID равны 2, 3 или 5:

```
SELECT * FROM customers WHERE customer id IN (2, 3, 5);
```

CUSTOMER ID	**FIRST** NAME	LAST NAME	**DOB**	**PHONE**
2	Cynthia	Green	05-FEB-68	800-555-1212
3	Steve	White	16-MAR-71	800-555-1213
5	Doreen	Blue	20-MAY-70	

Таблица 4.3 Результат выполнения оператора IN.

Оператор BETWEEN используется в выражении WHERE для получения строк, значения которых лежат в заданном диапазоне. Диапазон включает начальное и конечное значение. Следующий запрос использует оператор BETWEEN для получения строк из таблицы CUSTOMERS, у которых значение столбца CUSTOMER_ID находится в диапазоне от 1 до 3.

```
SELECT * FROM customers WHERE customer id BETWEEN 1 AND 3;
```

CUSTOMER ID	FIRSTNAME	LASTNAME	**DOB**	**PHONE**
1	John	Brown	01-JAN-65	800-555-1211
2	Cynthia	Green	05-FEB-68	800-555-1212

3	Steve	White	16-MAR-71	800-555-1213

Таблица 4.4 Результат выполнения оператора BETWEEN.

Вопросы к лекции

1. Что такое представления в ORACLE?
2. Для чего они используются ?
3. Каким образом организуются таблицы в ORACLE?
4. Каково назначение специальных предикатов *SQL*.

Список используемой литературы

Кевин Луни, Марлен Терьо«ORACLE 9i. Настольная книга администратора» изд. «ORACLEPress» перев. «Лори» 2004г.750 стр.

Ian Abramson, Michael Abbey, Michael J. Corey, Michelle Malcher«ORACLE Database 1 lg. A Beginner's Guide» изд.«McGrowHill» 2008r.415 стр.

Лекция 5 **Теоретико-множественные операции: соединение, объединение, вычитание, декартовое произведение.**

План

1. Теоретико-множественные операции ORACLE
2. Соединения в ORACLE
3. Декартовое произведение в ORACLE

Теоретико-множественные операции.

В ORACLE как и любой другой реляционной СУБД поддерживаются следующие виды теоретико-множественных операций:

- соединение;
- декартовое произведение;
- объединение;
- вычитание;
- пересечение.

Соединения в ORACLE.

Схемы базы данных обычно содержат более чем одну таблицу, которые хранят различные данные. Например, схема «Store» имеет таблицы, которые содержат информацию о клиентах, продуктах, работниках и др. Все запросы, показанные в предыдущих лекциях, запрашивали и выводили информацию только из одной таблицы. В реальности же необходимо запрашивать и получать информацию из нескольких таблиц.

Примеры соединений будем рассматривать на таблицах «Продукты» и «Типы продуктов».

PRODUCT_TYPE_ID	NAME
1	Book
2	Video
3	DVD
4	CD
5	Magazine

Таблица 5.1 «Типы продуктов» (product_types).

PRODUCT_ID	PRODUCT_TYPE_ID	NAME	DESCRIPTION	PRICE
1	1	Modern	A Science description of modern science	19.95
2	1	Chemistry	Introduction to Chemistry	30
3	2	Supernova	A star explodes	25.99
4	2	Tank War	Action movie about a future war	13.95
5	2	Z Files		49.99
6	2	2412: The Return		14.95
7	3	Space Force 9		13.49
8	3	From Another Planet		12.49
9	4	Classical Music		10.99
10	4	Pop3		15.99
11	4	Creative Yell		14.99
12	null	My Front Line		13.49

Таблица 5.2 «Продукты» (products).

Например, необходимо получить название продукта и название типа продукта под номером 3. Название продукта хранится в таблице «Продукты» PRODUCTS в столбце NAME (таб. 5.1), а название типа продукта в столбце NAME таблицы «Типы продуктов» PRODUCT_TYPES (таб. 5.2). Таблицы PRODUCTS и PRODUCT_TYPES связаны друг с другом через вторичный ключ PRODUCT_TYPE_ID; столбец PRODUCT_TYPE_ID (вторичный ключ) таблицы PRODUCTS указывает на столбец PRODUCT_TYPE_ID (вторичный ключ) таблицы PRODUCT_TYPES (первичный ключ). Следующий запрос выводит NAME и PRODUCT_TYPE_ID из таблицы PRODUCTS с номером 3:

```
SELECT name,  product_type_id
FROM products
WHERE product id =  3;
```

NAME	PRODUCT_TYPE_ID
Supernova	2

Таблица 5.3 Вывод названия продукта и типа продукта под номером 3.

Следующий запрос выводит столбец NAME из таблицы PRODUCT_TYPES с номером 2:

```
SELECT name FROM product_types WHERE product_type_id= 2;
```

NAME
Video

Рис. 5.4 Вывод названия типа продукта под номером 2.

Из этого запросы вы узнаете, что продукт под номером 2 - Video. В принципе ничего сложного, но нам действительно хотелось бы получить название продукта и тип продукта, используя один запрос. Можно сделать это, реализуя соединение двух таблиц.

Чтобы выполнить соединение необходимо включить обе таблицы в запрос после ключевого слова FROM и включить оба столбца из обеих таблиц в выражение WHERE. Запрос будет выглядеть следующим образом:

```
SELECT products.name, product_types.name
FROM products, product_types
WHERE  products.product_type_id =
product_types.product_type_id
    AND products.product_id =   3;
```

Соединение это первое условие в выражении WHERE (products .product_type_id = product_types .product_type_id). Обычно столбцы, использующиеся в условии соединения, являются первичным ключом в одной таблице и вторичным - в другой. Так как оператор (=) используется в условии соединения, такое соединение известно как ***эквивалентное соединение.*** Второе условие в выражение WHERE выбирает продукты с номером 3, который нас интересовал.

NAME	NAME
Supernova	Video

Таблица 5.5 Результат соединения.

Виды соединений.

Существуют следующие виды соединений:

- Внутренние соединения;
- Внешние соединения;

В зависимости от условия, которые используются при соединении, различают:

- Эквивалентные соединения;
- Неэквивалентные соединения.
- Декартовое произведение.

Отдельно можно выделить «Самосоединения».

Внутреннее соединение.

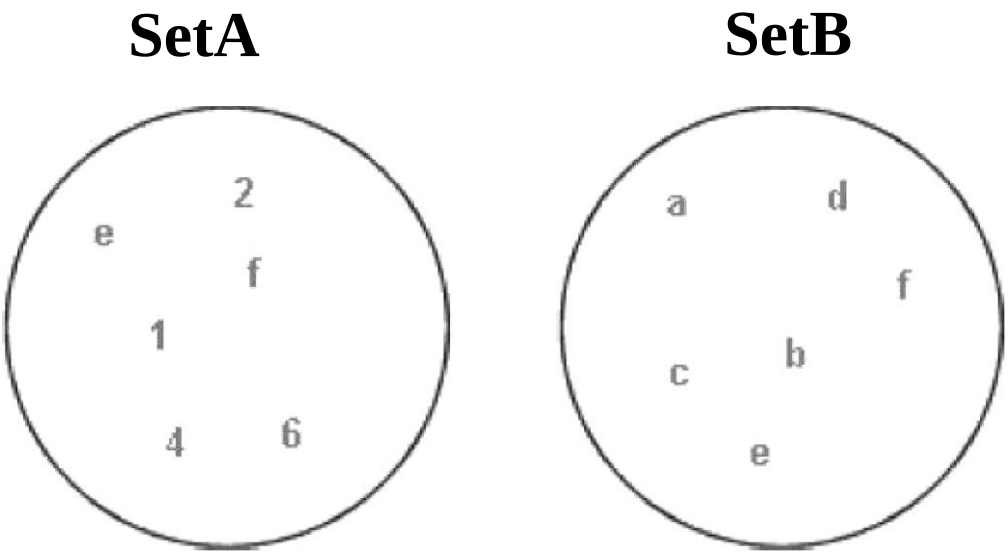


Рис. 5.1 Два множества.

На рис. 5.1. Представлены два независимых множества данных A и B. Оба множества содержат одинаковые элементы. До тех пор, пока мы не свяжем оба эти множества, они будут несвязанными.
На рис. 4.2 Мы видим эти же множества, только показано пересечение этих двух множеств. Пересечение этих множеств будет эквивалентно внутреннему соединению.

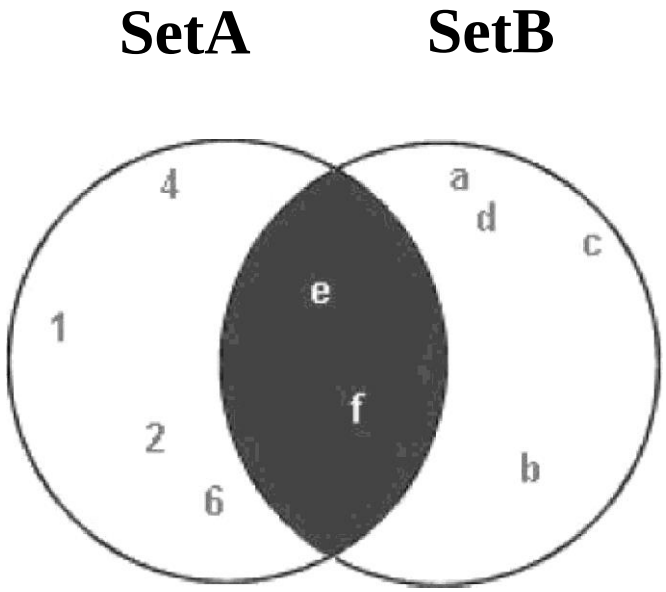


Рис. 5.2 Внутреннее соединение.

Эквивалентные соединения также известны как простые соединения или внутренние. Для выбранных двух или таблиц, внутреннее соединение возвращает результат, в котором общий столбец между любыми двумя парами таблиц имеет одинаковое значение (равное значение). Эквивалентное соединение обычно осуществляется между вторичным ключом в одной таблице и первичным ключом в другой таблице.

Внутреннее соединение можно выполнять в соответствии со стандартом ANSI SQL/86 следующим образом:

```
SELECT columns FROM table_l, table_2 WHERE
table_l.column_l = table2.column_2
```

Будут возвращены строки со столбцами columns из таблиц table_l и table_2, в которых значение столбца table_l. column_l равно значению table_2 . column_2.

Например следующий запрос выводит продукты и названия их типов, сортируя по названию продукта.

```
SELECT products.name, product_types.name FROM
products, product_types WHERE
products.product_type_id =
product_types.product_type_id ORDER BY
products.name;
```

NAME	NAME
2412: The Return	Video
Chemistry	Book
Classical Music	CD
Creative Yell	CD
From Another Planet	DVD
Modern Science	Book
Pop3	CD
Space Force 9	DVD
Supernova	Video
Tank War	Video
Z Files	Video
2412: The Return	Video

Таблица 5.6 Результат соединения таблиц products и productjtypes.

Продукт с названием «My Front Line» не был выведен. Значение producttypeid для этого продукты равно null, и условие соединения не выводит строку.

Можно также производить соединения согласно стандарту ANSI SQL/92:

```
SELECT columns FROM table_l [INNER] JOIN table_2
ON table_l.column_l = table2.column_2
```

Тогда наш пример будет выглядеть следующим образом:

```
SELECT products.name, product_types.name
FROM products INNER JOIN product_types ON
products.product_type_id = product_types.product_type_id
ORDER BY products.name;
```

Натуральные соединения.

Если в обеих таблицах соединение нужно произвести по столбцам, имеющим одинаковое имя в обеих таблицах можно воспользоваться так называемым натуральным соединением NAT URAL JO IN:

```
SELECT columns FROM table_l NATURAL JOIN table_2
ORDER BY products.name;
```

Например, осуществим предыдущие запросы с помощью натурального соединения:

```
SELECT products.name, product_types.name
FROM products NATURAL JOIN product_types;
```

Натуральные соединения можно производить в соответствии со стандартом ANSI SQL/92 с помощью ключевого слова USING. Необходимо соблюдать следующие ограничения при натуральных соединениях:

- соединение должно быть эквивалентным;
- имена столбцов в обеих таблицах должным быть одинаковыми.

```
SELECT columns
FROM table_1 INNER JOIN table_2 USING (column_name);
```

Где column_name - имя столбцов в обеих таблицах.

В соответствии с ANSI SQL/92 наш запрос будет выглядеть следующим образом:

```
SELECT products.name, product_types.name FROM
products INNER JOIN product_types USING
(product_type_id);
    ORDER BY products.name;
```

Внешние соединения.

Иногда необходимо связать две таблицы и возвратить все строки одной таблицы, не зависимо от того, содержат ли строки во второй таблице строки, удовлетворяющие условиям. Это также известно как произвести внешнее соединение между двумя таблицами. Внешнее соединение выбирает строку, даже если один из столбцов имеет значение null. Для того, чтобы произвести внешнее соединение в соответствии со стандартом ANSI SQL/86, используется заключение знака плюс в круглые скобки (+).

Внешние соединения бывают трех типов

- левое внешнее соединение
- правое внешнее соединение
- полное внешнее соединение.

Чтобы определить разницу между ними обратимся к рисункам 4.3, 4.4 и 4.5 соответственно.

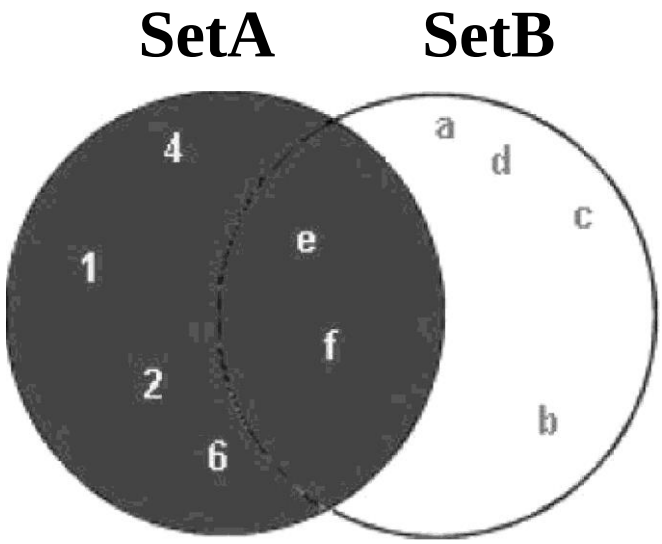


Рис. 5.3 Левое внешнее соединение.

При левом соединении выведены все значения множества A, которые содержатся во множестве B (общая середина), а также элементы множества A, которых нет в B (рис 4.3).

Такое соединения выполняется с помощью следующего синтаксиса ANSI SQL/92:

```
SELECT columns FROM table_1, table_2
WHERE table_1.column_1 = table2.column_2 (+);
```

Например, нам необходимо название всех продуктов и их типов, независимо от того имеются ли типы у них или нет.

```
SELECT products.name, product_types.name FROM
products, product_types WHERE
products.product_type_id =
product_types.product_type_id (+) ORDER BY
products.name;
```

В результате получим следующий вывод (таблица 5.7):

NAME	NAME
2412: The Return	Video
Chemistry	Book
Classical Music	CD
Creative Yell	CD
From Another Planet	DVD
My Front Line	null
Modern Science	Book
Pop3	CD
Space Force 9	DVD
Supernova	Video
Tank War	Video
Z Files	Video
2412: The Return	Video

Таблица 5.7 Результат левого соединения таблиц products и product_types.

Для записи соединения с помощью стандарта ANSI SQL/92 используется следующий вид запроса:

```
SELECT columns FROM table_1 LEFT [OUTER] JOIN table_2
ON table_1.column_1 = table2.column_2
```

Тогда наш пример будет выглядеть следующим образом:

```
SELECT products.name, product_types.name
FROM products LEFT JOIN product_types ON
products.product_type_id = product_types.product_type_id
ORDER BY products.name;
```

Правое внешнее соединение.

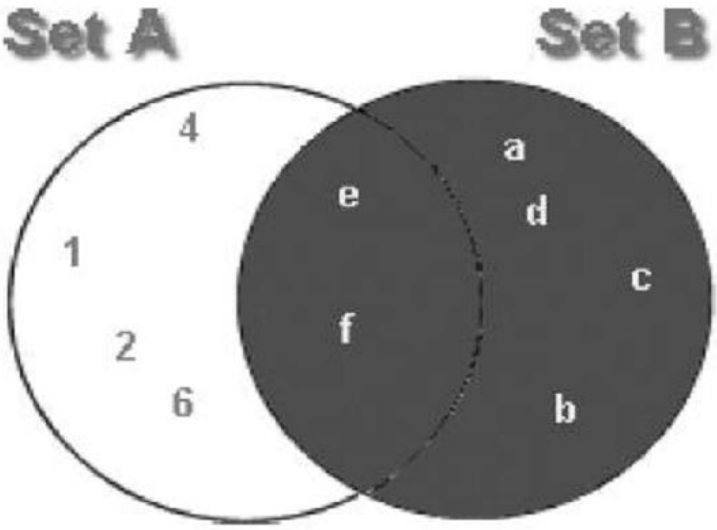
При правом соединении в отличие от левого выбираются все элементы, имеющиеся во множестве B, а также все общие для множеств A и B элементы (рис. 4.4).

Такое соединения выполняется с помощью следующего синтаксиса ANSI SQL/92:

```
SELECT columns FROM table_1, table_2
WHERE table_1.column_1 (+) = table2.column_2;
```

Например, нам необходимо название продуктов и всех типов, независимо от того имеются продукты такого типа или нет.

```
SELECT products.name, product_types.name FROM
products, product_types WHERE
products.product_type_id (+) =
product_types.product_type_id ORDER BY
products.name;
```



В результате получим следующий вывод (таблица 5.8)

Рис 5.4 Правое внешнее соединение.

NAME	NAME
2412: The Return	Video
Chemistry	Book
Classical Music	CD
Creative Yell	CD
From Another Planet	DVD
Modern Science	Book
Pop3	CD
Space Force 9	DVD
Supernova	Video
Tank War	Video
Z Files	Video
2412: The Return	Video
null	Magazine

Таблица 5.8 Результат правого соединения таблиц products и producttypes.

Для записи соединения с помощью стандарта ANSI SQL/92 используется следующий вид запроса:

```
SELECT columns FROM table_1 RIGHT [OUTER] JOIN table_2 ON
table_1.column_1 = table2.column_2
```

Тогда наш пример будет выглядеть следующим образом:

```
SELECT products.name, product_types.name
FROM products RIGHT JOIN product_types ON
products.product_type_id = product_types.product_type_id
ORDER BY products.name;
```

Полное внешнее соединение.

SetA **SetB**

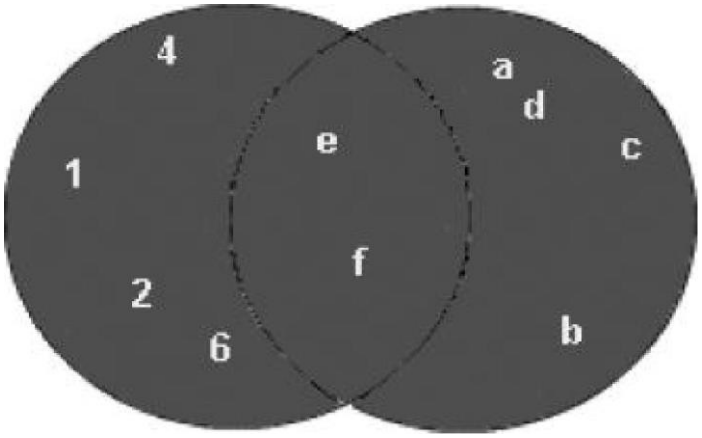


Рис 5.5Полное внешнее соединение.

Как уже стало понятно полное внешнее соединение - это все данные множеств A и B, независимо от того есть или нет у них пары во втором множестве (рис. 4.5).

Не возможно выполнить полное соединение в соответствии со стандартом ANSI SQL/86, поскольку не разрешается использование оператора (+) с обеих сторон от условия WHERE:

```
WHERE table_1.column_1 (+) = table2.column_2 (+);
```

Для того чтобы осуществить полное соединение в соответствии со стандартом ANSI SQL/92 необходимо соблюдать следующий синтаксис:

```
SELECT columns FROM table_1 FULL [OUTER] JOIN table_2
ON table_1.column_1 = table2.column_2
```

Тогда наш пример будет выглядеть следующим образом:

```
SELECT products.name, product_types.name
FROM products FULL JOIN product_types ON
products.product_type_id = product_types.product_type_id
ORDER BY products.name;
```

NAME	NAME
2412: The Return	Video
Chemistry	Book
Classical Music	CD
Creative Yell	CD
From Another Planet	DVD
My Front Line	null
Modern Science	Book
Pop3	CD
Space Force 9	DVD
Supernova	Video
Tank War	Video
Z Files	Video
2412: The Return	Video
null	Magazine

Таблица 5.9 Результат полного внешнего соединения таблиц products и productjtypes.

Использование (+) при внешних соединениях в стандарте ANSI SQL/92 имеет еще одно ограничение: нельзя использовать в выражение WHERE дополнительных условий отбора. Таким образом, следующий запрос будет выдавать ошибку:

```
SELECT p.name, pt.name
FROM products p, product_types pt
WHERE p.product_type_id (+) = pt.product_type_id
 OR p.product_type_id = 1;
```

Декартовое произведение.

Декартовым произведением называется соединение, при котором каждому элементу одного множества сопоставляются абсолютно все значения другого множества.

Декартовое произведение получается, если при соединении двух или более таблиц не указывается никакое условие соединения. Чтобы произвести декартовое произведение необходимо соблюдать следующий синтаксис при построении запроса по стандарту ANSI SQL/86:

```
SELECT columns FROM table_l, table_2;
```

Или в соответствии со стандартом ANSI SQL/92:

SELECT columns FROM table_l CROSS JOIN table_2;

Неэквивалентные соединения.

Неэквивалентные соединения используют оператор отличный от равенства (=). Такими операторами могут быть: не равно (о), меньше (<), больше (>), меньше или равно (<=), больше или равно (>=), LIKE, IN, BETWEEN. Ситуаций, при которых используются не эквивалентные соединения очень редки.

Рассмотрим следующий пример: Имеются различные группы зарплат (таблица 5.10), и у сотрудников определены их зарплаты (таблица 5.11), нам требуется получить, к какой группе зарплат относятся сотрудники предприятия.

SALARY GRADE П)	LOW SALARY	HIGH SALARY
1	1	250000
2	250001	500000
3	500001	750000
4	750001	999999

Таблица 5.10 Группы зарплат.

EMPLOYEE_ID	MANAGER_ID	FIRST_NAME	LAST_NAME	TITLE	SALARY
1	null	James	Smith	CEO	800000
2	1	Ron	Johnson	SalesManager	600000
3	2	Fred	Hobbs	Salesperson	150000
4	2	Susan	Jones	Salesperson	500000

Таблица 5.11 сотрудники.

Следующий запрос выбирает сотрудников и группы их зарплат используя оператор BETWEEN:

```
SELECT e.first_name, e.last_name, e.title, e.salary, sg.salary_grade_id
FROM employees e, salary_grades sg
```

```
WHERE  e.salary BETWEEN  sg.low_salary AND  sg.high_salary
ORDER BY  salary grade   id;
```

FIRST NAME	LAST NAME	TITLE	SALARY	SALARY_GRADE_ID
Fred	Hobbs	Salesperson	150000	1
Susan	Jones	Salesperson	500000	2
Ron	Johnson	Sales_Manager	600000	3
James	Smith	CEO	800000	4

Таблица 5.12 Сотрудники с группами зарплат.

Вопросы к лекции

1. Какие виды теоретико-множественных операций предусмотрены в ORACLE?
2. Каково назначение операции соединения?
3. Каково назначение операции декартового произведения?
4. Каково назначение операции пересечения?

Список используемой литературы

Кевин Луни, Марлен Терьо «ORACLE 9i. Настольная книга администратора» изд. «ORACLE Press» перев. «Лори» 2004г. 750 стр.
Ian Abramson, Michael Abbey, Michael J. Corey, Michelle Malcher «ORACLE Database 1 lg. A Beginner's Guide» изд. «McGrowHill» 2008r. 415 стр.

Лекция 6 **Подзапросы. Группирование и агрегатные функции. Последовательности. Синонимы и их создание.**

План

1. Группирование и агрегатные функции
2. Подзапросы.
3. Последовательности синонимы и их создание.

Группирование и агрегатные функции

Агрегатные функции

В Oracle есть групповые функции (group functions), позволяющие анализировать группы записей. Под группой записей понимается любой набор записей, имеющих что-то общее - например, относящихся к одному товару, одному отделу или одному временному интервалу.

Внимание: агрегатные функции также известны как групповые функции, потому что они работают с группами строк.

Функция	**Описание**
AVG(x)	Возвращает среднее значение среди x
COUNT(x)	Возвращает количество строк возвращаемых запросом включающих x
MAX(x)	Возвращает максимальное значений среди x

MIN(x)	Возвращает минимальное значений среди x
SUM(x)	Возвращает сумму значений x

Таблица 6.1 Некоторые групповые функции.

Функция AVG.

Функция AVG возвращает среднее значение по указанному столбцу. Поскольку для этого функция должна выполнить фактическое считывание всего столбца, нет смысла указывать «\» или какой-либо другой литерал в качестве аргумента; необходимо указать имя столбца. Например, следующий запрос выбирает среднее значение цены всех продуктов:

```
SELECT AVG(price)
FROM products;
```

AVG(PRICE)
19.7308333

Таблица 6.2 Результат выполнения функции A VG.

Можно использовать агрегатные функции с любыми правильными выражениями. Например, следующий запрос добавляет к каждой цене +2 перед тем как найти среднее значение столбца:

```
SELECT AVG(price + 2)
FROM products;
```

Можно также с помощью ключевого слова DISTINCT исключить одинаковые значения перед вычислением. Например, следующий запрос исключает одинаковые цены в столбце price перед тем как вычислить среднее значение.:

```
SELECT AVG(DISTINCT price)
FROM products;
```

AVG(PRICE)
19.7308333

Таблица 6.3 Результат выполнения функции AVG и DISTINCT.

Функция COUNT.

Функция COUNT, как нетрудно догадаться, подсчитывает записи. Возможно, вы удивитесь, насколько часто это бывает полезно. Например, чтобы определить, содержит ли таблица какие-нибудь записи, проще всего ввести такую команду:

```
SELECT COUNT(*) FROM products;
```

Указывая «*» вместо имен столбцов, вы неявно заставляете Oracle считывать всю таблицу. Это не страшно для маленьких учебных таблиц, но представляет собой серьезную проблему в системах с сотнями тысяч, а тем более с миллиардами записей. Полное сканирование таблицы намного замедляет выдачу результата и вынуждает Oracle отвлекать компьютерные ресурсы от главных задач, для выполнения которых предназначалась база данных, что тормозит работу всей компании. Гораздо лучше указать какой-нибудь один столбец, по которому функция COUNT будет считать записи:

```
SELECT COUNT(product_id)
FROM products;
```

Как правило, предпочтение отдается первому столбцу таблицы - по причинам, которые будут объяснены в следующей главе. Можно поступить еще проще, указав в качестве аргумента функции COUNT не имя столбца, а литеральное значение:

```
SELECT COUNT(1) FROM products;
```

Строго говоря, это заставляет Oracle возвращать значение «1» для каждой записи в таблице. С тем же успехом можно подставить в функцию фразу «Hi There»; какое именно литеральное значение будет использовано - неважно, поскольку само это значение функция игнорирует. Она лишь подсчитывает записи и сообщает, сколько их было найдено.

Функция COUNT имеет одно интересное свойство: если указать столбец таблицы, записи которой подсчитываются, будут учтены только записи, содержащие в этом столбце какое-либо значение. Этим можно воспользоваться для определения процента записей, имеющих null-значение в определенном столбце.

Функции MAX и MIN.

Функция **MIN** возвращает наименьшее из значений, содержащихся в указанном столбце. Как вы наверняка догадались, функция MAX возвращает наибольшее из значений указанного столбца.

Например, чтобы узнать, сколько стоит самый дорогой и дешевый товар:

```
SELECT MAX(price), MIN(price)
FROM products;
```

MAX(PRICE)	MIN(PRICE)
49.99	10.99

Таблица 6.4 Результат выполнения функции MAX и MIN.

Функция MAX применяется в целом ряде случаев. Например, в будущем перед вами может встать вопрос о сокращении длины текстового столбца существующей таблицы. Вы захотите узнать, какая часть столбца действительно используется, что потребует определения максимальной длины текста в этом столбце. Это легко сделать, указав в качестве аргумента функции MAX длину столбца, как показано во второй из следующих команд:

```
SELECT MAX(LENGTH(name)) FROM products;
```

Функции MAX и MEN можно использовать с любыми типами данных, включая строки и даты. Когда вы используете данные функции со строками, то значения столбца упорядочиваются по алфавиту, и максимальное значение находится первым, а минимальное - последним.

Функция SUM.

Функция SUM суммирует значения и возвращает итог. Чтобы увидеть, как она работает, введите следующий код:

```
SELECT SUM(price)
FROM products;
```

SUM(PRICE)
236.77

Таблица 6.5Результат выполнения функции SUM

Группирование строк.

Теперь, когда вы знаете о групповых функциях, пора научиться создавать группы. Это делается путем добавления конструкции GROUP BY в оператор SELECT.

После конструкции GROUP BY просто указывается столбец, по значениям которого будет выполняться группирование. Обычно это первый столбец в списке SELECT.

Конструкция GROUP BY может использоваться для группировки строк на группы, с общим значением. Например, следующий запрос группирует строки таблицы products на блоки с одинаковым product_type_id;

```
SELECT product_type_id
FROM products
GROUP BY product_type_id;
```

В оператор SELECT можно включать несколько различных групповых функций. Например, один и тот же оператор может выдавать суммарное, среднее, наименьшее и наибольшее значения по каждой группе, а также подсчитывать число записей в группах. Приведенный ниже код показывает, как это делается.

```
SELECT  product_type_id,   COUNT(ROWID),   AVG(price)
FROM products
GROUP  BY product_type_id
ORDER BY product  type   id;
```

PRODUCT TYPE ID	COUNT(ROWTD)	AVG(price)
1	**2**	24.975
2	**4**	26.22
3	**2**	13.24
4	**3**	13.99
	1	13.49

Таблица 6.6 Результат выполнения функции запроса.

Обратите внимание на то, что запрос вывел пять строк, в которой каждая строка обозначает один или несколько продуктов сгруппированных вместе по столбцу product_type_id. Из результатов запросы вы можете видеть, что есть последняя строка со значением COUNT (ROWID) = 1. Это произошло из-за того, что в таблице есть продукт у которого значение product_type_id равно NULL.

Внимание: если запрос содержит агрегатную функцию и запрашивает столбцы, которые не помещены в эту агрегатную функцию, то указанные столбцы должны быть включены в конструкцию GROUP BY.

Ограничение: Нельзя использовать конструкцию WHERE для фильтра строк в уже сгруппированных строках.

Конструкция HAVING

Как вы помните, конструкция WHERE позволяет фильтровать записи, возвращаемые оператором SELECT. При группировании записей конструкция WHERE работает точно так же: фильтрует отдельные записи, тем самым исключая их из вычислений, выполняемых групповыми функциями.

Однако после создания групп возникает новая задача: фильтрация самих групп на основе групповой информации. Предположим, что вашей компании не хватает складских площадей и она намерена сократить запас товаров на складе. Для поддержки этого мероприятия вам требуется составить список плохо продающихся товаров - например, тех, для которых общий объем продаж составляет менее пяти штук. Здесь и пригодится конструкция HAVING.

Она фильтрует группы на основе групповых значений. В отличие от конструкции WHERE, фильтрующей записи до группирования, конструкция HAVING фильтрует уже сформированные группы.

Порядок конструкции при использовании HAVING следующий:

```
SELECT ...
FROM ...
WHERE
GROUP BY ...
HAVING ...
ORDER BY
```

Например, если вам нужно составить список товаров имеющих цену выше средней цены товаров этого вида:

```
SELECT product_type_id, AVG(price)
FROM products
GROUP BY product_type_id
HAVING AVG(price) > 20;
```

PRODUCTTYPEID	AVG(PRICE)
1	24.975
2	26.22

Таблица 6.7 Результат выполнения GROUP BY и HAVING.

Внимание GROUP BY может использоваться без конструкции HAVING, а вот HAVING без GROUP BY использоваться не может.

Подзапросы.

Подзапросы помещают один запрос в другой. Второй запрос помещается внутрь WHERE для расширения запросов SELECT. Один или несколько значений, возвращаемых вложенным запросом, используется в основном запросе в качестве фильтра.

Типы операторов, используемых в выражении WHERE, зависит от того одно или несколько значений возвращается подзапросом. Если возвращается только одно значение вложенным запросом, используется оператор сравнения (=, !=, <, >, <=, >=) и др. Если же возвращается более одного значения, то используется такие операторы как IN, ANY, ALL, NOT.

Существуют 2 основных типа подзапросов:

• ***Нормальные подзапросы.*** Замкнутые запросы, предполагающие, что прямой связи между вызываемым подзапросом и запросом нет.

• ***Соотнесенные (корреляционные) подзапросы.*** Слово корреляция используется, чтобы описать связь между запросом и вызываемым подзапросом. Необходимо помнить об основном правиле, что корреляционный столбец из внепшего запроса должен быть внесен в подзапрос, но не наоборот. Таким образом, соотнесенный (корреляционный) подзапрос всегда зависит от основного запроса.

Подзапросы также можно разделить на вложенные и встраиваемые представления .

• *Вложенные подзапросы.* Подзапросы могут вызывать другие подзапросы и так без конца. Другими словами, подзапросы могут вкладываться в другие подзапросы.

• ***Встраиваемые представления (inline** views).* Встраиваемые представления это подзапросы, путем встраивания в предложение FROM запроса SELECT, который, кстати, тоже является подзапросом. Значения могут быть переданы из встраиваемого представления во внешний запрос или подзапрос.

Подзапросы по количеству возвращаемы строк могут разделяться на:

• *Однострочные* ***подзапросы,*** возвращающие одну строку во внешний запрос или не возвращающие ни одной строки. Есть один случай, когда однострочный подзапрос возвращает один столбец, такой подзапрос называют *скалярным.*

• *Многострочные* ***подзапросы*** возвращают несколько строк во внешний запрос.

Подзапросы можно вставлять почти везде в любом месте SQL, в любой SQL оператор, который поддерживает выражения. Далее представлены наиболее распространенные места SQL, в которых может быть помещен подзапрос:

• После ключевого слова SELECT в местах определения столбцов запросы.
• После ключевого слова WHERE как условие запроса.
• После ключевого слова FROM (вложенное представление).
• В месте определения новых значений после ключевого слова VALUES оператора INSERT.
• В операторе UPDATE следующим образом SET clause = (подзапрос).
• В качестве формального параметра процедуры или функции.

Однострочные подзапросы

Вы можете вставить подзапрос в выражение после ключевого слова WHERE внешнего запроса. Пример:

```
SELECT first_name, last_name FROM
customers WHERE customer_id =
(SELECT customer_id FROM
customers WHERE last name =
'Brown');
```

FIRST NAME	LASTNAME
John	Brown

Таблица 6.8 Результат выполнения запроса.

Этот пример получает данные first_name и last_name строки из таблицы customers, у которой значение last_name равно Brown. Давайте разобьем этот запрос и проанализируем, как он выполнился. Подзапрос внутри выражения после ключевого слова WHERE:

```
SELECT customer_id FROM customers
WHERE last_name = 'Brown';
```

Этот подзапрос выполняется сначала (и только один раз) и возвращает customer_id для строки, у которой значение last_name равно Brown. Значение customerid для этой строки равно 1, которое подставляется в выражение поле ключевого слова WHERE внешнего запроса. Таким образом, внешний запрос возвращает такой же результат, что и следующий запрос:

```
SELECT first_name, last_name  FROM customers
WHERE customer_id = 1;
```

Предыдущий запрос использовал в выражении после ключевого слова WHERE оператор эквивалентности (=). Вы также можете использовать другие операторы сравнения, такие как о, <, >, <=, >=, с однострочными подзапросами. Следующий пример использует > в выражении WHERE внешнего запроса; подзапрос использует функцию AVG () чтобы получить среднюю цену всех продуктов, которая будет помещена в выражение после ключевого слова WHERE во внепшем запросе. Весь запрос выбирает номер product_id, название name и цену price продуктов, имеющих цену больше средней цены всех продуктов:

```
SELECT product_id, name, price
FROM products WHERE price >
(SELECT AVG(price) FROM
products);
```

PRODUCT ID	NAME	PRICE
1	Modern Science	19.95

2	Chemistry	**30**
3	Supernova	25.99
5	Z Files	49.99

Таблица 6.9. Результат выполнения запроса с оператором сравнения.

Этот подзапрос является примером скалярного подзапроса, потому что возвращает ровно 1 строку, содержащую 1 столбец. Значение, возвращаемое скалярным подзапросом, обрабатывается как одиночное значение.

Как уже известно из предыдущего раздела, выражение HAVING используется для фильтрации сгруппированных строк. Можно поместить подзапрос в выражение HAVING внешнего запроса. Таким способом можно отфильтровать группы строк основываясь на результате, возвращаемом подзапросом. Следующий пример использует подзапрос в выражении HAVING:

```
 SELECT product_type_id, AVG(price)
FROM products GROUP BY
product_type_id HAVING AVG(price)
< (SELECT MAX(AVG(price)) FROM
products GROUP BY product_type_id)
ORDER BY product type id;
```

PRODUCT TYPE ID	AVG(PRICE)
1	24.975
3	13.24
4	13.99
	13.49

Таблица 6.10. Результат выполнения подзапроса в выражении HAVING.

Уведомление: подзапрос использует функцию AVG() для подсчета средней цены каждого типа продукта (таблица 4.2). Результат, возвращенный функцией AVG передается функции MAXO, которая возвращает наибольшую из средних цен.

Многострочные подзапросы

Многострочные запросы возвращают несколько строк. Операторы IN EXISTS, а также группа операторов сопоставления ANY, ALL, SOME позволяют работать с результатами многострочных подзапросов.

Возможности этих операторов приведены ниже:

• Множественный подзапрос может возвратить множество значений для сравнения оператора IN. Как вы помните оператор IN проверяет есть ли такое значение во множестве. Следующий пример выводит product_id и name из таблицы products, если product_id является одним из продуктов содержащим в названии букву «e»

```
SELECT product_id, name
FROM products WHERE
product_id IN (SELECT
product_id FROM
products WHERE name LIKE
'%e%');
```

• Сопоставление EXIST обычно использует индексы для нахождения пары, для значения в подзапросе со значением из внепшего запроса. Не обращая внимания на корреляционный индекс столбца между внешним запросом и подзапросом, EXIST останавливает выполнения подзапроса, когда хотя бы одно соответствующее значение

найдено. IN построит все значения множества, возвращаемого подзапросом, перед тем, как передать результаты во внешний запрос. Предпочтительное использование EXIST вместо IN часто выражается в лучшей производительности. EXIST может не работать производительнее, если множеством, возвращаемым подзапросом, является ограниченное множество литеральных значений или малое количество строк.

• ANY, ALL, и SOME подразумевают любое, все или некоторое количество значений соответственно. Из-за того, что подзапрос сопоставляется со множеством значений, подзапрос может возвращать как в реальности нуль, одну или множество строк.

Следующий запрос использует оператор ANY для того, чтобы получить сотрудников, у которых зарплата ниже самого низкого значения зарплаты в таблице `salary_grades`.

```
SELECT employee_id, last_name
FROM employees WHERE salary <
ANY (SELECT low_salary FROM
salary_grades);
```

EMPLOYEE_ID	LAST_NAME
2	Johnson
3	Hobbs
4	Jones

Таблица 6.11. Результат выполнения подзапроса с оператором ALL.

Внимание: Очень важно, чтобы многострочный подзапрос мог возвращать нуль строк, поскольку вхождение (IN), существование (EXIST) и соотнесение (ANY, ALL, SOME) возвращает множество значений. Пустое множество также может быть множеством значений. К тому же пустое множество является правильным множеством.Подзапросы можно помещать после ключевого слова FROM внешнего запроса. Этот тип подзапросов также известен как, встраиваемым представлением, потому что подзапрос представляет данные в качестве таблицы.

Например, получить название продуктов из таблицы продуктов, чей product_id меньше 3:

```
SELECT product_id
FROM
(SELECT * FROM
products WHERE
product_id < 3);
```

Следующий пример более является более наглядным: возвратить product_id и price из таблицы products во внешнем запросе, и подзапросом получить количество раз данный продукт был заказан:

```
SELECT prds.product_id, price, purchases_data.product_count
FROM products prds,(SELECT product_id, COUNT(product_id)
product_count
FROM purchases GROUP BY product_id) purchases_data WHERE
prds.product_id = purchases_data.product_id;
```

Соотнесенные подзапросы.

Соотнесенный подзапрос ссылается на один или несколько столбцов во внешнем запросе. Такие подзапросы еще называют корреляционными. Чаще всего соотнесенные подзапросы используются, когда необходимо ответить на вопрос, который зависит от

содержащегося значения каждой строки внешнего запроса. Следующий соотнесенный подзапрос получает информацию о продуктах которые имеют цену больше средней цены для данного типа продукта:

```
SELECT product_id, product_type_id, name, price FROM
products outer WHERE price > (SELECT AVG(price) FROM
products inner WHERE inner.product type id =
outer.product type id);
```

PRODUCT	PRODUCT TYPE	NAME	PRICE
2	1	Chemistry	30
5	2	Z Files	49.99
7	3	Space Force 9	13.49
10	4	Pop3	15.99
11	4	Creative Yell	14.99

Таблица 6.12 Результат соотнесенного подзапроса.

Обратите внимание, что в примере использованы псевдонимы outer для внешнего запроса и inner для подзапроса. В соотнесенном подзапросе каждая строка внешнего запроса имеет связь со строкой подзапроса. Подзапрос считывает каждую строку внешнего запроса и применяет ее значения для своего вычисления, до тех пор, пока не будет считана последняя строка. После чего формируется результат.

Можно использовать оператор EXIST для проверки существования строк, возвращаемых подзапросом.

Следующий пример показывает использование EXIST для получения списка сотрудников, которые управляют другими сотрудниками:

```
SELECT employee_id, last_name
```

FROM employees outer WHERE EXISTS (SELECT employee_id FROM employees inner WHERE inner.manager id = outer.employee id);

EMPLOYEE ID	LAST NAME
1	Smith
2	Johnson

Таблица 6.13 Результат соотнесенного подзапроса с оператором EXIST.

Вложенные подзапросы.

Можно вставлять подзапрос внутрь другого подзапроса, с глубиной до 255 раз. Эту технику необходимо использовать как можно реже, заменяя ее соединениями. Следующий пример содержит вложенный подзапрос:

```
SELECT product_type_id, AVG(price)
FROM products
GROUP BY product_type_id
HAVING AVG(price) <
(SELECT MAX(AVG(price)) FROM
products WHERE product_type_id IN
(SELECT product_id FROM purchases
WHERE quantity > 1) GROUP BY
product_type_id) ORDER BY product
type_id;
```

PRODUCT TYPE ID	AVG(PRICE)
1	24.975
3	13.24
4	13.99

13.49	

Таблица 6.14 Результат вложенного подзапроса.

Второй вложенный подзапрос выбирает нам только продукты, которые были хотя бы один раз заказаны. Первый запрос среди заказанных продуктов ищет максимальную цену среди средних значений цены разных продуктов. Внешний запрос выбирает продукты, у которых средняя цена для их типа не превышает максимальную среднюю цену, найденную вторым подзапросом.

Использование подзапросов с операторами UPDATE и DELETE

Можно также использовать подзапросы внутри операторов UPDATE и DELETE.. В запросе с оператором UPDATE подзапросом можно обновить значение столбца в том случае, если подзапрос возвращает одно значение, т.е. является однострочным. Например, изменить зарплату сотрудника под номером 4 в таблице employee на среднее значение столбца high_halary таблицы salary_grades:

```
UPDATE employees SET salary = (SELECT AVG(high_salary)
    FROM salary_grades) WHERE employee_id = 4;
    1 row updated.
```

Можно использовать строки, возвращаемые подзапросом в конструкции WHERE оператора DELETE. Например, можно удалить сотрудников, чья заработная плата больше среднего значения high_salary таблицы salary_grades:

```
DELETE FROM employees
WHERE salary > (SELECT
AVG(high_salary) FROM
salary_grades); 1 row
deleted.
```

Внимание: Если вы используете операторы UPDATE и DELETE не забывайте выполнять ROLLBACK или COMMIT чтобы восстановить или зафиксировать транзакцию.

Последовательности синонимы и их создание.

Последовательности

Базы данных предназначены для поддержания порядка в массивах информации. Одним из способов упорядочения записей является присваивание им последовательных номеров. Oracle позволяет создавать счетчики, называемые последовательностями (sequences), которые увеличиваются каждый раз, когда к ним происходит обращение. Ссылаясь на последовательность при вставке записей, можно гарантировать, что каждой записи будет присвоен новый уникальный номер.

Для создания последовательности используется следующий синтаксис:

```
CREATE SEQUENCE <имя_последовательности>;
```

Эта простая команда создает последовательность, которая начинается с 1 увеличивается на 1 при каждом обращении. Ничего другого от последовательностей часто и не требуется. Тем не менее, при определении последовательности можно использовать много дополнительных параметров. Взгляните на приведенный ниже синтаксис, где перечислены наиболее полезные из них:

```
CREATE SEQUENCE <имя_последовательности>
[INCREMENT BY <значение_инкремента>]
[START WITH <начальное_значение>]
[MAXVALUE <наибольшее_значение>]
[MINVALUE <наименьшее_значение>] [CYCLE]
```

Параметр INCREMENT BY позволяет создавать последовательности с инкрементом, отличным от 1. Значение этого параметра может содержать до 28 цифр (хотя трудно представить ситуации, где может найти применение такой инкремент!). Если

указать здесь отрицательное число, то значение последовательности будет уменьшаться при каждом обращении.
Параметр START WITH позволяет создать последовательность, начальное значение которой отлично от 1 . Это может пригодиться при создании последовательности для таблицы, уже содержащей записи, чтобы начать последовательность с числа, следующего за наибольшим существующим идентификатором записи.

Параметры MAXVALUE и MINVALUE позволяют ограничить интервал чисел, генерируемых последовательностью. Если использовать их в сочетании с параметром CYCLE, заданное множество значений будет циклически повторяться. Наиболее распространены последовательности с инкрементом 1 и без ограничений на генерируемые значения. Создайте такую последовательность с помощью приведенной ниже команды и переходите к следующему разделу, где будет рассмотрено ее использование.

```
CREATE SEQUENCE test_seq;
```

Чтобы получать значения последовательности, на нее необходимо ссылаться как на таблицу. Последовательности содержат два "псевдостолбца" с именами CURRVAL и NEXTVAL, которые возвращают текущее и следующее значения последовательности соответственно. Выборка из столбца NEXTVAL вызывает автоматический инкремент последовательности.

Чтобы увидеть, как это происходит, введите следующие команды и сравните:

```
SELECT test_seq.nextval FROM DUAL;
SELECT test_seq.nextval FROM DUAL;
SELECT test_seq.nextval FROM DUAL;
```

Теперь вам нужно научиться заполнять столбцы таблицы из последовательности. Это делается путем включения ссылки на последовательность в оператор INSERT, как в приведенных ниже командах[A]

```
CREATE TABLE test (
record_id NUMBER (18, 0),
record_text VARCHAR2 (10)
);
INSERT INTO test VALUES (test_seq.nextval, 'Record A');
INSERT INTO test VALUES (test_seq.nextval, 'Record B');
SELECT * FROM test;
```

Примечание: Хотя последовательности обычно создаются для одной таблицы, в Oracle нет ограничения, которое бы этого требовало. Последовательность является независимым объектом. Она может использоваться в одной таблице, во многих таблицах или ни в одной из таблиц.

В методе, который вы только что использовали, демонстрировалось обращение к последовательности при помощи явной ссылки в операторе INSERT. Можно также организовать автоматическое обращение к последовательности, чтобы не ссылаться на нее в операторе INSERT.

Созданную последовательность можно модифицировать различными способами. В частности, можно изменить значение инкремента, скорректировать или удалить минимальное и максимальное значения, разрешить или запретить циклический повтор по достижении граничного значения.
Синтаксис, позволяющий выполнять эти изменения в существующей последовательности, очень похож на синтаксис, используемый для ее создания. Он выглядит следующим образом:

```
ALTER SEQUENCE <имя последовательности>
[INCREMENT BY <значение_инкремента>] [MAXVALUE
<наибольшее_значение> | NOMAXVALUE] [MINVALUE
```

```
<наименьшее_значение> | NOMINVALUE] [CYCLE |
NOCYCLE]
```

Синонимы

Синоним (synonym) позволяет ссылаться на объект Oracle по имени, которое отличается от его настоящего имени. Синонимы можно определять для таблиц, представлений, последовательностей, а также других объектов - функций, процедур и пакетов. В этом разделе речь пойдет о синонимах таблиц, но все сказанное ниже применимо и к синонимам других объектов.

Зачем нужно создавать синоним для какого-либо объекта? Главным образом для удобства: если вы часто ссылаетесь на таблицу с длинным именем, то по достоинству оцените возможность использования короткого имени без переименования таблицы и изменения кода, который на нее ссылается.

Удобство синонимов проявляется и в том, что они могут облегчить доступ к вашим данным для других людей. Таблицы организуются по идентификатору пользователя Oracle, который их создает, поэтому если другой пользователь захочет обращаться к таблице, созданной вами, то в общем случае ему придется помещать перед именем таблицы ваше имя пользователя, как показано ниже:

```
SELECT * FROM <ваше_имя_пользователя>.<имя_вашей_таблицы>;
```

Это может оказаться утомительным занятием, а таблицу кому-то другому, то вдобавок потребуется менять весь код, который на нее ссылается. Синонимы позволяют сделать таблицу "видимой" для всех, даже если не указано имя ее владельца. Благодаря этому можно писать SQL-операторы, которые будут продолжать работать даже при передаче таблицы другому пользователю.

Создание синонима

Команда создания синонима имеет следующий синтаксис:

```
CREATE [PUBLIC] SYNONYM <имя_синонима> FOR <имя_объекта>
```

Если вам просто нужно сделать таблицу доступной другим пользователям, создайте синоним с тем же именем, что и у таблицы. Вот пример команды такого типа:

```
CREATE PUBLIC SYNONYM product FOR product;
```

Модификация существующего синонима.

Ввиду чрезвычайной простоты синонимов Oracle не предоставляет никаких средств для их изменения. При необходимости просто удалите старый синоним и создайте новый. Команда удаления синонима имеет следующий синтаксис:

```
DROP [PUBLIC] SYNONYM <имя_синонима>;
```

Чтобы удалить первый из созданных выше синонимов (PROD), введите следующую команду:

DROP SYNONYM prod;

Для удаления общего синонима введите такую команду:

DROP PUBLIC SYNONYM product;

Вопросы к лекции

1. В чем заключается процесс группирования?
2. Перечислите агрегатные функции ORACLE?
3. Каково их назначение?
4. Что такое подзапросы?
5. Какие виды подзапросов вы знаете?
6. Каково назначение последовательностей и синонимов в ORACLE?

Список используемой литературы

Том Кайт «ORACLE для профессионалов. Книга 1. Архитектура и основные особенности.» изд. «APress» перев. «ДиаСофтЮП» 2003г 672 стр.
Jason Price «Oracle Database 1 lg SQL» «McGrowHill» 2007г. 656 стр.

Лекция №7. **PL/SQL - процедурное расширение языка SQL. Структура программы на PL/SQL. Переменные и константы и типы**

План

1. PL/SQL - процедурное расширение языка SQL
2. Структура программы на PL/SQL
3. Переменные и константы
4. Комментарии

PL/SQL - процедурное расширение языка SQL.

PL/SQL - это развитый язык программирования, используемый для доступа к базам данных Oracle из различных сред. PL/SQL интегрирован с сервером базы данных, поэтому программы PL/SQL обрабатываются быстро и эффективно. Этот язык доступен и в некоторых клиентских инструментальных средствах Oracle.

PL/SQL предоставляет средства, позволяющие выполнять сложную обработку информации. Вы хотите каждую ночь переносить итоги рабочего дня в сводную таблицу - пакеты (packages) PL/SQL помогут это сделать. Вам нужно своевременно узнавать о поступлении крупных заказов, чтобы привлекать для их обслуживания дополнительных поставщиков, - PL/SQL предоставляет триггеры (triggers), выдающие уведомление, как только объем сделанного заказа превысит установленный вами предел. Вы можете использовать хранимые процедуры (stored procedures) PL/SQL для определения эффективности работы служащих, чтобы на основе этих данных принимать решение о выплате премий. Элегантные функции PL/SQL могут рассчитывать налоговые вычеты для служащих.

PL/SQL позволяет использовать все команды манипулирования данными, управления курсорами и транзакциями, присутствующие в SQL, а также все SQL-функции и операторы. За счет этого вы можете гибко и безопасно манипулировать данными Oracle. Кроме того, PL/SQL полностью поддерживает типы данных SQL, что уменьшает количество преобразований типов при передаче информации между приложениями и базой данных. PL/SQL также поддерживает динамический SQL - усовершенствованную программную технологию, позволяющую делать приложения более гибкими и универсальными. Ваши программы могут создавать и обрабатывать SQL-операторы определения данных, управления данными и управления сеансами "на лету", во время выполнения.

Структура программы на PL/SQL.

Базовой единицей PL/SQL является ***блок (block).*** Все программы PL/SQL строятся из блоков, которые могут быть вложены один в другой.

Как правило, в программе каждый блок выполняет определенную логическую единицу работы, что помогает отделить друг от друга различные задачи. Базовый блок PL/SQL состоит из четырех секций: секции заголовка (header section), необязательной секции объявлений (declaration section), выполняемой секции (execution section) и необязательной секции исключений (exception section):

```
     /* Раздел заголовка - здесь объявляется тип создаваемого
блока PL/SQL. */
     DECLARE
     /* Раздел объявлений - здесь перечисляются переменные, типы,
курсоры и локальные подпрограммы PL/SQL. */
     BEGIN
     /* Выполняемый раздел - процедурные и SQL-операторы. Это
основной раздел блока и единственный обязательный. */
     EXCEPTION
     /* Раздел обработки исключительных ситуаций - здесь
находятся операторы обработки ошибок. */
     END;
```

Существуют два различных типа блоков: анонимные блоки и именованные. Анонимные блоки обычно создаются динамически и выполняются только один раз. Этот тип блоков, как правило, создается в клиентской программе для вызова подпрограммы, хранящейся в базе данных. Именованные блоки отличаются тем, что они имеют имя. Именованные блоки могут быть разбиты на категории следующим образом:

• ***Помеченные блоки (labeled block)*** являются анонимными блоками с меткой, которая дает блоку имя. Они создаются обычно динамически и выполняются только один раз. Помеченные блоки используются так же, как и анонимные блоки, но метка позволяет ссылаться на переменные, которые иначе были бы недоступны.

• ***Подпрограммы (subprogram)*** делятся на процедуры и функции. Они могут храниться в базе данных как автономные объекты, как часть модуля или как методы объектного типа. Подпрограммы обычно не изменяются после своего создания и выполняются неоднократно. Подпрограммы могут объявляться в других блоках. Независимо от того, где они объявлены, подпрограммы выполняются явно посредством вызова процедуры или функции.

• ***Триггеры (triggers)*** - это именованные блоки, которые ассоциируются с некоторым событием, происходящим в базе данных. Они, как правило, не изменяются после своего создания и выполняются многократно неявным образом при наступлении соответствующих событий. Событием, активизирующим триггер, может быть выполнение оператора языка манипулирования данными (DML, data manipulation language) над некоторой таблицей базы данных. Это может также быть оператор языка определения данных (DDL, data definition language), такой как CREATE или DROP, или событие базы данных, например запуск или остановку.

Анонимный блок (anonumous block) - это блок PL/SQL без секции заголовка, иначе говоря, секции имени, поэтому он и называется анонимным. Анонимные блоки могут выполняться из SQL*Plus и использоваться в функциях, процедурах и триггерах PL/SQL. Вспомните, что сами процедуры, функции и триггеры также состоят из базовых блоков. Это означает, что базовый блок можно помещать в другой базовый блок. Чуть ниже вы узнаете об этом более подробно.

По-видимому, лучший способ понять, что представляет собой базовый блок, — это рассмотреть конкретный пример. Сначала введите команду, которая позволит просматривать в SQL*Plus информацию, выводимую программами:

```
set serveroutput on
```

Теперь введите код анонимного блока:

```
DECLARE
   Num_a NUMBER := 6;
   Num_b NUMBER;
         BEGIN
   Num_b = 0;
   Num_a = Num_a / Num_b
   Num b = 7;
```

```
dbms_output.put_line ( ' Value of Num_b ' || Num_b) ; EXCEPTION
WHEN ZERO_DIVIDE
THEN
dbms_output.put_line ('Trying to divide by zero'); dbms_output.put_line ('
Value of Num_a ' || Num_a) ; dbms_output.put_line (' Value of Num_b '
|| Num_b) ; END;
```

Секция заголовка

Секция заголовка блока бывает разных видов в зависимости от того, какому компоненту программы принадлежит блок. Вспомните, что процедуры, функции, триггеры и анонимные блоки состоят из базовых блоков. Как минимум они имеют один базовый блок, составляющий их тело. Этот блок может содержать внутри себя другие базовые блоки. Заголовок базового блока верхнего уровня для функции, процедуры или триггера содержит спецификацию этой функции, процедуры или триггера. Для анонимных блоков заголовок содержит только ключевое слово DECLARE.

Секция объявлений

Секция объявлений не является обязательной. В случае использования она начинается после секции заголовка и оканчивается перед ключевым, словом BEGIN. Эта секция содержит объявления переменных, констант, курсоров, исключений, функций и процедур PL/SQL, которые будут использоваться в выполняемой секции и секции исключений. Все объявления переменных и констант должны размещаться до объявлений функций или процедур.

Объявление сообщает PL/SQL о том, что нужно создать переменную, константу, курсор, функцию или процедуру согласно приведенной спецификации.

Когда выполнение базового блока завершается, все элементы, объявленные в секции объявлений, перестают существовать. Элементы, объявленные в секции объявлений базового блока, могут использоваться только в пределах этого блока.

Таким образом, после выполнения нашего демонстрационного блока в SQL*Plus переменную Num_a будет невозможно передать другой процедуре PL/SQL. Num_a и Num_b после выполнения блока просто исчезают. Но если из выполняемой секции блока будет вызываться функция или процедура PL/SQL, то Num_a и Num_b можно передать в качестве фактических параметров. Одним словом, все, что находится в секции объявлений, принадлежит блоку и может использоваться только внутри него, а следовательно, существует только на протяжении его времени жизни. Часть кода, в которой может использоваться переменная, называется областью видимости (scope).

Областью видимости переменных Num_a и Num_b является блок, в котором они объявлены. Эта область видимости простирается от начала секции объявлений и до конца выполняемой секции.

Выполняемая секция

Выполняемая секция начинается с ключевого слова BEGIN и заканчивается либо ключевым словом EXCEPTION, если присутствует секция исключений, либо ключевым словом END, за которым следуют необязательное имя функции или процедуры и точка с запятой. Выполняемая секция содержит один и более PL/SQL-операторов, выполняемых при передаче управления данному блоку. Структура выполняемой секции показана ниже.

```
BEGIN
/* один и более PL/SQL-операторов */
[EXCEPTION] END [имя функции или
процедуры];
```

В выполняемом коде PL/SQL чаще всего встречается оператор присваивания (: =). Он указывает, что нужно вычислить выражение справа и поместить результат в

переменную слева. Выполняемая секция нашего демонстрационного блока содержит три оператора присваивания. Первый оператор присваивает переменной Num_b нулевое значение.

Второй оператор присваивает переменной Num_a значение Num_a, деленное на Num_b. Обратите внимание, что после успешного вьшолнения этого оператора значение Num_a изменится. Третий оператор присваивает переменной Num_b значение 7.

Секция исключений

В ходе выполнения PL/SQL-оператора может возникнуть ошибка, которая сделает невозможным дальнейшее выполнение программы. Такие исключительные ситуации называются ***исключениями (exceptions).*** Пользователь, вызвавший процедуру, должен быть проинформирован о возникновении исключения, а также о причинах, его вызвавших. Вы можете выдать пользователю содержательное сообщение об ошибке, или предпринять некоторые корректирующие действия и повторить операцию, выполнявшуюся до возникновения ошибки. Вы также можете откатить изменения, которые были произведены в базе данных к этому моменту.

PL/SQL помогает вам во всех этих случаях, предоставляя средства обработки исключений (exception handling). Рассмотрим структуру секции исключений.

```
EXCEPTION
WHEN <имя_исключения>
THEN
/* действия, предпринимаемые при возникновении исключения */
WHEN <имя_исключения>
THEN
/* действия, предпринимаемые при возникновении исключения */
```

Секция исключений начинается с ключевого слова EXCEPTION и продолжается до конца блока. Каждому исключению соответствует оператор WHEN <имя_исключения>, указывающий, что должно быть сделано при возникновении данного исключения. В нашем примере таких операторов три, все они выводят текст на экран SQL*Plus. Пакет DBMS_OUTPUT и процедура PUT_LINE являются частью базы данных Oracle; вместе они позволяют построчно отображать текст на экране SQL*Plus.

Все операторы, находящиеся между оператором, вызвавшим ошибку, и секцией исключений, игнорируются. Таким образом, в демонстрационном блоке присваивание значения 7 переменной Num_b не выполняется. Вы можете убедиться в этом, посмотрев на значение Num_b в выдаваемой распечатке.

Выполнение оператора, указанного в секции исключений, называется ***обработкой исключения (exception handling).***

Процесс, включающий в себя обнаружение ошибки, определение, какое исключение описывает ее наилучшим образом, и передачу PL/SQL информации, позволяющей найти соответствующий код в секции исключений, называется ***возбуждением исключения (raising exception).*** **В** демонстрационном коде исключение возбуждает сам PL/SQL, обнаружив попытку деления на нуль. В PL/SQL это исключение имеет предопределенное имя - ZERO_DIVIDE. Во многих ситуациях ошибку должен обнаруживать ваш код, а не PL/SQL.

Переменные и константы

В блоках PL/SQL взаимодействие с базой данных осуществляется посредством переменных. ***Переменные (variables)*** — это области памяти, в которых могут храниться некоторые значения данных. По мере выполнения программы содержимое переменных модифицируется. Переменной может быть присвоена определенная информация, хранимая

в базе данных, либо содержимое переменной может быть внесено в базу данных. Переменные могут изменяться непосредственно командами PL/SQL. Переменные определяются в разделе объявлений блока.

По существу, переменные - это именованные контейнеры. Они могут содержать информацию (данные) различных видов. В зависимости от того, какую информацию в них можно помещать, они имеют различные типы данных, а чтобы отличать их друг от друга, им присваиваются имена. PL/SQL хранит числа в переменных типа NUMBER, а текст - в переменных типа CHAR или VARCHAR2.

Объявление переменных PL/SQL.

Синтаксис объявления переменной в PL/SQL может иметь любую из следующих форм:

<имя_переменной> <тип_данных> [[NOTNULL] := <значение_по_умолчанию>;

<имя_переменной> <тип_данных> [[NOT NULL] DEFAULT <значение_по_умолчанию>;

<Имя_переменной> - это любой правильный идентификатор PL/SQL. Правильный идентификатор PL/SQL должен:

- Иметь не более 30 символов в длину и не содержать пробельных символов (собственно пробелов и знаков табуляции).
- Состоять только из букв, цифр от 0 до 9, символа подчеркивания (_), знака доллара ($) и знака фунта (#).
- Начинаться с буквы.
- Не совпадать с зарезервированными словами PL/SQL или SQL, которые имеют специальное значение. Например, именем переменной не может быть слово BEGIN, которое обозначает начало выполняемой секции базового блока PL/SQL.

<Тип_данных> - это любой допустимый тип данных SQL или PL/SQL..

Модификатор NOT NULL требует, чтобы переменная имела значение. Если он указан, переменной должно быть присвоено значение по умолчанию. Создаваемой переменной можно присвоить значение по умолчанию, заданное соответствующим выражением. Это просто сокращенный способ присваивания значений переменным.

Объявление констант PL/SQL.

Синтаксис объявления константы имеет следующий вид:

```
<имя_переменной>  <тип_данных> CONSTANT := <значение>;
```

В отличие от переменных константам обязательно присваивается значение, которое нельзя изменять на протяжении времени жизни константы. Константы очень полезны для поддержания безопасности и дисциплины при разработке больших и сложных приложений. Например, если вы хотите гарантировать, что процедура PL/SQL не будет модифицировать передаваемые ей данные, можете объявить их константами. Если процедура все же попытается их модифицировать, PL/SQL возбудит исключение.

Присвоение значения переменным.

Есть три способа изменения значения переменной. Во-первых, ей можно присвоить значение выражения, использовав оператор присваивания PL/SQL. Вы уже видели ряд примеров такого сорта. Вот соответствующий синтаксис:

```
<имя_переменной> := <выражение>;
```

Во-вторых, переменная может быть передана PL/SQL-процедуре в качестве фактического параметра, соответствующего одному из формальных параметров IN OUT или OUT. После завершения процедуры значение переменной может измениться. Это демонстрируется в приведенном ниже примере, где для вызова процедуры использована именованная нотация.

```
CREATE PROCEDURE hike_priees (oldjprice NUMBER, percent_hike
NUMBER := 5, new_price OUT NUMBER) IS BEGIN
       new_price := old_price + old_price * percent_hike / 100;
     END hike_prices;

     DECLARE
       price_to_hike  NUMBER(6,2)   := 20;
       hiked_price NUMBER(6,2)   := 0;
     BEGIN
       dbms_output.put_line('Price before  hike  '   || price_to_hike);
       dbms_output.put_line('hiked_price  before  hike  '    || hiked_price);
       hike_prices   (old_price => price_to_hike,   new_price  => hiked_price)   ;
       dbms_output.put_line ( 'price_to_hike after hike ' ||
price_to_hike) ;
       dhms_output .put_line ( 'hiked_price after hike ' ||
hiked_price) ;
     END
```

Выражения

Выражения PL/SQL используются в качестве значений выражения (rvalues), поэтому использование выражения как самостоятельного оператора бессмысленно - оно должно быть элементом оператора. Например, выражение может находиться в правой части операции присваивания или быть элементом SQL-оператора. Тип выражения определяют составляющие его операции и типы операндов.

Операнд - это аргумент операции. В операциях PL/SQL используется либо один аргумент (унарная, или одноместная, операция), либо два аргумента (бинарная, или двухместная, операция). Например, операция отрицания (-) является унарной, а операция умножения (*) - бинарной.

Приоритет операций выражения определяет порядок их выполнения. Рассмотрим числовое выражение:

3 + 5*7

Приоритет операции умножения выше приоритета операции сложения, поэтому результатом этого выражения будет 38 (3+35), а не 56 (8*7).

Операторы	Тип	Описание
**, NOT	Бинарные	Операция возведения в степень,
***, /**	Унарные	логическое отрицание
=, *!=*,<, >, <=,=>,*IS NULL	Бинарные	Тождественность, отрицание
+,-,II	Бинарные	Умножение, деление
LIKE, BETWEEN, IN	Бинарные (за исключением IS NULL, который является унарным)	Сложение, вычитание, конкатенация строк
AND	Бинарный	Логическое сравнение, Логическое соединение, Логическое включение
OR	Бинарный	Символьные выражения

Таблица 7.1 Операторы PL/SQL.

Существует лишь одна символьная операция - операция конкатенации, или сцепления (||). С ее помощью соединяются две или большее количество символьных строк (или аргументов, которые могут быть неявно преобразованы в символьные строки).

Если все операнды в выражении конкатенации имеют тип CHAR, то и оно само будет иметь тип CHAR. Если же хотя бы один операнд имеет тип VARCHAR2, выражение будет иметь тип VARCHAR2. Считается, что строковые литералы имеют тип CHAR, поэтому результатом предыдущего примера является значение типа CHAR. Однако в следующем блоке переменной vResult присваивается выражение, результат которого значение типа VARCHAR2:

```
DECLARE
v_TempVar
v_Result
BEGIN
v_Result := v_TempVar
END;
```

Логические выражения.

Во всех управляющих структурах PL/SQL (за исключением GOTO) используются логические выражения, называемые также условиями. Логическое, или булево, выражение — это любое выражение, которое дает в результате логическое значение (TRUE (истина), FALSE (ложь) или NULL). Ниже приведен ряд логических выражений:

```
X  >  Y NULL (4  >  5)  OR
(-1  !=  Z)
```

В трех операциях - AND (и), OR (или) и **NOT** (не) - логические значения используются в качестве аргументов и возвращаются в качестве результата. Возможные комбинации описываются в таблицах истинности (см. рис. 5.1). С помощью этих операций реализуется стандартная трехзначная логика. AND возвращает TRUE только в том случае,

если истинны оба операнда, OR возвращает FALSE только тогда, когда оба операнда ложны.

NOT	TRUE	FALSE	NULL
AND	FALSE	TRUE FALSE	NULL
TRUE FALSE NULL OR TRUE FALSE NULL	TRUE FALSE NULL TRUE TRUE TRUE TRUE	FALSE FALSE FALSE FALSE	FALSE NULL NULL
		TRUE FALSE NULL	TRUE NULL NULL

Рис. 7.1 Таблицы истинности.

NULL-значения усложняют логические выражения (напомним, что NULL - это пропущенное или неизвестное значение). Результатом выражения TRUE AND NULL является NULL, так как неизвестно, истинен ли второй операнд.

В операциях сравнения, или отношения, в качестве операндов используются числа, символы или данные, а возвращаются логические значения. Ниже приведена таблица, описывающая эти операции.

Операция	**Описание**
=	Равно (равенство)
<>	Не равно (неравенство)
<	Меньше
>	Больше
<=	Меньше или равно
>=	Больше или равно

Таблица 7.2 Операции сравнения.

Оператор IS NULL возвращает значение TRUE только тогда, когда операндом является NULL. NULL-значения не могут быть проверены на истинность при помощи операций отношения, так как любое выражение отношения, операндом которого является NULL, возвращает NULL.

Оператор LIKE (подобие) применяется для сопоставления строк символов с некоторым образцом, подобно тому, как это делается в регулярных выражениях системы Unix. Знак подчеркивания (_) соответствует одному символу, а знак процента (%) - нулю и более символам. Приведенные ниже выражения возвращают TRUE:

```
'Scott'  LIKE  'Sc%t'
'Scott'  LIKE  'Scott'
'Scott'  LIKE  ‘%
```

Оператор BETWEEN (между) объединяет операции <= и >= в одном выражении. Например, приведенное ниже выражение возвращает значение FALSE:

100 BETWEEN 110 AND 12 0 A

это выражение возвращает TRUE:

```
100 BETWEEN 90 AND 110
```

Оператор IN (в) возвращает TRUE, если первый операнд содержится в наборе, определяемом вторым операндом. Например, результат этого выражения - FALSE:

'Scott' IN ('Mike', 'Pamela', 'Fred')

Если в наборе содержатся NULL-значения, они игнорируются, так как при сравнении некоторого значения с NULL всегда будет возвращаться NULL.

Комментарии

Комментарии повышают удобочитаемость программ и делают их более понятными. Компилятор PL/SQL игнорирует комментарии. Существуют комментарии двух видов: однострочные и многострочные. Последние, часто называют комментариями С-типа.

Однострочные комментарии

Однострочный комментарий начинается с двух символов тире и продолжается до конца строки (ограниченной символом возврата каретки).

Рассмотрим некоторый блок PL/SQL:

```
DECLARE
  v_Department CHAR(3);
  v_Course NUMBER;
BEGIN
  INSERT INTO classes (department, course) VALUES
(v_Department, v_Course); END;
```

Теперь добавим к этому блоку однострочные комментарии, чтобы сделать его более понятным:

```
DECLARE
  v_Department CHAR(3);      -- Переменная для хранения
-- трехсимвольного -- кода факультета v_Course NUMBER; --
Переменная для хранения номера курса BEGIN
  -- Введем в таблицу classes базы данных курс, указанный в
  -- v_Department и v_Course.
  INSERT INTO classes (department, course) VALUES
(v_Department, v_Course); END;
```

Внимание: Если комментарий распространяется более чем на одну строку,

то двойное тире (—) необходимо указывать в начале каждой строки.

Многострочные комментарии

Многострочные комментарии начинаются с ограничителя /* и заканчиваются ограничителем */, как это делается в языке программирования С.

Например:

```
DECLARE
```

```
        v_Department CHAR(3);          /* Переменная для хранения
трехсимвольного кода факультета */
        v_Course NUMBER;               /* Переменная для хранения
номера курса */
    BEGIN

        /* Введем в таблицу classes базы данных курс, указанный в v_Department и
        v_Course. */
  INSERT INTO classes (department, course) VALUES (v_Department,
 v_Course); END;
```

Многострочные комментарии могут распространяться на сколь угодно большое число строк, однако они не могут быть вложенными, т.е. до начала одного комментария другой должен быть закончен.

Вопросы к лекции

1. Что такое PL/SQL?
2. Как выглядит структура программы на PL/SQL?
3. Как объявляются переменные и константы на PL/SQL?
4. Каково назначение комментариев на PL/SQL?

Список используемой литературы

Скотт Урман «ORACLE 9i Программирование на языке PL/SQL.» изд. «ORACLE Press» перев. «Лори» 2004г. 529 стр.

Michael McLaughlin «Oracle Database 1 lg PL/SQL Programming» изд. «McGrowHiU» 2008r. 835 стр.

Лекция №8 **Управление выполнением программы: операторы ветвления, цикла, GOTO. Обработка исключительных ситуаций.**

План

1. Управление выполнением программы
2. Операторы ветвления в ORACLE.
3. Циклы
4. Оператор CONTINUE
5. Оператор GOTO и метки

Управление выполнением программы

Во многих случаях программа должна выполнять разные действия в зависимости от того, выполнено ли некоторое условие. В PL/SQL, как и в других языках программирования третьего поколения, имеются различные структуры, служащие для управления работой блока. Этими структурами являются условные операторы (операторы ветвления) и циклы, а также оператор безусловного перехода. Именно эти структуры совместно с переменными обеспечивают мощь и гибкость PL/SQL.

Операторы ветвления в ORACLE.

IF-THEN-ELSE.

Синтаксис оператора IF-THEN-ELSE (если-то-иначе):

```
IF <логическое_выражение> THEN
  <последовательъностъ_операторов1>;
 [ELSIF <логическое_выражение2> THEN
  <последовательностъ_операторов2>; ]
 [ELSE
  < последовательностъ_операторов2>; ]
END IF;
```

где <логическое_выражение> - любое выражение, результатом которого является логическое значение (см. выше). Условия ELSIF и ELSE необязательны, причем условий ELSIF может быть сколь угодно много. Например, ниже представлен блок, демонстрирующий использование оператора IF-THEN-ELSE с одним условием ELSIF и с одним условием ELSE:

```
DECLARE
   v_NumberSeats rooms.number_seats%TYPE;
   v_Comment VARCHAR2(35);
BEGIN
/* Извлекаем число мест в аудитории, идентификатор которой 20008. Сохраняем
результат в v_NumberSeats. */ SELECT number_seats INTO v_NumberSeats
      FROM rooms WHERE room_id = 20008; IF
v_NumberSeats < 50 THEN
   v_Comment := 'Fairly small'; ELSIF
v_NumberSeats < 100 THEN
   v_Comment := 'A little bigger'; ELSE
   v_Comment := 'Lots of room'; END
IF; END;
```

Этот блок функционирует в точности так, как это указано с помощью ключевых слов. Если первое условие истинно, выполняется первая последовательность операторов.

В нашем случае первым условием является

- v_NumberSeats < 50

А первая последовательность операторов

```
v_Comment := 'Fairly small[1];
```

Если число мест не менее 50, оценивается второе условие:

- v_NumberSeats < 100

Если оно истинно, вьшолняется вторая последовательность операторов:

- v_Comment := 'A little bigger';

Наконец, если число мест не менее 100, вьшолняется завершающая последовательность операторов:

- v_Comment := 'Lots of room';

Каждая последовательность операторов выполняется только в случае истинности соответствующего условия. В этом примере в каждой последовательности операторов имеется только один процедурный оператор. Однако такие последовательности могут содержать сколь угодно операторов (процедурных либо SQL-операторов).

Например:

```
DECLARE
```

```
      v_NumberSeats rooms.number_seats%TYPE;
      v_Comment VARCHAR2(35);
    BEGIN
/* Извлекаем число мест в аудитории, идентификатор которой 20008.
Сохраняем результат в v_NumberSeats. */ SELECT number_seats INTO
v_NumberSeats
      FROM rooms WHERE room_id = 20008; IF
    v_NumberSeats < 50 THEN
      v_Comment := 'Fairly small';
      INSERT INTO temp_table (char_.col) VALUES ('Nice and
cozy');
    ELSIF v_NumberSeats < 100 THEN
      v_Comment := 'A little bigger';
      INSERT INTO temp_table (char_col) VALUES ('Some breathing
room') ;
    ELSE
      v_Comment := 'Lots of room'; END
    IF; END;
```

Внимание: Обратите внимание на правописание ELS IF — в этом слове отсутствует E и нет пробела. Такой синтаксис заимствован из языка программирования Ada.

Последовательность операторов в операторе IF-THEN-ELSE выполняется только в случае истинности соответствующего условия. Если результатом условия является FALSE или NULL, последовательность операторов не вьшолняется. В качестве примера рассмотрим два блока:

```
    /* Блок 1 */
    DECLARE
      v_Numberl NUMBER;
      v_Number2 NUMBER;
      v_Resuit VARCHAR2(7);
    BEGIN
      IF v_Numberl < v_Number2 THEN v_Result := 'Yes';
      ELSE v_Result := 'No';
      END IF;
    END;
    /* Блок 2 */
    DECLARE
      v_Numberl NUMBER;
      v_Number2 NUMBER;
      v_Resuit VARCHAR2(7) ;
    BEGIN
      IF v_Numberl >= v_Number2 THEN v_Result := 'No';
      ELSE v_Result := 'Yes';
      END IF;
    END;
```

Одинаково ли будут вести себя эти два блока? Предположим, что v_Numberl = 3, а v_Number2 = 7. В результате условие блока 1 (3 < 7) будет истинным, и переменная v_Result будет установлена в 'Yes'. Условие блока 2 будет ложным (3 >= 7), переменная v_Result также будет установлена в 'Yes'. Для любых значений v_Numberl и v_Number2, не являющихся NULL, приведенные блоки функционируют одинаково.

Теперь предположим, что v_Numberl = 3, а v_Number2 содержит NULL. Что произойдет в этом случае? Условие блока 1 (3 < NULL) дает в результате NULL, поэтому

будет выполнено условие ELSE и переменной v_Result будет присвоено значение 'No'. Условие блока 2 (3 >= NULL) также дает в результате NULL, поэтому будет выполнено условие ELSE и переменной v_Result будет присвоено значение 'Yes'. Если одна из переменных (v_Numberl или v_Number2) содержит NULL, то блоки функционируют по-разному.

Оператор CASE.

Существует два типа оператора CASE в PL/SQL. Оба используют селектор. Селектор - это переменная, функция или выражение, которое CASE пытается сравнить с блоком WHEN. Селектор следует сразу после ключевого слова CASE. Если вы не будете использовать переменную-селектор, то PL/SQL использует логическое значение TRUE в качестве переменой-селектора. В качестве переменной-селектора можно использовать любой тип данных PL/SQL за исключением BLOB, BFILE или составного типа.

Общий синтаксис оператора CASE выглядит следующим образом:

```
CASE [ TRUE | [<переменная_селектор>]] WHEN
[<выражение 1> | Оначение 1>] THEN
   <блок операторов 1>; WHEN [<выражение 2> |
Оначение 2>] THEN
   <блок операторов 2>;
WHEN [<выражение (n+1)> | Оначение (N+1)>] THEN
   <блок операторов (п+1)>;
ELSE
   <блок опереторов>; END
CASE;
```

В *простом операторе CASE* селектор - это переменная, которая возвращает значение любого простого типа данных за исключением типа Boolean. В ***операторе*** *CASE с* ***поиском*** селектором является логическая переменная или функция, возвращающая логическое значение. Переменная-селектор - это логическая переменная, принимающая по умолчанию значение TRUE. В операторе CASE с поиском селектором пренебрегают в случае, если выражение стоящее после WHEN принимает значение TRUE. Как и в выражении IF, CASE также может иметь выражение ELSE. ELSE срабатывает, ели не найден не один подходящий вариант WHEN. Нельзя оставлять неопределенным выражение ELSE иначе вы получите ошибку CASE_NOT_FOUND или PLS-06592, если ни одно значение WHEN, соответствующее переменной селектору не будет найдено.

Оператор CASE является блоком операторов. Оно начинается с метки-идентификатора или ключевого слова CASE и заканчивается с завершающим идентификатором END CASE и точкой с запятой. Все блоки операторов требуют хотя бы один оператор также как и все анонимные или именованные блоки. Оператор CASE требует хотя бы один оператор в каждом выражении WHEN и блок ELSE.

Также как и оператор IF-THEN-ELSIF-THEN-ELSE оператор CASE выполняет блоки WHEN последовательно, проверяя на соответствие переменной-селектора этому блоку WHEN, а затем прерывает выполнение оператора CASE. Не возможно неудачное поведение оператора в PL/SQL. Поэтому ELSE выполняется только если ни один из выражений WHEN не соответствует селектору или ни одно выражение WHEN не принимает значение TRUE.

Простой оператор CASE.

Простой оператор CASE принимает в качестве переменной-селектора любой тип данных PL/SQL за исключением BLOB, BFILE или составного типа.

Синтаксис простого оператора CASE выглядит следующим образом:

```
CASE <переменная_селектор> WHEN
Оначение 1> THEN
  <блок операторов 1>; WHEN
Оначение 2> THEN
  <блок операторов 2>;
WHEN Оначение (N+l)> THEN
  <блок операторов (N+l)>; ELSE
  <блок-операторов>; END
CASE;
```

Простой оператор CASE требует определения переменной-селектора. Вы можете большое количество блоков WHEN, как показано в примере, но при увеличении возможных значений уменьшается эффективность такого решения. Такое решение является легко управляемым, если у вас есть не более 10 возможных значений для выбора. Управляемость снижается при увеличении блоков WHEN.

```
DECLARE
  selector NUMBER := 0;
BEGIN
  CASE selector
  WHEN 0 THEN
    dbms_output.put_line('Case 0 ! ') ;
  WHEN 1 THEN
    dbms_output.put_line('Case 1!');
  ELSE
  dbms_output.put_line('No match!');
END CASE; END; /
```

Анонимный блок выбирает первое сравнение, поскольку переменная-селектор содержит значение 0. По этой причине, первый блок WHEN соответствует переменной-селектору. Оператор CASE прекращает выполнение поиска по выражениям WHEN и выполняет блок операторов найденного WHEN до завершения выполнения оператора. Анонимный блок выводит на экран «Case O!».

Можно заменить переменную-селектор любым другим типом данных PL/SQL. Возможным выбором могут стать CHAR, NCHAR, и VARCHAR2.

Оператор CASE с поиском.

В операторе CASE с поиском селектор установлен неявным образом, пока вы пытаетесь найти условие TRUE или FALSE. Часто оператор CASE с поиском основан на динамическом использовании во время выполнения логики. В качестве выражения WHEN можно использовать функцию, возвращающую логическое значение, при условии динамического определения переменной-селектора. Оператор CASE с поиском использует только логические выражения в выражениях WHEN.

Синтаксис оператора CASE с поиском следующим образом:

```
CASE [{TRUE | FALSE}]
WHEN [<выражение 1> | <логическое_значение 1>] THEN
  <блок операторов 1>; WHEN [<выражение 2> |
<логическое_значение 2>] THEN
  <блок операторов 2>;
WHEN [<выражение (N+1)> | <логическое_значение (N+1)>] THEN
  <блок операторов (N+l)>; ELSE
  <блок операторов>; END
CASE;
```

Как и в простом операторе CASE вы можете добавлять любое количество блоков WHEN.

Рассмотрим пример оператора CASE с поиском:

```
BEGIN
   CASE
   WHEN 1=2 THEN
      dbms_output.put_line('Case  [1 = 2 ] ' ) ;  WHEN
   2=2  THEN
      dbms_output.put_line('Case  [2  =  2]'); ELSE
   dbms_output.put_line('No match'); END CASE;
END; /
```

Анонимный блок выполнит второй блок WHEN потому, что селектором по умолчанию является значение TRUE и именно второй блок соответствует значению TRUE. Будет выведено на экран «Case [2 = 2]».

Циклы

В PL/SQL можно повторять операторы посредством циклов (loops). Циклы подразделяются на четыре категории. Простые циклы, циклы WHILE и циклы FOR, а также курсорные циклы FOR, которые будут рассматриваться в следующих лекциях. Выход из цикла осуществляется в зависимости от некоторого условия.

Простой цикл LOOP.

Конструкция цикла LOOP имеет следующий синтаксис:

```
«имя_цикла»
LOOP
   <операторы>; EXIT  <имя_цикла>  [WHEN
<условие_выхода>] ;
   <операторы>;
END  LOOP;
```

При наличии конструкции WHEN все операторы в теле цикла повторяются до тех пор, пока выражение <условие_выхода> не примет положительное значение (т.е. не станет истинным). Условие выхода проверяется на каждом проходе, иначе называемом итерацией. Как только выражение принимает значение TRUE, все операторы после EXIT пропускаются, итерации прекращаются и выполнение продолжается с первого оператора, следующего за END LOOP. Если условие WHEN отсутствует, операторы между LOOP и EXIT выполняются только один раз. Очевидно, что опустив условие WHEN, вы поступите нелогично. В конце концов идея цикла состоит в том, чтобы обеспечить потенциально многократное выполнение кода.

В этом примере просто распечатываются первые десять чисел.

```
DECLARE
BEGIN
  just_a_num NUMBER := 1;;
  «just_a_loop»
  LOOP
    dbmsAoutput  ,put_line  (just_a_num)  ; EXIT
    just_a_loop
      WHEN  (just_a_num >=  10);
    just_a_num  := just_a_num +  1;
  END LOOP;
END;
```

Каждая итерация увеличивает just_a_num на 1. По достижении значения 10 выполняется условие выхода и цикл завершается.

Цикл WHILE.

Синтаксис цикла WHILE (цикл с условием продолжения) таков:

```
WHILE <условие> LOOP
  <последовательность_операторов>; END
LOOP;
```

Проверка условия происходит перед каждой итерацией (шагом) цикла. Если условие истинно, выполняется последовательность операторов. Если же проверка условия дает ложное или NULL-значение, цикл завершается и управление программой передается оператору, следующему за оператором END LOOP. Теперь перепишем рассматриваемый блок, применив цикл WHILE:

```
DECLARE
  v_Counter BINARY_INTEGER := 1;
BEGIN
  - Проверяем счетчик цикла перед каждой итерацией
  - на предмет того, что значение счетчика меньше 50.
  WHILE v_Counter <= 50 LOOP
    INSERT INTO temp_table
  VALUES (v_Counter, 'Loop index');
v_Counter := v_Counter + 1; END LOOP;
END;
```

Чтобы прервать цикл и выйти из него, можно внутри цикла WHILE использовать оператор EXIT или EXIT WHEN.

Учтите, что если при первой проверке условие цикла не истинно, цикл не выполняется ни разу. Если в нашем примере убрать инициализацию переменной v_Counter, то результатом проверки условия v_Counter <= 50 будет NULL-значение и в таблицу temptable не будет введено ни одной строки:

```
DECLARE
  v_Counter BINARY_INTEGER;
BEGIN
  - Результатом проверки условия будет NULL, так как по
  - умолчанию счетчик v_Counter содержит NULL-значение.
  WHILE v_Counter <= 50 LOOP
    INSERT INTO temp_table
    VALUES (v_Counter, 'Loop index');
v_Counter := v_Counter + 1; END LOOP;
END;
```

Цикл FOR.

В цикле FOR для подсчета итераций используется переменная-счетчик, называемая также индексом цикла (loop index). По завершении каждой итерации счетчик увеличивается, начиная с нижнего предела, или уменьшается, начиная с верхнего предела. Как только его значение выйдет за указанный диапазон, цикл завершается. Синтаксис цикла FOR выглядит следующим образом:

```
FOR <счетчик> IN [REVERSE] <нижняя_граница> .. <верхняя граница> LOOP
    <операторы>;
END LOOP;
```

Рассмотрим цикл FOR, использовав приведенный ниже код:

```
BEGIN
  FOR just_a_num  IN  1..10
  LOOP
    dbms_output .put_line (just_a_num) ;
  END LOOP;
END; /
```

Если использовать в этом цикле FOR команду REVERSE, то в результате числа должны быть показаны в обратном порядке - от 10 до 1.

Оператор CONTINUE

СУБД ORACLE 11g предлагает новую возможность для циклов: оператор CONTINUE. Использование этого оператора прерывает текущую итерацию цикла и немедленно переходит к выполнению следующей итерации. Этот оператор, как и EXIT, имеет две формы: CONTINUE и CONTINUE WHEN. Рассмотрим простой пример оператора CONTINUE WHEN для пропуска выполнения тела цикла для четных чисел:

```
BEGIN
  FOR l_index IN 1 .. 10
  LOOP
  CONTINUE WHEN MOD (l_index, 2) = 0;
DBMS_OUTPUT.PUT_LINE ('Loop index = ' || TO_CHAR
(l_index));
  END LOOP;
END; /
```

Результат выполнения анонимного блока описан ниже:

```
Loop index = 1
Loop index = 3
Loop index = 5
Loop index = 7
Loop index = 9
```

Конечно же вы могли бы использовать оператор IF вместе с CONTINUE для получения такого же результата, однако оператор CONTINUE WHEN предоставляет более элегантное решение и является более описательным для логики, которую вы пытаетесь реализовать. К тому же при вьшолнении оператора CONTINUE WHEN выполняются точно определенные действия, и выполнение переходит к следующей итерации цикла.

Также CONTINUE может прервать вложенный цикл и перейти к выполнению следующей операции внешнего цикла. Чтобы сделать это, вам необходимо использовать метки. Вот как выглядит пример:

```
BEGIN
  «outer»
  FOR outer_index IN 1 .. 5
  LOOP
    DBMS_OUTPUT.PUT_LINE ('Outer index = ' || TO_CHAR
(outer_index));
    <<inner»
    FOR inner_index IN 1 .. 5
    LOOP
      DBMS_OUTPUT.PUT_LINE (' Inner index = ' || TO_CHAR
(inner_index));
```

```
    CONTINUE outer; END
  LOOP inner; END LOOP
  outer; END; /
```

Результат выполнения анонимного блока будет следующим:

```
Outer index = 1
Inner index = 1
Outer index = 2
Inner index = 1
Outer index = 3
Inner index = 1
Outer index = 4
Inner index = 1
Outer index = 5
Inner index = 1
```

Оператор GOTO и метки

В языке программирования PL/SQL имеется оператор безусловного перехода GOTO Его синтаксис:

```
GOTO <метка>
```

где метка - это метка, определяемая в блоке PL/SQL. Метки заключаются в двойные угловые скобки. При выполнении оператора GOTO управление программой сразу же передастся оператору, на который указывает метка. Представим рассматриваемый пример цикла следующим образом:

```
DECLARE
  v_Counter BINARY_INTEGER := 1;
BEGIN
LOOP
    INSERT INTO temp_table VALUES (v_Counter, ' Loop
count');
    v_Counter := v_Counter + 1;
    IF v_Counter > 50 THEN
    GOTO !_EndOfLoop: END
  IF; END LOOP;
  «!_EndOfLoop» --Пример метки
  INSERT INTO temp_table (char_col) VALUES ('Done!');
END;
```

Ограничения при использовании GOTO.

В PL/SQL на использование операторов GOTO налагаются определенные ограничения. Нельзя передавать управление программой во внутренний блок, цикл или оператор IF. Приведем пример:

```
BEGIN
  GOTO l_InnerBlock;  -- Неверно, так как нельзя передавать
                      -- управление во внутренний блок
  BEGIN
  «l_InnerBlock» END; GOTO l_lnsidelf;   -- Неверно, так как
  нельзя передавать
                  -- управление в оператор IF IF x
> 3 THEN «l_lnsidelf» INSERT INTO ... END IF; END;
```

Если бы это допускалось, то операторы внутри оператора IF могли бы быть выполнены, даже если бы условие IF не было истинным. В примере, приведенном выше,

оператор INSERT выполнился бы даже тогда, когда x = 2. Кроме того, запрещается передавать управление из одной последовательности операторов условного оператора IF в другую:

```
BEGIN
  IF x > 3 THEN
    GOTO l_NextCondition;
  ELSE
  «l_NextCondition»
END IF; END;
```

Наконец, нельзя передавать управление из обработчика исключительных ситуаций обратно в текущий блок:

```
DECLARE
  v_Room rooms%ROWTYPE;
BEGIN
  -- Выберем одну строку в таблице rooms.
  SELECT * INTO v_Room FROM rooms WHERE rowid = 1;
  «l_Insert»
  INSERT INTO temp_table (char_col) VALUES ('Found a row!);
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    GOTO l_Insert;   -- Неверно, так как нельзя передавать
                     -- управление в текущий блок
END;
```

Помеченные циклы.

Циклы могут быть помечены. При этом метка может быть указана в операторе EXIT для определения цикла, который нужно прервать. Например:

```
BEGIN
  «l_0uter»
  FOR v_Outerlndex IN 1.. 50 LOOP
    «l_Inner»
    FOR v_lnnerlndex IN 2.. 10 LOOP IF
      v_0uterlr«iex > 40 THEN
        EXIT l_0uter;          -- выход из обоих циклов
  END IF; END LOOP
l_Inner; END LOOP l_0uter;
END
```

Если цикл помечен, имя метки можно указывать после оператора END LOOP, как показано в этом примере.

Рекомендации по применению GOTO.

При использовании операторов GOTO следует соблюдать осторожность Липшие операторы GOTO могут привести к нарушению структуры программы: управление программой может перемещаться с места на место без видимой причины, что затруднит ее понимание и сопровождение.Все случаи применения операторов GOTO могут быть представлены с помощью других управляющих структур PL/SQL, например циклов или условных выражений. Для выхода из вложенного цикла можно вместо .оператора перехода к концу блока воспользоваться исключительной ситуацией.

NULL как оператор

В некоторых случаях необходимо явно указывать, что никаких действий выполнять не нужно. Это можно сделать при помощи оператора NULL, который не приводит ни к каким действиям; он служит лишь в качестве заполнителя. Например:

```
DECLARE
  v_TempVar NUMBER := 7;
BEGIN
  IF v_TempVar < 5 THEN
    INSERT INTO tempjtable (char_col) VALUES ('Too smalll);
     ELSIF vJTempVar < 20 THEN NULL;      — никаких
действий
                                          -- не выполняется
  ELSE
  INSERT INTO temp_table (char_col) VALUES (Too bigl);
END IF; END;
```

Вопросы к лекции

1. Каким образом происходит управление выполнением программы на PL/SQL?
2. Какова функция операторов ветвления ?
3. Какова функция операторов цикла ?
4. Что значит оператор GOTO ? А оператор метки?

Список используемой литературы

Кристофер Аллен «ORACLE PL/SQL» изд. «McGrowHil» перев. «Лори» 2001г. 350 стр.

Jason Price «Oracle Database 1 lg SQL» «McGrowHiU» 2007г. 656 стр.

Steven Feuerstein «Oracle PL/SQL Programming» изд. «O'Reilly» 2009г. 1256 стр.

Лекция №9. **Процедуры, функции и пакеты. SQL-функции ORACLE. Параметры функций и процедур.**

План

1.Процедуры и функции
2.Параметры функций и процедур
3.Пакеты
4.Встроенные функции

Процедуры

Процедуры и функции PL/SQL очень похожи на процедуры и функции, используемые в других языках третьего поколения, и обладают аналогичными свойствами. В совокупности процедуры и функции называются ***подпрограммами (subprogram).***

Хранимая процедура - это определенный набор инструкций, написанных на языке PL/SQL. Вызов процедуры приводит к выполнению содержащихся в ней инструкций. Процедура хранится в базе данных, поэтому и называется хранимой.

Хранимая процедура может выполнять SQL-операторы и манипулировать данными в таблицах. Ее можно вызывать из другой хранимой процедуры PL/SQL, хранимой функции или триггера, а также непосредственно из строки приглашения SQL*Plus. По мере чтения главы вы научитесь использовать все перечисленные методы вызова.

Процедура состоит из двух основных частей: спецификации и тела. Спецификация процедуры (procedure specification) включает в себя имя процедуры и описание ее входных и выходных данных. Эти входные и выходные данные называются формальными параметрами (formal parameters) или формальными аргументами (formal argimients). Если при вызове процедуры указываются параметры командной строки или другие входные данные, эти значения называются фактическими (actual) параметрами или фактическими аргументами.

Как и другие операторы CREATE, создание процедуры является операцией DDL, поэтому до и после создания процедуры неявно выполняются операторы COMMIT. При этом можно использовать как ключевое слово IS, так и ключевое слово AS - они эквивалентны друг другу.

Тело процедуры Тело (body) процедуры — это блок PL/SQL, содержащий раздел объявлений, выполняемый раздел и раздел исключительных ситуаций. Раздел объявлений располагается между ключевым словом IS или AS и ключевым словом BEGIN; выполняемый раздел (единственный обязательный) - между ключевыми словами BEGIN и EXCEPTION, а раздел исключительных ситуаций - между ключевыми словами EXCEPTION и END.

Совет: В объявлении процедуры или функции ключевое слово DECLARE отсутствует. Вместо него используется ключевое слово IS или AS. В этом просматривается аналогия PL/SQL с языком программирования Ada.

Следовательно, структура оператора создания процедуры такова:

```
CREATE [OR REPLACE] PROCEDURE <имя_процедуры>
([список_парометров]) AS | IS /* Раздел объявлений */
BEGIN
```

/* Выполняемый раздел */
EXCEPTION
/* Раздел исключительных ситуаций */ END
[имя_процедуры];

При объявлении процедуры ее имя можно при желании указать после последнего оператора END. Если после END идет идентификатор, он должен соответствовать имени процедуры.

Совет: Указание имени процедуры в конце является хорошим стилем программирования, так как выделяет оператор END, соответствующий оператору CREATE, что позволяет выявлять несоответствия в парах BEGIN-END на самых ранних этапах программирования.

Теперь рассмотрим некоторые примеры спецификаций процедур. (Помните, что спецификация не содержит никакого кода; в ней определяется только имя процедуры, а также ее входные и выходные параметры.)

```
CREATE PROCEDURE rtm_ytd_reports
```

Эта простая спецификация содержит только имя процедуры. Данная процедура не имеет параметров.

```
CREATE PROCEDURE  increase_prices (percent_increase NUMBER)
```

Этой процедуре при вызове может быть передано значение. Внутри процедуры значение будет известно под именем PERCENTENCREASE. Обратите внимание, что здесь указан тип данных: NUMBER.

```
CREATE PROCEDURE increase_salary_find_tax (increase_percent
IN NUMBER := 7, sal IN OUT NUMBER, tax OUT NUMBER)
```

Здесь мы видим процедуру с тремя формальными параметрами. Слово IN после имени параметра означает, что при вызове процедура может считать из этого параметра входное значение. Слово OUT означает, что процедура может использовать данный параметр для возврата значения в ту программу, из которой она была вызвана. Комбинация IN OUT после имени параметра говорит о том, что параметр может использоваться как для передачи значения процедуре, так и для возврата значения.

Параметру INCREASE_PERCENT в этом примере присвоено значение по умолчанию (default value), равное 7, путем добавления := 7 после типа данных.

Таким образом, если процедура будет вызвана без указания процента прироста, она увеличит переданное значение зарплаты на 7% и рассчитает налог, исходя из новой зарплаты.

Примечание: Типы данных в процедуре не могут иметь спецификаций размера. Например, вы можете указать для параметра тип данных NUMBER, Но не NUMBER (10,2).

Тело процедуры (procedure body) — это блок PL/SQL-кода.

```
CREATE OR REPLACE PROCEDURE AddNewStudent (
p_FirstName students.first_name%TYPE,
p_LastName students.last_name%TYPE,
p_Major students.major%TYPE) AS
BEGIN
-- Внесем новую строку в таблицу students. Воспользуемся
-- последовательностью students_sequence для
-- генерации идентификатора нового студента и 0 для
-- current_credits,
INSERT INTO students (ID, first_name, last_name,
major, current_credits)
```

```
VALUES (student_sequence.nextval, p_FirstName, p_LastName,
p_Major, 0);
END AddNewStudent;
```

В этом примере демонстрируется ряд важных аспектов:

• Процедура AddNewStudent создается при помощи оператора CREATE OR REPLACE PROCEDURE. После создания процедуры она сначала компилируется, а затем сохраняется в базе данных в скомпилированном виде. Скомпилированный код можно впоследствии выполнить из другого блока PL/SQL. Исходный код процедуры также сохраняется в базе данных.
• При вызове процедуры ей можно передавать параметры. В рассмотренном примере процедуре во время ее выполнения передаются имя, фамилия и профилирующий предмет студента. Внутри процедуры параметр p_FirstName будет иметь значение 'Zelda', p_LastName 'Zudnik' , а p_Major - 'Computer Science', так как именно эти литералы указаны при вызове процедуры.
• Вызов процедуры - это оператор PL/SQL; в выражениях процедуры не вызываются. При вызове процедуры управление программой передается первому исполняемому оператору внутри нее. Когда процедура заканчивается, управление возвращается оператору, следующему за вызовом процедуры. В этом смысле процедуры PL/SQL функционируют точно так же, как и процедуры других языков третьего поколения (языков 3GL). Функции вызываются в выражениях.
• Процедура - это блок PL/SQL, в состав которого входят раздел объявлений, выполняемый раздел и раздел обработки исключительных ситуаций. Как и в анонимных блоках, необходимым здесь является только выполняемый раздел. Например, в процедуре AddNewStudent содержится лишь выполняемый раздел.

Функции

Функция PL/SQL похожа на процедуру PL/SQL: она также имеет спецификацию и тело. Главное различие между процедурой и функцией в том, что функция предназначена для возврата значения, которое может использоваться в более крупном SQL-Операторе.

Функции похожи на процедуры. И те, и другие получают некоторые параметры того или иного вида (параметры описываются ниже). Функции и процедуры - это различные формы блоков PL/SQL, в состав каждого из которых могут входить раздел объявлений, выполняемый раздел и раздел исключительных ситуаций. Как функции, так и процедуры можно хранить в базе данных или объявлять в блоке. Однако вызов процедуры является оператором PL/SQL, в то время как вызов функции - это часть некоторого выражения.

Синтаксис, применяемый при создании хранимой функции, похож на синтаксис создания процедуры:

```
CREATE [OR REPLACE] FUNCTION <имя_функции>
  ([<список_парометров>]) RETURN <возвращаемый_тип> {IS | AS}
/* Раздел объявлений */ BEGIN
/* Выполняемый раздел */
RETURN <выражение>;
EXCEPTION
/* Раздел исключительных ситуаций */
END [имя_функции]
```

где <имя_функции> - это имя функции, аргумент и тип аналогичны аргументу и типу, указываемым при создании процедуры, <возвращаемый_тип> - это тип значения, возвращаемого функцией, а <тело_функции> - блок PL/SQL, содержащий программный код данной функции. Для тела функции применимы те же правила, что и для тела

процедуры. Например, имя функции можно при желании указать после закрывающего END.

Внутри тела функции оператор RETURN используется для возврата управления программой в вызывающую среду с некоторым значением. Общий синтаксис оператора RETURN:

RETURN <выражение>;

где <выражение> — это возвращаемое значение. Значение выражения преобразуется в тип, указанный в команде RETURN при описании функции, если, конечно, это значение уже не имеет данный тип. При выполнении оператора RETURN управление программой сразу же возвращается в вызывающую среду.

В функции может быть несколько операторов RETURN, хотя выполняться будет только один из них. Отсутствие в функции оператора RETURN является ошибкой. Ниже приведен пример использования в одной функции нескольких операторов RETURN. Хотя в этой функции пять различных операторов RETURN, выполняется лишь один из них, какой именно — зависит от заполнения учебной группы, указанной при помощи

```
CREATE OR REPLACE FUNCTION Classlnfo (
/* Возвращает 'Full1, если группа заполнена до предела,
Some Room1, если группа заполнена на 80%,
More Room', если группа заполнена на 60%,
Lots of Room', если группа заполнена менее чем на 60%, и
Empty1, если ни одного студента не зарегистрировано. */
p_Department classes.department%TYPE,
p_Course classes.course%TYPE)
RETURN VARCHAR2 IS
v_CurrentStudents NUMBER;
 v_MaxStudents
 NUMBER;
 v_PercentFull
 NUMBER; BEGIN
-- Получим текущее и максимальное число студентов в
указанной группе.
SELECT current_students, max_students
 INTO v_CurrentStudents, v_MaxStudents
FROM classes
WHERE department = p_Department
AND course = p_Course;
- Рассчитаем текущее соотношение.
v_PercentFull := v_CurrentStudents / v_MaxStudents * 100;
 IF v_PercentFull = 100 THEN
RETURN 'Full';
ELSIF v_PercentFull > 80 THEN
RETURN 'Some Room';
ELSIF v_PercentFull > 60 THEN
RETURN 'More Room';
ELSIF v_PercerttFull > 0 THEN
RETURN 'Lots of Room';
ELSE
RETURN 'Empty';
END IF;
END Classlnfo;
```

При использовании оператора RETURN в функции с ним должно быть связано некоторое выражение. Однако RETURN можно применять и в процедуре. В этом случае аргументы, приводящие к немедленной передаче управления в вызывающую среду, не указываются. Текущие значения формальных параметров, описанных как OUT или IN OUT, присваиваются фактическим параметрам, и выполнение программы продолжается с оператора, следующего за вызовом процедуры.

Вызов процедур и функций

Команда EXECUTE запускает процедуру на выполнение. Вы также можете вызвать процедуру из анонимного блока, как показано ниже.

```
BEGIN
AddNewStudent('Vasya', 'Ivanov', 'Economics');
END;
```

Использование хранимых функций в SQL-операторах

Вызовы подпрограмм по сути своей реализуются в виде процедур, поэтому их нельзя выполнять в SQL-операторах. Однако если автономная или модульная функция отвечает определенным условиям, ее можно вызывать во время выполнения SQL-оператора. Эта возможность стала впервые доступна в PL/SQL версии 2.1 (Orac1e7 редакции 7.1), а в Oracle 10g она была расширена.

Функция, созданная пользователем, вызывается точно так же, как и встроенные функции (например, TOCHAR, UPPER или ADDMONTHS). В зависимости от версии Oracle и от области применения функции она должна отвечать определенным условиям. Эти условия определяются так называемыми уровнями строгости.

Уровни строгости

Для функций существуют четыре различных уровня строгости. Уровень строгости (purity level) определяет структуры данных, которые может считывать или модифицировать функция. Эти уровни представлены в таблице 6.1. В зависимости от уровня строгости функции на нее налагаются следующие ограничения:

- Функция, вызываемая из SQL-оператора, не может модифицировать таблицы базы данных (WNDS). (В Oracle 10g функция, вызываемая из оператора, отличного от SELECT, может модифицировать таблицы базы данных)
- Для того чтобы функция могла быть выполнена дистанционно (с помощью соединения баз данных) или параллельно, она не должна считывать или записывать значения модульных переменных (RNPS и WNPS).
- Функции, вызываемые из предложений SELECT, VALUES или SET, могут записывать модульные переменные. Функции во всех остальных предложениях должны иметь уровень строгости WNPS.
- Функция строга настолько, насколько строги вызываемые ею подпрограммы. Если функция вызывает хранимую процедуру, которая, к примеру, обновляет информацию (оператор UPDATE), то эта функция не имеет уровня строгости WNDS и, следовательно, не может быть использована в операторе SELECT.
- Независимо от уровня строгости хранимые функции PL/SQL нельзя вызывать из предложения ограничения CHECK команды CREATE TABLE или ALTER TABLE, а также использовать для указания значения по умолчанию для столбца, так как в этих ситуациях требуется, чтобы определения не изменялись.

Кроме приведенных ограничений, функция, созданная пользователем, должна отвечать также дополнительным требованиям, чтобы ее можно было вызывать из SQL-операторов. Заметим, что все встроенные функции тоже должны отвечать этим требованиям.

- Функция должна храниться в базе данных автономно или как часть модуля. Она не

должна быть локальной по отношению к другому блоку.

- Функция может принимать только параметры IN, но не IN OUT или OUT.
- Для формальных параметров должны использоваться только те типы, которые применяются в базе данных, но не типы PL/SQL, такие как BOOLEAN или RECORD.

Типы базы данных - это NUMBER, CHAR, VARCHAR2, ROWID, LONG, RAW, LONG RAW и DATE, а также новые типы Oracle9i и OraclelOg.

- Тип, возвращаемый функцией, также должен быть типом базы данных.
- Функция не должна заканчивать текущую транзакцию оператором COMMIT или ROLLBACK, либо выполнять откат к точке сохранения до своего выполнения.
- В функции не должны вызываться команды ALTER SESSION и ALTER SYSTEM.

В качестве примера рассмотрим функцию FullName, входным параметром которой является идентификатор студента и которая возвращает конкатенированные имя и фамилию.

```
CREATE OR REPLACE FUNCTION FullName (
p_StudentID students.ID%TYPE)
RETURN VARCHAR2 IS
v_Result VARCHAR2(1Q0);
BEGIN
SELECT first_name I I ' ' I I last_name
INTO v_Result
FROM students
WHERE ID = p_StudentID;
RETURN v_Result;
END FullName;
```

Функция FullName удовлетворяет всем ограничениям, поэтому ее можно вызвать из SQL-оператора:

```
SQL> SELECT ID, FullName(ID) "Full Name"
FROM students;
```

ID	**Full Name**
10000	Scott Smith
10001	Margaret Mason
10002	Joanne Junebug
10003	Manish Murgratroid
10004	Patrick Poll
10005	Timothy Taller
10006	Barbara Blues
10007	David Dinsmore
10008	Ester Elegant
10009	Rose Riznit
10010	Rita Razmataz
10011	Shay Shariatpanahy

```
12 rows selected.
SQL> INSERT INTO temp_table(char_col)
2 VALUES (FullName(10010));
1 row created.
```

Удаление процедур и функций.

Как и таблицы, процедуры и функции могут быть удалены. При выполнении этой операции процедура или функция удаляется из словаря данных. Синтаксис удаления процедуры выглядит следующим образом:

DROP PROCEDURE <имя_процедуры>;

А синтаксис удаления функции:

DROP FUNCTION <имя_фУнкДии>;

где <имя_процедуры> - имя существующей процедуры, а <имя_функции> -имя существующей функции. Например, при помощи следующего оператора удаляется процедура AddNewStudent:

```
DROP PROCEDURE AddNewStudent;
```

Если удаляемый объект является функцией, то нужно использовать оператор DROP FUNCTION, а если - процедурой, то DROP PROCEDURE.

DROP - это команда DDL, поэтому перед оператором DROP и после него неявно выполняется оператор COMMIT. Если подпрограмма не существует, оператор DROP порождает ошибку "ORA-4043 : Object does not exist".

Параметры функций и процедур

Как и в других языках программирования третьего поколения, можно создавать процедуры и функции с параметрами. Параметры могут быть разного вида, и их разрешается передавать по значению или по ссылке.

Пакеты

Модуль (package) - еще одно средство, пришедшее в PL/SQL из языка программирования Ada. Модуль - это конструкция PL/SQL, которая позволяет хранить связанные объекты в одном месте. Модуль состоит из двух частей: описания и тела. Они хранятся по отдельности в словаре данных. В отличие от процедур и функций, которые могут содержаться локально в блоке или храниться в базе данных, модули могут быть только хранимыми и никогда локальными. Помимо того, что модули позволяют группировать связанные объекты, они полезны еще и тем, что ограничений, налагаемых зависимостями, в них меньше, чем в хранимых подпрограммах. Кроме того, они имеют ряд свойств, улучшающих функционирование системы.

В сущности, модуль представляет собой именованный раздел объявлений. Все, что может входить в состав раздела объявлений блока, может входить и в модуль: процедуры, функции, курсоры, типы и переменные. Размещение их в модуле полезно тем, что это позволяет обращаться к ним из других блоков PL/SQL, поэтому в модулях можно описывать глобальные переменные PL/SQL (внутри одного сеанса работы с базой данных).

Тело пакета

Тело модуля (package body) - это объект словаря данных, хранящийся отдельно от заголовка модуля. Тело модуля нельзя скомпилировать, если ранее не был успешно скомпилирован заголовок. В теле содержится текст подпрограмм, предварительно объявленных в заголовке модуля. В нем могут находиться также дополнительные объявления, глобальные для тела модуля, но не видимые в его описании.

Вопросы к лекции

1. Что такое хранимая процедура?
2. Какова структура оператора создания процедуры?

3. Что такое функция PL/SQL?
4. Каким образом можно вызвать функции и процедуры?
5. В чем заключается назначение пакетов?

Список используемой литературы

Том Кайт «ORACLE для профессионалов. Книга 1. Архитектура и основные особенности.» изд. «APress» перев. «ДиаСофтЮП» 2003г 672 стр.

Jason Price «Oracle Database 1 lg SQL» «McGrowHill» 2007r. 656 стр.

Лекция 10 Функции для работы со строками. Числовые функции. Функции, оперирующие с датами.

План

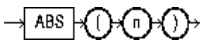
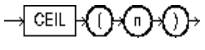
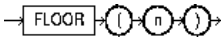
1. Числовые функции
2. Функции даты
3. Фразы GROUP BY и HAVING

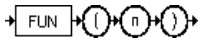
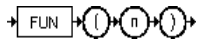
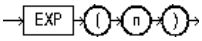
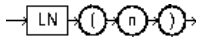
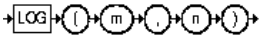
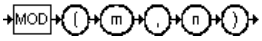
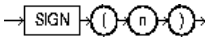
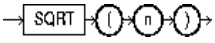
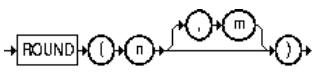
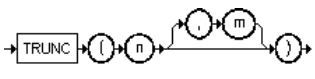
Функции SQL Oracle

Функции SQL являются встроенными в Oracle и доступны для использования в соответствующих фразах различных предложений SQL. Не смешивайте функции SQL с пользовательскими функциями, написанными на языке PL/SQL.
Если вы вызываете функцию со значением аргумента null, то она автоматически возвращает значение null. Единственными функциями, которые соответствуют этому правилу, являются `CONCAT`, `DECODE`, `DUMP`, `NVL` и `REPLACE`.
Имеется следующие две категории функций SQL:

Функции одной строки	Эти функции возвращают единственное значение для каждой строки таблицы. Эти функции могут использоваться в списке select (если предложение `SELECT` не содержит фразы `GROUP BY`) и во фразе `WHERE`.
Агрегатные функции	Они возвращают одно значение на основании совокупности строк таблицы. Агрегатные функции могут использоваться в списке select и фразе `HAVING`.

Числовые функции

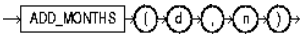
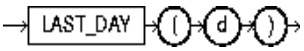
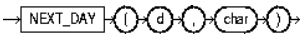
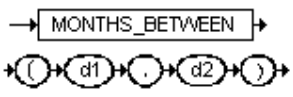
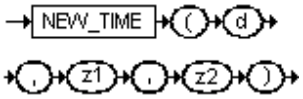
Функция	Синтаксис	Назначение	Пример
`ABS`		Возвращает абсолютное значение *n*.	`SELECT ABS(-15) "Absolute" FROM DUAL;`
`CEIL`		Возвращает наименьшее целое, которое больше или равно *n*.	`SELECT CEIL(15.7) "Ceil" FROM DUAL;`
`FLOOR`		Возвращает наибольшее целое, которое меньше или равно *n*.	`SELECT FLOOR(15.7) "Floor" FROM DUAL;`

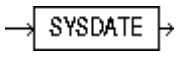
SIN, COS, TAN	 FUN – имя функции.	Возвращает sin, cos или tan n (угол в радианах).	SELECT SIN(30*3.1415/180) FROM DUAL;
SINH, COSH, TANH	 FUN- имя функции.	озвращает гиперболический sin, cos или tan *n*.	SELECT SINH(1) AS "Hyperbolic sine of 1" FROM DUAL;
EXP		Возвращает e в степени *n*, где e = 2.71828183 ...	SELECT EXP(4) AS "e to the 4th power" FROM DUAL;
LN		Возвращает натуральный логарифм *n*, где *n* больше или равно 0.	SELECT LN(95) AS "Natural log of 95" FROM DUAL;
LOG		Возвращает логарифм *n* по основанию *m*. Основание *m* может быть любым положительным числом, отличным от 0 или 1 и *n* могжет быть любым положительным числом.	SELECT LOG(10,100) AS "Log base 10 of 100" FROM DUAL;
MOD		Возвращает остаток от деления *m* на *n*. Возвращает *m*, если *n* = 0.	SELECT MOD(11,4) FROM DUAL;
POWER		Вовращает *m* в степени *n*. Основание *m* и показатель степени *n* могут быть любыми числами, однако, если *m* отрицательное, то *n* долдно быть целым числом.	SELECT POWER(3,2) FROM DUAL;
SIGN		Если *n*<0, то функция возвращает -1. Если *n*=0, то функция возвращает 0. Если *n*>0, то функция возвращает 1.	SELECT SIGN(-15) "Sign" FROM DUAL;
SQRT		Возвращает корень квадратный от *n*. Значение *n* не может быть отрицательным. SQRT возвращает в качестве результата число типа "real".	SELECT SQRT(26) FROM DUAL;
ROUND		Возвращает *n* округленное до *m* позиции справа от десятичной точки. Если *m* опущено, *n* округляется до позиции 0. *m* может быть отрицательным для округления до позиции слева от десятичной точки. *m* должно быть целым числом.	SELECT ROUND(15.193,1) AS "Round" FROM DUAL;
TRUNC		Возвращает *n*, отсеченное до позиции *m* справа от десятичной точки. Если *m* опщено, *n* отсекается до позиции 0. *m* может быть отрицательным. В этом случае отсечение (устанавливаются в нулевое значение) *m* цифр слева от	SELECT TRUNC(15.79,1) AS "Truncate" FROM DUAL;

		десятичной точки.	

Функции даты

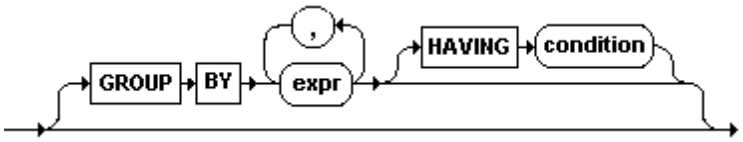
Функции даты оперируют со значениями типа DATE. Все функции даты возвращают значение типа DATE, за исключением MONTHS_BETWEEN, которая возвращает число.

Функ-ция	Синтаксис	Назначение	Пример
ADD_ MONTHS		Возвращает дату *d* плюс *n* месяцев. Аргумент *n* может быть любым целым числом. Если *d* последний день месяца или если результирующий месяц содержит меньше дней, чем компонента дня в *d*, то результат будет содержать последний день результирующего месяца. В противном случае результат будет иметь ту же компоненту дня, что и *d*.	`SELECT TO_CHAR( ADD_MONTHS(Hiredate 1), 'DD-MON-YYYY') FROM TEACHER WHERE Name = 'John';`
LAST_ DAY		Возвращает дату с последним днем месяца из *d*. Можно использовать эту функцию для определения сколько дней осталось в текущем месяце.	`SELECT SYSDATE, LAST_DAY(SYSDATE) LAST_DAY(SYSDATE) SYSDATE FROM DUAL;`
NEXT_ DAY		Возвращает дату того дня недели, который указан в *char* и который следует за датой *d*. Аргумент *char* должен быть названием дня недели, в полном виде или сокращенном, заданным согласно языка, используемого в вашем сеансе работы.	`SELECT NEXT_DAY( '15-MAR-98','TUESDAY') AS "NEXT DAY" FROM DUAL;`
MONTHS_ BETWEEN		Возвращает количество месяцев между датами *d1* и *d2*. Если дата *d1* позже, чем дата *d2*, то результат положителен; если раньше, то отрицателен. Если *d1* и *d2* содержат либо одну и ту же компоненту дня, либо указанные дни являются последними в месяце, то результат всегда целое число. В противном случае Oracle дробную часть месяцев с на основе 31-дневного месяца и с учетом разницы в компонентах времени дат *d1* и *d2*	`SELECT MONTHS_BETWEEN ( TO_DATE('28.10.2002 , 'DD.MM.YYYY'), TO_DATE('28.10.2002 ' 'DD.MM.YYYY')) FROM DUAL`
NEW_ TIME		Возвращает дату и время во временной зоне *z2*, когда дата и время временной зоне *z1* равна *d*. Аргументы *z1* и *z2* могут быть следующими текстовыми строками: AST, ADT - Atlantic Standard or Daylight Time	

		BST, BTD - Bering Standard or Daylight Time CST, CDT - Central Standard or Daylight Time EST, EDT - Eastern Standard or Daylight Time GMT - Greenwich Mean Time HST, HDT - Alaska-Hawaii Standard Time or Daylight Time MST, MDT - Mountain Standard or Daylight Time NST - Newfoundland Standard Time PST, PDT - Pacific Standard or Daylight Time YST, YDT - Yukon Standard or Daylight Time	
SYSDATE		Возвращает текущую дату и время. Аргументы отсутствуют. Эту функцию нельзя использовать в условии ограничения CHECK.	SELECT TO_CHAR(SYSDATE, 'DD-MM-YYYY HH24:MI:SS)' FROM DUAL;

Фразы GROUP BY и HAVING

Синтаксис:



Назначение

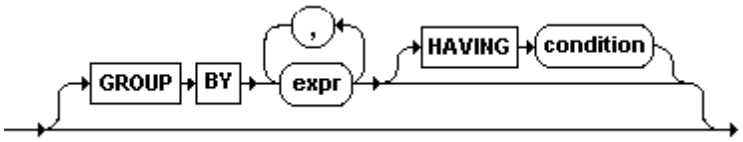
Фраза GROUP BY определяет столбец или список столбцов (выражение над столбцом или список выражений над столбцами), которые используются для группирования строк таблицы. Выражения фразы GROUP BY могут содержать любые столбцы из таблиц фразы FROM, не зависимо от того, появляетсяь ли столбец во списке select. Если запрос содержит фразу GROUP BY, то список select может содержать только:

- константы,
- выражения, включающие только агрегатные функции,
- выражения из фразы GROUP BY,
- выражения, которые включают упомянутые выше выражения.

Областью действия агрегатной функции являются все строки каждой группы. Таким образом, SQL применяет агрегатные функции в списке select к каждой группе строк и возвращает единственную результирующую строку для каждой группы. То есть, каждая сформированная группа порождает одну результирующую строку.
Назначение фразы HAVING – определить условие выбора на группах строк. Она ограничивает группы строк теми, на которых указанное условие равно TRUE..
Указывайте GROUP BY и HAVING фразы после фразы WHERE..
См. также описание синтаксиса для *выражений* в ПРиложении Лаб.4 и синтаксис описания *условий* в Приложении Лаб.2 .

Фраза ORDER BY

Синтаксис:



Назначение

Фраза `ORDER BY` позволяет упорядочить строки результата запроса. При отсутствии этой фразы нет ни какой гарантии, что будучи выполненным дважды, запрос выдаст строки результата в одном и том же порядке.

- *expr* – означает, что строки упорядочиваются согласно значения выражения *expr*. Выражение может базироваться на столбцах из списка **select** или на столбцах из таблиц фразы **FROM.**
- *position* – упорядочивает строки на основании значений выражения в указанной позиции списка select; *position* должно быть целым числом.
- *c_alias* – упорядочивает строки на основании значения столбца (выражения), который имеет указанный алиас в списке select.
- `ASC` и `DESC` указывают порядок сортировки: по возрастанию или убыванию. `ASC` – знчение по умолчанию.

Можно указать множество выражений во фразе `ORDER BY`. В этом случае производится многоуровневая сортировка. Oracle располагает значения null в конце при сортировке в порядке возрастания и в начале при сортировке в порядке убывания.

Ограничения:

- Если вы указали фразу `DISTINCT`, то фраза `ORDER BY` должна ссылаться только на столбцы из списка select.
- Если также задана фраза `GROUP BY`, то фраза `ORDER BY` ограничивается следующими выражениями:
  - Константы
  - Агрегатные функции
  - Выражения, идентичные тем, что используются во фразе group by
  - Выражения, включающие приведенные выше выражения, которые вычисляют одно и то же значение для всх строк в группе.

Фразы GROUP BY и HAVING

GROUP BY и агрегатные функции в списке select.

Пример. Сколько кафедр в каждом из корпусов:

```
SELECT Building, COUNT(*)
FROM   DEPARTMENT
GROUP BY  Building;
```

Example. Какова сумма зарлпты (Salary+Commission) по всем имеющимся должностям:

```
SELECT Post, SUM(Salary + Commission)
FROM   TEACHER
GROUP BY Post;
```

Группирование и фраза `WHERE`. Если запрос содержит фразы `WHERE` и `GROUP BY`, то фраза`WHERE` обрабатывается первой, а затем применяется группирование.

Пример. Для каждого корпуса подсчитать количество аудиторий вместимостью более 50:

```
SELECT Building, COUNT(*)
FROM   ROOM
WHERE  Seats > 50
GROUP BY Building;
```

Пример. Для корпусов 5, 7 и 12 подсчитать количество аудиторий вместимостью более 50:

```
SELECT Building, COUNT(*)
FROM   ROOM
WHERE  UPPER(Building) IN ('5', '7', '12') AND Seats > 50
```

```
GROUP BY Building;
```

Группирование по многим столбцам. Можно группировать по многим столбцам.
Пример. Для каждой недели и дня недели подсчитать количество лекций типа “lab”:

```
SELECT Day, Week, COUNT(*)
FROM   LECTURE
WHERE  UPPER(Type) = 'LAB'
GROUP BY Week, Day;
```

Группирование и соединение различных таблиц. Имеется возможность соединить две или более таблиці и зптем произвести группирование по соединенной таблице.
Пример. По каждому факультету выдать количество кафедр:

```
SELECT f.Name, COUNT(*)
FROM   FACULTY f, DEPARTMENT d
WHERE  f.FacNo = d.FacNo
GROUP BY f.Name;
```

Пример. По каждому факультету выдать количество профессоров:

```
SELECT f.Name, COUNT(*)
FROM   FACULTY f, DEPARTMENT d, TEACHER t
WHERE  f.FacNo = d.FacNo AND d.DepNo = t.DepNo AND
UPPER(t.Post) = 'PROFESSOR'
GROUP BY f.Name;
```

Пример. Для каждой кафедры каждого факультета подсчитать количество профессоров:

```
SELECT f.Name, d.Name, COUNT(*)
FROM   FACULTY f, DEPARTMENT d, TEACHER t
WHERE  f.FacNo = d.FacNo AND d.DepNo = t.DepNo AND
UPPER(t.Post) = 'PROFESSOR'
GROUP BY f.Name, d.Name;
```

Использование фразы HAVING. Она задает условие на группу. Фразы `HAVING` обычно содержат агрегатную функцию.
Example. Вывести корпуса, в которых суммарное количество мест в аудиториях меньше 1000:

```
SELECT Building
FROM   ROOM
GROUP BY Building
HAVING SUM(Seats) < 1000;
```

Фраза ORDER BY

Она испольуется для упорядочения результатов запроса. Можно упорядочивать

- по любому столбцу таблицы,
- по выражению над столбцами,
- по списку столбцов или выражений.

Упорядочение по столбцу из списка select.
Пример. Выдать назвыание факультета и его декана, упорядочив результат по факультетам:

```
SELECT Name, Dean
FROM   FACULTY
ORDER BY  Name;
```

Упорядочение по столбцу таблицы. Можно упорядочивать по столбцу таблицы, даже если он не присутствует в списке select. Эта возможность не поддерживается стандартом ANSI, но допустима в SQL Oracle.
Пример. Выдать имена преподавателей и их ставки, упорядочив результат по должностям:

```
SELECT Name, Salary
FROM   TEACHER
ORDER BY  Post;
```

Упорядочение по списку столбцов. Используя список столбцов можно произвести многоуровневое упорядочение.
Пример. По каждой группе вывести ее номер, курс и количество студентов. Упорядочить результат по курсу и номеру группы:

```
SELECT Course, Num, Quantity
FROM   SGROUP
ORDER BY  Course, Num;
```

Упорядочение по выражению над столбцами. Можно упорядочивать по выражению над столбцами.
Example. Вывести имя ставку и надбавку преподавателей. Упорядочить результат по выражению Salary+Commission:

```
SELECT Name, Salary, Commission
FROM   TEACHER
ORDER BY  Salary + Commission ASC;
```

Использование порядкового номера столбца в списке select. Можно использовать порядковый номер столбца в списке select для ссылки на него во фразе `ORDER BY`. Это может оказаться удобным, когда список select содержит выражения.
Пример. Вывести имя преподавателя и его зарплату (Salary+Commission). Упорядочить результат по выражению Salary+Commission в порядке убывания:

```
SELECT Name, Salary + Commission
FROM   TEACHER
ORDER BY  2 DESC;
```

Использование алиаса столбца из списка select. Вы можете воспользоваться алиасом столбца из списка select для ссылки на него во фразе `ORDER BY`.
Пример. Вывести имена преподавателей и даты их поступления на работу. Если дата не определена, то вывести строку “not defined”. Упорядочить результат по дате в убывающем порядке:

```
SELECT Name, NVL(TO_CHAR(hiredate,'yyyy-mm-dd'),'not defined')
AS Teacher_hiredate
FROM   TEACHER
ORDER BY  Teacher_hiredate DESC;
```

Соединение таблиц и упорядочение. Если вы соединяете таблицы, то можно упорядочивать по любому столбцу соединенной таблицы.
Пример. Вывести имна преподавателей и их зарплаты факультета инфоматики. Упорядочить результат по зарплате в убывающем порядке.

```
SELECT t.Name, Salary + Commission
FROM   FACULTY f, DEPARTMENT d, TEACHER t
WHERE  f.FacNo = d.FacNo AND d.DepNo = t.DepNo AND
UPPER(f.Name) = 'INFORMATICS'
ORDER BY  2 DESC;
```

Группирование и упорядочение. Для упорядочение групп необходимо удовлетворить ограничения, описанные в разделе “2.3. Фраза ORDER BY”.
Пример. Вывести количество студентов на каждом курсе. Расположить результат в порядке возрастания курса.

```
SELECT Course, SUM( Quantity ) FROM        SGROUP
GROUP BY Course
ORDER BY  Course ASC;
```

Вопросы к лекции

1. Какие типы фукций SQL вы знаете?
2. Какая область действия агрегатных функций при наличии и отсутствии фразы `GROUP BY`?
3. Как SQL Oracle оперирует с датами?
4. Какие выражения может содержать список select, если присутствует фраза `GROUP BY`?
5. Каковы цели фраз `GROUP BY` и `HAVING`?

Список используемой литературы

Том Кайт «ORACLE для профессионалов. Книга 1. Архитектура и основные особенности.» изд. «APress» перев. «ДиаСофтЮП» 2003г.

Jason Price «Oracle Database 1 lg SQL» «McGrowHill» 2007r.

Лекция №7 Создание пользовательских процедур и функций. Пакеты. Триггеры базы данных.

План

1. Триггеры
2. Типы триггеров
3. Создание триггеров
4. Создание замещающих триггеров

Триггеры

Триггер — это процедура PL/SQL, которая выполняется автоматически, когда происходит некоторое заданное событие, называемое триггерным событием (triggering event).

Триггеры похожи на процедуры и функции тем, что также являются именованными блоками PL/SQL и имеют раздел объявлений, выполняемый раздел и раздел обработки исключительных ситуаций. Подобно модулям, триггеры хранятся как автономные объекты в базе данных и не могут храниться локально в блоке или модуле. Процедура вызывается явным образом из другого блока, при вызове ей могут передаваться различные аргументы. Триггер же выполняется неявно всякий раз, когда происходит запускающее его событие, и триггер не имеет аргументов. Акт выполнения триггера называется его активизацией (firing). Событием, запускающим триггер, является операция DML (INSERT, UPDATE или DELETE), выполняемая над таблицей или представлением базы данных. В Oracle 10g эти функции расширены: триггер может срабатывать на системное событие,

например на запуск или останов базы данных, а также на определенные виды операций DDL.

Вы можете писать триггеры, срабатывающие в одной из следующих ситуаций:

- Применение оператора DML к определенному объекту схемы
- Выполнение оператора DDL внутри схемы или базы данных
- Вход пользователя в систему или выход его из системы, ошибка сервера, запуск базы данных или останов экземпляра.

Между триггерами и процедурами PL/SQL есть три различия:

- Триггеры нельзя вызывать из кода программы. Oracle вызывает их автоматически в ответ на определенное событие.
- Триггеры не имеют списка параметров.
- Спецификация триггера немного отличается от спецификации процедуры.

Сходство между триггерами и процедурами состоит в следующем:

- Тело триггера выглядит точно так же, как и тело процедуры — и то, и другое суть базовые блоки PL/SQL.
- Триггер не возвращает никаких значений.
- Триггеры можно использовать для выполнения разнообразных задач.
- Триггеры автоматически генерируют производные значения столбцов. Например, при обработке каждого нового заказа комиссионные торгового агента могут рассчитываться и вводиться в соответствующую таблицу автоматически.
- Триггеры помогают предотвращать неверные транзакции. Например, триггер может воспрепятствовать увеличению зарплаты служащего на большую величину, чем предусмотрено бюджетом. Заметьте, что это нельзя сделать с помощью контрольных ограничений, поскольку контрольное ограничение имеет дело только со значениями текущей строки, а следовательно, не может определять общий размер бюджета.
- Триггеры можно использовать для реализации сложных процедур авторизации. Например, значение зарплаты могут изменять только руководители и только для людей, находящихся в их подчинении.
- Триггеры используются для реализации сложных бизнес-правил. Например, триггер может проверять, доступен ли заказанный товар к дате отгрузки, и если нет, то автоматически размещать заказ на пополнение запасов, чтобы на момент отгрузки на складе находилось достаточное количество этого товара.
- Триггеры могут обеспечивать прозрачное протоколирование событий, например, отслеживать, сколько раз отдельный продавец обращался к таблице с инвентарной ведомостью.
- Триггеры могут обеспечивать сложный аудит. Например, с помощью триггера легко выяснить, какое среднее число обращений к базе данных требуется данному продавцу для обработки одного заказа.
- Триггеры могут собирать статистику доступа к таблицам. Например, триггер позволяет зафиксировать, сколько раз в течение дня выполнялись запросы на "Small Widget" к таблице с каталогом товаров.

Как вы уже, вероятно, поняли, триггеры представляют собой весьма мощные инструменты. Однако их следует использовать аккуратно, поскольку триггер может срабатывать при каждом обращении к таблице, тем самым увеличивая нагрузку на сервер базы данных. Разумное практическое правило состоит в том, чтобы использовать триггер только для действий, которые нельзя выполнить другими средствами. Например, если есть возможность создать одно или несколько ограничений, выполняющих работу некоторого триггера, следует использовать эти ограничения.

Типы триггеров

Триггеры DML

Триггер DML активизируется оператором DML, и тип триггера определяется типом этого оператора. Триггеры DML задаются для операций ввода, обновления и удаления информации (INSERT, UPDATE, DELETE). Они активизируются до или после операции, на уровне строки или оператора.

Триггеры замещения

В Oracle l0g предлагается еще один вид триггеров. Триггеры замещения (instead of) можно создавать только для представлений (либо объектных, либо реляционных). В отличие от триггеров DML, которые выполняются в дополнение к операторам DML, триггеры замещения выполняются вместо операторов DML, вызывающих их срабатывание. Триггеры замещения должны быть строковыми триггерами. Для примера рассмотрим представление classes_rooms:

```
CREATE OR REPLACE VIEW classes_rooms AS
SELECT department, course, building, room_number
FROM rooms, classes
WHERE rooms.room_id = classes.room_id;
```

Системные триггеры

В Oracle 10g и выше существует третий тип триггеров. Системный триггер активизируется не на операцию DML, выполняемую над таблицей, а на системное событие, например, на запуск или останов базы данных. Системные триггеры срабатывают и на операции DDL, такие как создание таблицы. Предположим, что необходимо регистрировать моменты создания объектов словаря данных. Это можно сделать, создав следующую таблицу:

```
CREATE TABLE ddl_creations (
user_id VARCHAR2(30),
object_type VARCHAR2(20),
object_name VARCHAR2(30),
object_owner VARCHAR2(30),
creation_date DATE);
```

Создание триггеров

Вне зависимости от типа все триггеры создаются одинаково. Общий синтаксис создания триггера таков:

```
CREATE [OR REPLACE] TRIGGER <имя_триггера>
{BEFORE | AFTER | INSTEAD OF] <активизирующее_событие>
<ссылочное_предложение> [WHEN
<условие_срабатывания>] [FOR
EACH ROW]"
<тело_триггера>;
```

где <имя_триггера> — это имя триггера, <активизирующее_событие> указывает событие, которое запускает триггер (может содержать конкретную таблицу или представление), а <тело_триггера> — основной программный текст триггера. <Ссылочное_предложение> используется для ссылки на данные в модифицируемой в конкретный момент строке с помощью другого имени. Если присутствует <условие_срабатывания> в конструкции WHEN (когда), то оно оценивается первым. Тело триггера выполняется только в том случае, если это условие истинно.

Создание замещающих триггеров

В отличие от триггеров DML, срабатывающих как дополнение к операции INSERT, UPDATE или DELETE (или до, или после них), замещающие триггеры активизируются вместо операций DML. К тому же замещающие триггеры создаются только для представлений, в то время как триггеры DML - для таблиц. Замещающие триггеры используются в двух случаях:

- Для того чтобы сделать представление модифицируемым, если иначе это сделать нельзя.
- Для модификации столбцов в столбце вложенной таблицы представления.

Модифицируемые и немодифицируемые представления

Модифицируемым (modifiable) называется такое представление, по отношению к которому можно выполнить оператор DML. Как правило, представление является модифицируемым, если оно не содержит:

- Операций над множествами (UNION, UNION ALL, MINUS)
- Функций агрегирования (SUM, AVG и т.д.)
- Конструкций GROUP BY, CONNECT BY и START WITH
- Операции DISTINCT
- Соединений

Вопросы к лекции

1.Что такое триггер?
2. Какие типы триггеров существуют?
3. Как создаются триггеры?
4. Какова роль системных триггеров?

Лекция №12 **Стандартные пакеты ORACLE. Сохраненные процедуры. Динамический SQL.**

План

1. Создание операторов динамического SQL
2. Динамические параметры
3. Динамические курсоры
4. Сохраненные процедуры

Создание операторов динамического SQL

Операторы динамического SQL - в отличие от операторов встроенного SQL - формируются не на этапе компиляции, а на этапе выполнения приложения.
Операторы динамического SQL формируются как текстовые переменные.

Например:

```
Stmt1:='SELECT * FROM tbl1';
```

Для динамического формирования оператора можно выполнять последовательное объединение строк.

Операторы динамического SQL можно использовать:

- однократно, производя за один шаг компиляцию и выполнение оператора. Будем называть такое применение одношаговым интерфейсом;
- многократно, разделяя процесс компиляции оператора, на котором строится план выполнения, и процесс непосредственного выполнения оператора. Будем называть такое применение многошаговым интерфейсом.

Одношаговый интерфейс

одношаговый интерфейс реализуется SQL-оператором EXECUTE IMMEDIATE, который имеет в стандарте SQL-92 следующее формальное описание:

```
EXECUTE IMMEDIATE :variable;
```

На оператор, указываемый переменной (variable), накладываются следующие ограничения:

- оператор не может использовать INTO-переменные;
- оператор не может использовать переменные связи.

Следующий пример иллюстрирует применение динамического SQL с одношаговым интерфейсом:

```
stmt_str := 'INSERT INTO ' || table_name ||
        ' values (:f1, :f2, :f3)';
EXEC SQL EXECUTE IMMEDIATE :stmt_str;
```

Многошаговый интерфейс

Оператор EXECUTE IMMEDIATE удобен для одноразового выполнения, но при необходимости неоднократного выполнения, например в цикле одного и того же оператора, но с различными параметрами, более эффективно использовать многошаговый интерфейс, реализуемый операторами PREPARE и EXECUTE.

При выполнении оператора PREPARE, указываемый им SQL-оператор передается в СУБД. Далее выполняется синтаксический разбор оператора и строится план выполнения. После этого при каждом выполнении оператора EXECUTE используется уже "откомпилированный" SQL-оператор, что значительно повышает производительность. Дополнительно при выполнении оператора EXECUTE на сервер передаются значения переменных связи (если они есть), используемые, в частности, для вычисления предиката фразы WHERE.

Оператор PREPARE имеет в стандарте SQL-92 следующее формальное описание:

```
PREPARE [ GLOBAL | LOCAL ] operator_sql FROM string_variable;
```

Параметр operator_sql определяет идентификатор SQL-оператора, указываемый далее для выполнения в оператореEXECUTE или для включения в курсор в операторах ALLOCATE CURSOR или DECLARE CURSOR.

Параметр string_variable указывает строку, содержащую динамически сформированный текст SQL-оператора.

Например:

```
stmt_str := 'INSERT INTO ' || table_name ||
        ' values (:f1, :f2, :f3)';
EXEC SQL PREPARE GLOBAL stmt1 FROM :stmt_str;
```

Фразы GLOBAL и LOCAL определяют область видимости оператора: GLOBAL указывает, что оператор с данным идентификатором доступен всем процессам данного сеанса работы с СУБД, а LOCAL ограничивает доступ рамками данного выполняемого модуля (значение по умолчанию).

Если создаются два одноименных оператора, но один как GLOBAL, а другой - как LOCAL, то СУБД создает два отдельных плана выполнения как для разных операторов. В противном случае при компиляции оператора с уже существующим именем просто строится новый план выполнения оператора.
Для освобождения подготовленного SQL-оператора используется оператор DEALLOCATE PREPARE, который освобождает все ресурсы, занимаемые подготовленным SQL-оператором.

Например:

```
EXEC SQL DEALLOCATE PREPARE GLOBAL stmt1;
```

Для выполнения откомпилированного SQL-оператора используется оператор EXECUTE, который в стандарте SQL-92 имеет следующее формальное описание:

```
EXECUTE [ GLOBAL | LOCAL ] operator_sql
  [ INTO {variable .,:}
    | { SQL DESCRIPTOR [ GLOBAL | LOCAL ]
       descriptor_name } ]
  [ USING {variable .,:}
    | { SQL DESCRIPTOR [ GLOBAL | LOCAL ]
       descriptor_name } ]
```

Фраза INTO указывается в том случае, если выполняемый SQL-оператор представляет собой запрос, возвращающий одну строку.

Динамические параметры

Значения динамических параметров передаются на сервер каждый раз при выполнении откомпилированного SQL-оператора. Динамическими параметрами могут быть как переменные связи, так и INTO-переменные.

динамические параметры можно использовать как во встроенном SQL, так и в динамическом SQL.
динамические параметры задаются в тексте SQL-оператора символами "знак вопроса". Стандарт не определяет максимально допустимое число динамических параметров. Как правило, СУБД могут иметь ограничения только на размер вводимого SQL-оператора.

Например:

```
stmt_str :='INSERT INTO tbl1
            VALUES (?, ?, ?)';
EXEC SQL PREPARE stmt2 FROM :stmt_str;
```

При выполнении данного откомпилированного оператора вместо динамических параметров значения будут подставляться в порядке, указанном в SQL-операторе EXECUTE или в области SQL-дескриптора.

Список значений для динамических параметров может быть указан:

- фразой USING оператора EXECUTE - для динамических параметров, не указываемых фразой INTO откомпилированного оператора;
- фразой INTO оператора EXECUTE - для динамических параметров, указанных во фразе INTO откомпилированного оператора.

Например:

```
stmt_str1 :='INSERT INTO tbl1 (f1,f2,f3)
```

```
            VALUES (?, ?, ?)';
EXEC SQL PREPARE stmt2 FROM :stmt_str1;
EXEC SQL EXECUTE stmt2 USING :f1, :f2, :f3;
```

Значение переменных f1, f2 и f3 основного языка программирования будут переданы на сервер для выполнения откомпилированного оператора с идентификатором stmt2.

Возможен вариант, когда откомпилированный оператор содержит динамические параметры и во фразе INTO операторaSELECT, и в предикате.

Например:

```
stmt_str2 :='SELECT f1, f2, f3
        FROM tbl1 INTO ?, ?, ?
        WHERE f2= ?';
EXEC SQL PREPARE stmt3 FROM :stmt_str2;
EXEC SQL EXECUTE stmt3 INTO :f1, :f2, :f3
      USING :f4;
```

Переменные f1, f2 и f3 основного языка программирования будут использованы как INTO-переменные, а значение переменной f4 будет передано на сервер для выполнения откомпилированного оператора с идентификатором stmt3.

Динамические курсоры

В динамическом SQL можно использовать не только курсоры встроенного SQL (создаваемые статически оператором DECLARE CURSOR), но и два дополнительных типа курсоров:

- объявляемые курсоры (declared cursors), создаваемые как DECLARE CURSOR;
- размещаемые курсоры (allocated cursors), создаваемые как ALLOCATE CURSOR .

Этот тип курсоров иногда называется выделенными курсорами.

Объявляемые и размещаемые курсоры могут иметь динамические параметры.

Для создания курсоров используются следующие операторы:

- ALLOCATE CURSOR, в котором курсор указывается идентификатором переменной, описывающей SQL-оператор;
- DECLARE CURSOR, в котором курсор указывается идентификатором откомпилированного SQL-оператора.

Например:

```
str1:='INSERT INTO tbl1 VALUES (1,10) ';
EXEC SQL ALLOCATE cur1 CURSOR FOR :str1;
EXEC SQL PREPARE stmt1 FROM :str1;
EXEC SQL DECLARE cur2 CURSOR FOR stmt1;
```

Открываются и закрываются динамические курсоры, как и статически создаваемые, операторами OPEN и CLOSE. Но при открытии курсора, имеющего динамические параметры, должна быть указана фраза USING.

Например:

```
str1:='SELECT f2 FROM tbl1 WHERE f1=? ';
EXEC SQL ALLOCATE cur1 CURSOR FOR :str1;
EXEC SQL OPEN cur1 USING :f2;
EXEC SQL FETCH cur1 INTO :f1;
```

Во фразе INTO оператора FETCH может быть указан как список INTO-переменных, так и SQL-дескриптор.

Сохраненные процедуры

Сохраненная процедура — это набор операторов SQL, обычно называемый *функцией* или *подпрограммой,* созданный программистом для удобства использования в программах (сохраненную процедуру использовать проще, чем каждый раз записывать весь набор входящих в нее операторов SQL). Кроме того, сохраненные процедуры можно вкладывать одну в другую, т. е. одни сохраненные процедуры могут вызывать другие, последние, в свою очередь, тоже могут вызывать сохраненные процедуры и т. д.Возможность сохранять процедуры — основа процедурного программирования. Команды SQL (CREATE TABLE, INSERT, UPDATE, SELECT'и т.д.) дают вам возможность сообщить базе данных, что делать, но не как делать. Посредством составления процедур вы получаете возможность сообщить ядру базы данных, каким образом следует обрабатывать данные.

Сохраненная процедура — это набор из одного или нескольких операторов SQL или функций, сохраненный в базе данных в откомпилированном виде, готовом для выполнения пользователем базы данных. *Сохраненная функция* является сохраненной процедурой, предполагающей в результате своего выполнения возврат некоторого значения. Функции вызываются процедурами. При вызове функции ей, как процедуре, могут передаваться параметры, затем функция вычисляет некоторое значение и возвращает его вызывающей процедуре для дальнейшего использования.При сохранении процедуры в базе данных сохраняются также и все входящие в эту процедуру подпрограммы и функции (использующие SQL). Все эти сохраняемые процедуры предварительно анализируются и сохраняются в виде, готовом для немедленного использования по команде, инициируемой пользователем.

В Oracle синтаксис оператора следующий.

CREATE [OR REPLACE] PROCEDURE *ИМЯ__ПРОЦЕДУРЫ*

[(АРГУМЕНТ [{IN | OUT | IN OUT)] *ТИП,*

АРГУМЕНТ [{IN 1 OUT | IN OUT}] *ТИП]]* {IS I AS)

ТЕЛО_ПРОЦЕДУРЫ

Вот пример оператора, создающего достаточно простую процедуру.

```
CREATE PROCEDURE NEW_PRODUCT
(PROD_ID IN VARCHAR2, PROD_DESC IN VARCHAR2, COST IN NUMBER)
AS
BEGIN
INSERT INTO PRODUCTS_TBL
VALUES (PRODJT.D, PROD_DESC, COST);
COMMIT;
END;
```

Процедура создана.

Преимущества использования процедур

Использовать сохраненные ранее процедуры удобнее, чем отдельные операторы SQL по целому ряду причин. Некоторые из этих причин перечислены ниже.

• Операторы сохраненной процедуры уже сохранены в базе данных.
• Операторы сохраненной процедуры уже проверены и находятся в готовом для использования
• Возможность сохранения процедур позволяет использовать модульное программирование
• Сохраненные процедуры могут вызывать другие процедуры и функции.
• Сохраненные процедуры могут вызываться другими программами.
• При использовании сохраненных процедур результат ответ от базы данных обычно получаетсябыстрее
• Использовать процедуры очень просто.

Вопросы к лекции

1. Как создаются операторы динамического SQL?
2. Что такое одношаговый и многошаговый интерфейс?
3. Что такое сохраненная процедура?
4. Каковы преимущества использования процедур?

Лекция № 13 **Файловый вывод/ввод. Управление заданиями. Управление объектов LOB. Использование PL /SQL-функций в SQL- выражениях.**

План

1. Файловый вывод/ввод
2. Компоненты и типы данных LOB
3. Использование пакета DBMS_LOB
4. Основные правила работы с DBMS_LOB

Файловый вывод/ввод

Большинство программ, работающих с БД, позволяют сохранять выборки в различных форматах, но не все позволяют сохранить в виде простого текстового файла. Например, была ситуация, когда в БД была необходимая информация о сотрудниках (более тысячи человек) для одной стандартной формы. А формат файла, описывающий эту форму, оказался текстовым. Поэтому проблема решилась выборкой в файл и вставкой его содержимого в файл формы. А соответствующий отдел был избавлен от лишней работы. Для конкретности примера, воспользуемся следующей таблицей с данными.

```
-- таблица цветов
create table colors(id integer,cname varchar(45), constraint
pk_colors primary key (id)
);
insert into colors values(1,'красный');
insert into colors values(2,'синий');
insert into colors values(3,'зеленый');
insert into colors values(4,'белый');
insert into colors values(5,'черный');
```

```
commit;
```

В Oracle вывод в файл реализуется с помощью команды SQL plus *spool*. Она служит как для начала вывода в файл, так и для остановки. По умолчанию расширение файла lst.

```
-- открываем вывод в файл
spool c:\\myfile;

-- выборка
select t.id || ' ' || t.cname from colors t;

-- закрываем вывод в файл
spool off;
```

Так как это не SQL команда, то ее нельзя использовать внутри PL/SQL блока, но можно поместить блок между этими командами. По этой же причине, чтобы выполнить этот пример в PL/SQL developer, нужно открыть Command window.

На некоторых операционных системах поддерживается дополнительный параметр out, позволяющий сразу распечатать выборку на принтере установленном по умолчанию.

```
-- открываем вывод в файл
spool c:\myfile;

-- выборка
select t.id || ' ' || t.cname from colors t;

-- закрываем вывод файл и распечатываем
spool out;
```

Что такое LOB

Large Objects (LOB) это тип данных используемый для хранения больших объектов – различные форматы текстов, изображения, видео, звуковые файлы. Использование LOB для хранения данных позволяет эффективно манипулировать данными в приложении.

Компоненты LOB

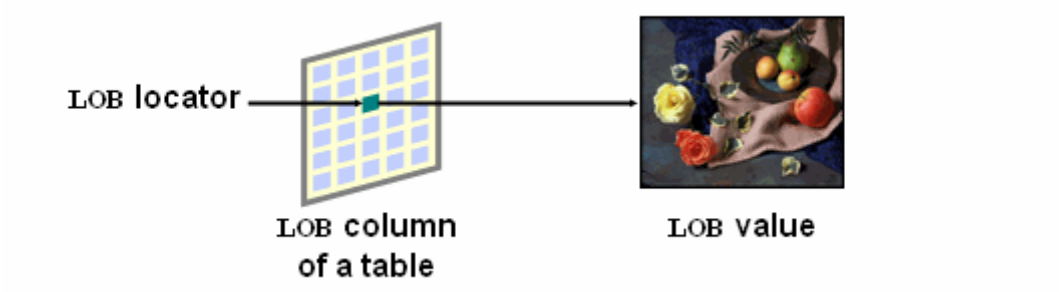
LOB состоит из локатора и значения.

Локатор – это внутренний указатель на фактическое значение большого объекта.

Значение – это реальное содержимое объекта.

LOB хранит локатор в таблице и данные в различных сегментах, за исключением случая, когда размер данных меньше 4000 байт.

Максимальный размер LOB составляет (**4 GB** (4 294 967 295 байт) – 1) * (значение CHUNK parameter of LOB storage); размер может достигать до 128 терабайт.



Типы данных LOB

SQL Datatype	Описание
BLOB	Двоичный большой объект (Binary Large OBject) Хранит данные в двоичном формата, используется для хранения изображений, аудио и видео, а также компилированного программного кода
CLOB	Символьный большой объект (Character Large Object) Хранит текстовые данные в формате БД
NCLOB	Национальный символьный большой объект (National Character Set Large Object) Хранит текстовые данные в национальной кодировке.
BFILE	Внешний двоичный файл (External Binary File) Файл хранящийся вне базы данных, как файл операционной системы, но доступный из таблиц БД. BFILEs имеет доступ только для чтения. Когда LOB удаляется, Oracle сервер не удаляет сам файл. BFILE поддерживает только случайное(не последовательное) чтение, не участвует в транзакции.

Виды LOB

Как внутренние, так и внешние большие объекты могут быть использованы как столбцы таблицы, переменные в pl/sql, атрибуты объектов.

Internal LOB

Внутренние большие объекты – хранятся в табличных пространствах БД, поддерживаются следующие типы данных: BLOB, CLOB, and NCLOB.

Persistent and Temporary LOB

Внутренние большие объекты могут быть временными или постоянными.
Постоянные LOB существуют в строках таблицы БД.
Временные LOB создается для использования только в пределах локального приложения.
Временный экземпляр становится постоянным если его вставить в строку таблицы.

Свойства Temporary LOB

- Данные хранятся не в таблице, а во временном табличном пространстве
- Временные LOB быстрее чем постояннные, так как не генерируется redo и rollback данные
- Используется только в сесии. По окончанию сесии удаляется
- Для создания временного LOB используется DBMS_LOB.CREATETEMPORARY

External LOB

Внешние большие объекты – вид данных, который хранится в файлах операционной системы, вне табличных пространств БД. Используется тип данных – BFILE. BFILE может быть только внешним.

Использование пакета DBMS_LOB

DBMS_LOB предоставляет методы манипулирования внутренними и внешними LOBами.

Методы DBMS_LOB можно условно разделить на два типа mutators и observers:

- **mutators** – могут изменять LOB : APPEND, COPY, ERASE, TRIM, WRITE, FILECLOSE, FILECLOSEALL, and FILEOPEN.
- **observers** – могу читать LOB: COMPARE, FILEGETNAME, INSTR, GETLENGTH, READ, SUBSTR, FILEEXISTS, and FILEISOPEN.

Для инициализации LOB локаторов используются следующие функции:

- **EMPTY_CLOB()** – функция возвращает LOB локатор для CLOB колонки
- **EMPTY_BLOB()** – функция возвращает LOB локатор для BLOB колонки

CLOB и BLOB колонки, так же могут быть инициализированы с помощью символьной или raw строки, если длина не превыщает 4000 байт.

Основные правила работы с DBMS_LOB

- Нельзя использовать в качестве параметра пустой локатор или локатор, имеющий значение null
- Прежде чем осуществить доступ к внешнему LOBу, ассоциированный с ним файл должен быть открыт
- Перед завершение pl/sql блока нужно закрывать файл объекта bfile
- Прежде чем выполнять запись во внутренний большой объект, необходимо заблокировать строку, которая содержит столбец этого объекта. Это можно сделать явными блокировками или sql операторами: select for update, update, insert.
- Правила работы с согласованными по чтению и обновленными локаторами применимы ко всем процедурам и функциям данного пакета.

Вопросы к лекции

1. Что значит файловый вывод/ввод в ORACLE?
2. Виды и типы данных LOB.
3. Где используется пакет DBMS_LOB?
4. Основные правила работы с DBMS_LOB

Список используемой литературы

Скотт Урман «ORACLE 9i Программирование на языке PL/SQL.» изд. «ORACLE Press» перев. «Лори» 2004г.

Лекция № 14 **Средства обеспечения целостности данных. Транзакции и их роль в СУБД. Начало и завершение транзакций.**

План

1. Понятие транзакции
2. Свойства транзакций
3. Некоторые особенности выполнения транзакций в ORACLE

Понятие транзакции

Транзакция – это действие или серия действий, выполняемых одним пользователем или прикладной программой, которые осуществляют доступ или изменение содержимого БД. Транзакция является логической единицей работы, выполняемой в БД. Она может быть представлена отдельной программой, являться частью алгоритма программы или даже отдельной командой (например, insert, update).

/*Например, изменение заработной платы сотруднику с некоторым табельным номером. Пользователь выполняет операцию

update sotrudnik set z.p. = z.p.*1.1 where tabnomer= 23;

Транзакция же будет состоять из чтения указанного элемента данных (поля з.п. некоторого кортежа), получения нового значения и записи указанного элемента, то есть из трех операций */

С точки зрения БД, выполнение программы некоторого приложения может расцениваться как серия транзакций, в промежутках между которыми выполняется некоторая обработка данных, осуществляемая вне среды БД. Например, весь процесс пересмотра зарплаты сотрудникам организации. Выполняется транзакция, извлекающая список сотрудников с их зарплатой. После этого выполняется некоторая обработка данных по удобному представлению их пользователю, возможно он их будет переупорядочивать различным образом, увеличивать или уменьшать шрифт представления – это все будет обработка вне БД, если конечно программа не написана так, что при переупорядочивание данные не считываются заново, то есть если эта функция не решена средствами SQL. За тем может идти произвольное чисто простых транзакций, описанных ранее. Если не табельный номер, а суррогат, то обработка вне БД может заключаться в написании правильного запроса, так как пользователь не видит суррогаты, но они считываются и где-то размещаются в программе.

Любая транзакция всегда должна переводить БД из одного согласованного состояния в другое согласованное, хотя допускается, что согласованность БД будет нарушаться в ходе выполнения транзакции.

/* Пусть студент отучился на специальности некоторое время и получил оценки согласно плану этой специальности. При переводе его на другую специальность транзакция будет состоять из следующих действий: найти предметы, которые должен был изучить студент согласно плану новой специальности; для каждого этого предмета найти оценку у переводимого студента и перезачесть ее; все остальные оценки удалить. То есть и до и после выполнения транзакции БД находится в согласованном состоянии: все студенты имеют оценки согласно плану обучения по специальности, но в ходе выполнения часть оценок студента не относятся к специальности на которой он обучается.

Пример демонстрирует и сложность транзакции, и согласованность БД, и нарушение согласованности в ходе выполнения транзакции.

Любая транзакция завершается одним из двух возможных способов. В случае успешного завершения результаты транзакции фиксируются (commit), и БД переходит в новое согласованное состояние. Если выполнение транзакции не увенчалось успехом, она отменяется. В этом случае в БД должно быть восстановлено то согласованное состояние, в

котором она находилась до начала данной транзакции. Этот процесс называется откатом (roll back) транзакции. Зафиксированная транзакция не может быть отменена. Если зафиксированная транзакция была ошибочной, то необходимо выполнить другую транзакцию, возвращающую БД в предыдущее согласованное состояние.

В большинстве языков манипулирования данными для указания границ отдельных транзакций используются операторы begin transaction, commit, rollback.

COMMIT. Оператор COMMIT завершает транзакцию и делает любые выполненные в ней изменения постоянными. Освобождаются блокировки.
ROLLBACK. Оператор отката завершает транзакцию и отменяет все выполненные в ней и незафиксированные изменения. Для этого он читает информацию из сегментов отката и восстанавливает блоки данных в состояние, в котором они находились до начала транзакции. Освобождаются блокировки.

По завершении транзакции необходимо явно указывать одну из команд завершения транзакции иначе за вас это сделает среда, в которой вы работаете (а среда не всегда это делает так, как вы предполагаете).
SAVEPOINT. Позволяет создать в транзакции точку сохранения. В одной транзакции можно выполнять оператор SAVEPOINT несколько раз, устанавливая несколько точек сохранения. Точки сохранения позволяют устанавливать маркеры внутри транзакции таким образом, чтобы была возможность отмены только части работы, проделанной в транзакции. Оправдано использование точек сохранения в продолжительных и сложных транзакциях. ORACLE освобождает блокировки, которые были установлены отменённым оператором.
ROLLBACK TO <точка сохранения>. Этот оператор используется совместно с представленным выше оператором SAVEPOINT. Транзакцию можно откатить до указанной точки сохранения, не отменяя все сделанные до нее изменения. Таким образом, можно выполнить два оператора UPDATE, затем — оператор SAVEPOINT, а после него — два оператора DELETE. При возникновении ошибки или исключительной ситуации в ходе выполнения операторов DELETE транзакция будет откатываться до указанной оператором SAVEPOINT точки сохранения; при этом будут отменяться операторы DELETE, но не операторы UPDATE.
SET TRANSACTION. Этот оператор позволяет устанавливать атрибуты транзакции, такие как уровень изолированности и то, будет ли она использоваться только для чтения данных или для чтения и записи. Этот оператор также позволяет привязать транзакцию к определенному сегменту отката.

Свойства транзакций

Существуют свойства, которыми должна обладать транзакция. Они называются ACID:

1. ***Атомарность.*** **(«все или ничего» atomic). Любая транзакция представляет собой неделимую единицу работы, которая может быть либо выполнена вся целиком, либо не выполнена вовсе.**
2. ***Согласованность*** **(coordination). Каждая транзакция должна переводить БД из одного согласованного состояния в другое.**
3. ***Изолированность*** **(insulativity). Все транзакции выполняются независимо одна от другой. Другими словами, промежуточные результаты незавершенной транзакции не должны быть доступны другим транзакциям.**
4. ***Продолжительность*** **(duration). Результаты успешно завершенной транзакции должны сохраняться в БД постоянно и не должны быть утеряны в результате последующих сбоев.**

Некоторые особенности выполнения транзакций в ORACLE:

Транзакция обычно состоит из нескольких операторов DML . Если один оператор дает сбой, то он один откатывается. То есть все операторы, которые раньше были выполнены, не откатываются автоматически – результаты их работы не пропадают. Вы можете дальше продолжать транзакцию. Затем её или зафиксировать, или откатить. А получаем мы такой эффект потому, что ORACLE каждый оператор транзакции помещает в неявные операторы Savepoint так, как это показано далее:

Savepoint statement1;
Оператор1;
If error then rollback to statement1;
Savepoint statement2;
Оператор2;
If error then rollback to statement2;

Понятие неделимости распространяется на необходимую глубину. Например, мы вставляет записи в таблицу 1, что вызывает срабатывание триггера на вставку записей в таблицу 2, что в свою очередь вызывает срабатывание триггера на обновление таблицы 3 и так далее. Если в какой-то момент происходит откат нашего оператора по таблице 1, то отменяются и все изменения, произведенные в таблице 2,3, и т.д. То есть или все изменения фиксируются, или все отменяется.
ORACLE анонимный блок PL/SQL считает оператором. Например,
begin оператор1;
оператор2; end;
То есть для него применимо предыдущее замечание.

Ограничение целостности проверяются после выполнения каждого sql-оператора. ORACLE разрешает делать некоторые строки таблицы несогласованными до конца выполнения sql-оператора.
В ORACLE есть возможность отложить проверку целостности на любой момент времени до конца транзакции. Это реализуется с помощью ограничения deferrable таблицы и перевода ограничения в режим deferred.
В ORACLE можно использовать распределенные транзакции, то есть выполнять транзакцию, в которой операторы работают на удаленных сервера (распределенная база данных). Для доступа к удаленной базе данных используется объект database link.
В ORACLE продолжительность транзакции не ограничивается, потому что проблемы поедания ресурсов блокировками не существует. Транзакция длится столько, сколько нужно приложению. Единственная проблема: при очень длительных транзакциях и маленьком сегменте отката возможна ошибка ORA-1555.

Вопросы к лекции

1. Объясните понятие транзакции.
2. Какие операторы используются для указания границ отдельных транзакций?
3. Свойства транзакций.
4. Из чего обычно состоит транзакция?

Лекция № 15 **Редактирование и создание записей вычисления. SQL выражения управления транзакциями (COMMIT, SAVE POINT, ROLL BACK, WORK и др).**

План

1. SQL выражения управления транзакциями.
2. COMMIT
3. ROLLBACK
4. SAVEPOINT

Модель транзакций реляционной СУБД Oracle основана на «единице работы» (unit of work) и поддерживается большую часть операций над транзакциями (нельзя использовать ROLLBACK FORSE). Транзакция неявно начинается с началом сеанса или при выполнении первой команды SQL после последней команды COMMIT или ROLLBACK (фактически при первом изменении данных). Транзакция заканчивается при выполнении команды COMMIT или ROLLBACK.

Транзакция не зависит от блоков PL/SQL. Транзакции могут охватывать несколько системных блоков, или в одном блоке может быть несколько транзакций (автономных). PL/SQL поддерживает следующие команды для транзакций: COMMIT, ROLLBACK, SAVEPOINT, TRANSACTION и LOCK TABLE.

COMMIT делает изменения в БД постоянными и видимыми для других сеансов в соответствии со следующим синтаксисом:

COMMIT [WORK] [COMMENT *текст*];

где WORK – необязательное ключевое слово, применяется для улучшения читаемости и соответствия стандарту. Необязательный комментарий COMMENT текст может иметь длину до 50 символов.

ROLLBACK отменяет не зафиксированные изменения, сделанные в текущей транзакции, влияет до начала транзакции или до указанной точки сохранения.

ROLLBACK [WORK] [TO [SAVEPOINT]*имя_точки_сохранения*];

SAVEPOINT устанавливает точку сохранения (именованную точку обработки) для текущей транзакции. Установка точки транзакции делает возможным выполнение частичного отката.

SAVEPOINT *имя_точки_сохранения;*

где *имя_точки_сохранения* – это необъявленный идентификатор. Внутри транзакции может быть установлено несколько точек сохранения. Если повторно использовать имя точки сохранения то точка передвинется на новую позицию и откат к исходной позиции данной точки сохранения будет невозможен.

SET TRANSACTION управляет типом транзакции

SET TRANSACTION *тип_транзакции* NAME *имя*
где имя – имя транзакции, доступное на протяжении ее выполнения; а *тип_транзакции м*ожет принимать следующие значения:

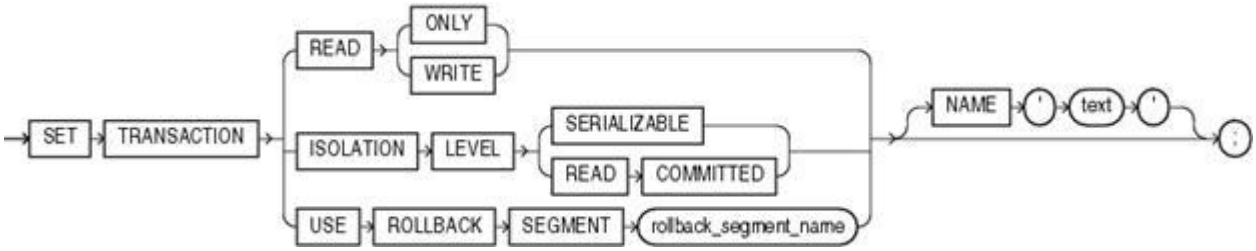
- READ ONLY – означает начало транзакции только для чтения. Указывает СУБД, что следует обеспечить целостность по чтению БД в пределах транзакции (по умолчанию для команды). Транзакцию заканчивают команды COMMIT или ROLLBACK. Внутри такой транзакции разрешены только команды LOCK TABLE, SELECT, SELECT INTO, OPEN, FETCH, CLOSE, COMMIT и ROLLBACK. Попытка выполнения в транзакции только для чтения других команд, такие как INSERT или UPDATE, приводит к появлению ошибки ORA-1456.

- READ WRITE – обозначает начало транзакции READ WRITE; это тип по умолчанию.

- ISOLATION LEVEL SERIALIZABLE – аналогично транзакции READ ONLY обеспечивается целостность по чтению в пределах транзакции вместо поведения по умолчанию – целостности по чтению на уровне команды. Сериализуемые транзакции не позволяют изменять данные.

- ISOLATION LEVEL READ COMMITTED – если транзакции необходимы строки, заблокированные другими транзакциями, она будет ждать снятия блокировки.

- USE ROLLBACK SEGMENT *имя_сегмента_отката* – указывает СУБД, что следует использовать указанный сегмент отката. Полезна, когда лишь один сегмент отката имеет большой размер, и известно, что программе необходим большой сегмент отката (например, при операции закрытия в конце месяца).



LOCK TABLE обходит неявные блокировки БД на уровне строк, явно блокируя целиком одну или несколько таблиц в указанном режиме.
LOCK TABLE *список_таблиц* IN *режим_блокировки* MODE [NOWAIT];
где *список_таблиц* – список таблиц, разделенных запятыми; *режим_блокировки* – может принимать значения ROW SHARE (SHARE UPDATE), ROW EXCLUSIVE, SHARE, SHARE ROW EXCLUSIVE или EXCLUSIVE.
ROW SHARE разрешает конкурирующий доступ к заблокированной таблице, но запрещает пользователям блокировать всю таблицу для эксклюзивного доступа (EXCLUSIVE). ROW SHARE и SHARE UPDATE это синонимы, используются для совместимости синтаксиса более ранних версий.
ROW EXCLUSIVE аналогично ROW SHARE, за исключением того, что дополнительно запрещает блокировать в режиме SHARE. ROW EXCLUSIVE применяется СУБД автоматически при операциях INSERT, UPDATE, DELETE.

SHARE разрешает одновременные запросы к таблице, но запрещает обновлять заблокированную таблицу.
SHARE ROW EXCLUSIVE используется для просмотра всей таблицы и разрешения пользователям просматривать строки в таблицы, но запрещает другие блокировки в режиме SHARE и запрещает обновлять строки.EXCLUSIVE разрешает запросы к заблокированной таблице, но запрещает все остальные действия.NOWAIT означает, что база данных передает управление пользователю или программе данным разделом, подразделом или таблицей незамедлительно, даже если на них наложена блокировка другим пользователем. В этом случае БД возвращает предупреждающее сообщение о том, что другим пользователем уже была наложена блокировка. Если встречается блокировка (и указано ключевое слово NOWAIT), то СУБД генерирует исключение: ORA-00054: resource busy and acquire with NOWAIT specified.

Другими словами, NOWAIT указывает, что СУБД не должна ждать освобождения блокировки, в то время как по умолчанию при встрече блокировки СУБД бесконечно ждет ее освобождения.

Вопросы к лекции

1. На чем основана модель транзакций СУБД Oracle?
2. Какова роль SQL выражений для управления транзакциями?
3. Объясните назначение COMMIT**,** ROLLBACK**,** SAVEPOINT.
4. Какова функция EXCLUSIVE?

Лекция №16 **Обеспечение защиты базы данных.**

План

1. Представления словаря данных
2. Управление пользователями базы данных
3. Обеспечение целостности базы данных

Представления словаря данных

Словарь данных - не только центральное хранилище в каждой базе данных ORACLE. Это также важный инструмент для всех пользователей, от конечных пользователей до разработчиков приложений и администраторов базы данных. Даже начинающие пользователи могут извлекать выгоду из понимания и использования словаря данных.

Введение в словарь данных

Словарь данных является одной из важнейших частей базы данных ORACLE. Словарь данных - это набор таблиц, используемых как справочник только для чтения, который предоставляет информацию об ассоциированной с ним базе данных. Например, словарь данных может предоставлять следующую информацию:

- имена пользователей ORACLE
- привилегии и роли, которые были предоставлены каждому пользователю
- имена объектов схем (таблиц, представлений, снимков, индексов, кластеров, синонимов, последовательностей, процедур, функций, пакетов, триггеров и т.д.)
- информацию об ограничениях целостности
- умалчиваемые значения для столбцов

- сколько пространства было распределено и в настоящее время используется объектами в базе данных
- информацию аудита, например, кто обращался к различным объектам и обновлял их
- другую общую информацию о базе данных

Словарь данных структурирован через таблицы и представления, как и любые другие данные в базе данных. Для обращения к словарю данных вы используете SQL. Так как словарь данных можно только читать, пользователи могут выдавать лишь запросы (предложения SELECT) по таблицам и представлениям словаря данных.

Структура словаря данных

В состав словаря данных базы данных входят:

Базовые таблицы	Основу словаря данных составляет совокупность базовых таблиц, хранящих информацию о базе данных. Эти таблицы читаются и пишутся ТОЛЬКО самим ORACLE; они редко используются непосредственно пользователем ORACLE любого типа, потому что они нормализованы, и большая часть данных в них закодирована.
Доступные пользователю представления	Словарь данных содержит доступные пользователю представления , которые суммируют и отображают, в удобном представлении формате информацию из базовых таблиц словаря. Эти представления декодируют информацию базовых таблиц, представляя ее в полезном виде, таком как имена пользователей или таблиц, и используют соединения и фразы WHERE, чтобы упростить информацию. Большинство пользователей имеют доступ к этим представлениям вместо базовых таблиц словаря.

Все базовые таблицы и представления словаря данных принадлежат пользователю ORACLE с учетным именем SYS. Поэтому ни один пользователь ORACLE не должен изменять никаких объектов, содержащихся в схеме SYS, а администратор безопасности должен строго контролировать использование этого центрального учетного имени.

Данные в базовых таблицах словаря данных необходимы для функционирования ORACLE. Поэтому только ORACLE должен записывать или изменять информацию словаря данных.

Во время операций по базе данных ORACLE читает словарь данных, чтобы удостовериться, что нужные объекты существуют, и что пользователи имеют к ним должный доступ. ORACLE также непрерывно обновляет словарь данных, чтобы отражать происходящие изменения в структурах базы данных, аудите, грантах и данных.

Вы можете добавлять в словарь данных новые таблицы или представления. Если вы добавляете новые объекты в словарь данных, их владельцем должен быть SYSTEM или какой-либо третий пользователь ORACLE. Когда включен режим аудита, эта таблица может неограниченно расти. Хотя пользователи не должны удалять эту таблицу, администратор безопасности может удалять из нее данные, потому что строки этой таблицы служат лишь для информации и не являются необходимыми для работы ORACLE.

Представления словаря данных выступают как справочники для всех пользователей базы данных. Доступ к этим представлениям осуществляется через SQL. Некоторые представления доступны всем пользователям, тогда как некоторые другие предназначены лишь для администраторов.

Словарь данных всегда доступен при открытой базе данных. Он размещается в табличном пространстве SYSTEM, которое всегда находится в состоянии онлайн, когда база данных открыта.

Словарь данных состоит из нескольких наборов представлений. Во многих случаях такой набор состоит из трех представлений, содержащих аналогичную информацию и отличающихся друг от друга своими префиксами:

Префикс	**Назначение**
USER	обзор пользователя (что есть в схеме пользователя)
ALL	расширенный обзор пользователя (к чему есть доступ)
DBA	обзор администратора (к чему все пользователи имеют доступ)

Столбцы в каждом представление в наборе идентичны, со следующими исключениями:

В представлениях с префиксом USER обычно нет столбца с именем OWNER (владелец); в представлениях USER под владельцем подразумевается пользователь, выдавший запрос.
Некоторые представления DBA имеют дополнительные столбцы, которые содержат информацию, полезную для АБД.
Представления с префиксом USER:
отражают собственное окружение пользователя в базе данных, включая информацию об объектах, созданных этим пользователем, грантах, предоставленных им, и т.д.
выдают лишь строки, имеющие отношение к пользователю
имеют столбцы, идентичные с другими представлениями, с тем исключением, что столбец OWNER подразумевается (текущий пользователь)
возвращают подмножество информации, предоставляемой представлениями ALL могут иметь сокращенные общие синонимы для удобства
Представления с префиксом ALL отражают общее представление о базе данных со стороны пользователя. Эти представления возвращают информацию об объектах, к которым пользователь имеет доступ через общие или явные гранты привилегий и ролей, помимо тех объектов, которыми владеет этот пользователь. Представления с префиксом DBA показывают общее представление о базе данных, так что они предназначены только для администраторов базы данных. Точнее, опрашивать представления словаря с префиксом DBA может любой пользователь, имеющий системную привилегию SELECT ANY TABLE.

Управление пользователями базы данных

Центральное место в средствах защиты занимает учётная запись пользователя базы данных Oracle.
CREATE USER – это команда SQL, которая может использоваться для определения учётной записи Oracle в базе данных. После создания учётной записи пользователя Oracle она не может использоваться, пока пользователь не получит, по меньшей мере одну системную привилегию. Системная привилегия CREATE SESSION позволяет пользователю создавать сеанс по отношению к базе данных Oracle. Это – необходимая привилегия, которую должна иметь учётная запись пользователя, без неё учётная запись пользователя Oracle не может использоваться.При первоначальном создании пользователя Oracle можно определить заданное по умолчанию табличное пространство, в котором будут создаваться объекты пользователя. Если заданное по умолчанию табличное пространство не определено, пользователю будет назначено табличное пространство SYSTEM в качестве заданного по умолчанию, которое будут использовать объекты базы данных. В составе оператора CREATE USER может использоваться фраза DEFAULT TABLESPACE для определения того, что объекты пользователя должны быть помещены в

табличное пространство, отличное от SYSTEM. Пользователю Oracle также должна быть назначена квота, которая определяет, сколько памяти он может использовать в табличном пространстве.
Другой способ создания пользователя состоит в том, чтобы предоставить пользователю роли CONNECT, RESOURCE и DBA. Хотя это и быстрый метод, он включён, прежде всего, для совместимости с предыдущими версиями программного обеспечения Oracle. Команда CREATE USER – более предпочтительный метод, поскольку в одной команде можно указать и квоту и другие установки.Можно использовать команду ALTER USER для изменения таких параметров пользователя, как пароль, заданные по умолчанию временные табличные пространства и квота памяти.Для удаления пользователя из базы данных используется команда DROP USER, которая удаляет запись пользователя из словаря данных Oracle. Если пользователь Oracle владеет какими-либо объектами базы данных, можно либо удалить каждый из объектов перед использованием команды DROP USER, либо использовать в DROP USER опцию CASCADE для автоматического уничтожения всех объектов при удалении учётной записи пользователя.

Обеспечение целостности базы данных

Целостность данных определением правил проверки достоверности данных гарантирующих, что недействительные данные не попадут в ваши таблицы. Oracle позволяет определять и хранить эти правила для объектов БД, которых они касаются, таким образом, чтобы кодировать их только однажды. При этом они активируются всякий раз, когда какой-либо вид изменение проводится в таблице, независимо от того, какая программа выполняет вставки, модификации или удаления. Этот контроль осуществляется в форме ограничений и триггеров БД. Ограничения – это правила, применимые к таблицам во временя или после создания, распространяемые на то, как эти таблицы могут заполняться.
Ограничение целостности устанавливает правила на уровне БД, 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. Поскольку ограничения условности проверяются на уровне БД, они выполняются независимо от того, откуда были инициированы операторы вставки, модификации или удаления. Для таблиц можно задавать следующие типы ограничений целостности:
NOT NULL
PRIMARY KEY
UNIQUE KEY
FOREIGN KEY (REFERENCES)
CHECK
INDEX (ИНДЕКСЫ)
TRRIGERS и PROCEDURES

NOT NULL. Это ограничение устанавливается для столбца, чтобы указать, что столбец должен иметь значение в каждой строке, т.е. некоторое непустое значение.
PRIMARY KEY (первичный ключ) . Ограничение определяет столбец или группу столбцов, которую можно использовать для уникальной идентификации строки. Никакие две строки в таблице не могут иметь одинаковые значения столбцов первичного ключа. Кроме того, столбцы первичного ключа должны всегда содержать значение. Все эти условия гарантируют то, что в нашем распоряжение будет одна и только одна строка, соответствующая критериям связывания. Первичные ключи могут быть или именованные (пользователем) или неименованные (Oracle составляет имя сам). В первичных ключах не могут использоваться столбцы типа: raw, long, long raw.
UNIQUE (уникальный). Ограничение UNIQUE используется для определения того, что значения в столбце не должно повторяться в другой строке этой таблицы, определяет

вторичный ключ для таблицы. Это столбец или группа столбцов, которые можно использовать как уникальную идентификацию строки. Никакие две строки не могут иметь одинаковые значения для столбца или столбцов ключа UNIQUE. Столбцы для ограничения UNIQUE не обязательно NOT NULL. Можно сформировать ограничение таблицы, указав, что в таблице не должна повторяться комбинация столбцов. К примеру: можно в начале объявить стандартно emp_id number(5), person_id date а под конце объявить что: unique(emp_id, person_id) – и получиться, что сочетание значений этих полей, должно быть уникальным в каждой строке.

FOREIGN KEY (внешний ключ). FOREIGN KEY, устанавливает отношение целостности между таблицами. Оно требует, чтобы столбец или набор столбцов в одной таблице совпадал с первичным или вторичным ключом другой таблицы. С момента создания внешнего ключа ссылающегося на первичный ключ некой таблицы удаление таблицы будет – запрещено. И обойти это ограничение можно только удалив ограничение. Внешние ключи могут быть именованные или неименованные.

CHECK. Ограничение CHECK определяет логику проверки, которая должна жать результат true (истина) для оператора вставки, модификации или удаления из таблицы. Ограничение CHECK гарантирует, что значение в измененной строке удовлетворяют заданному набору проверок правильности.

ИНДЕКСЫ (INDEX). Ограничения PRIMARY KEY и UNIQUE автоматически создают индексы на столбцах, для которых они определены, если ограничение активизируется при создании. Если индекс уже существует на столбцах, которые составляют ограничение PRIMARY KEY и UNIQUE, то использует именно этот индекс и Oracle не может создать новый.

TRRIGERS (Триггеры) – с программный элемент хранимый в БД выполняемый автоматически, в определенных ситуациях, не имеющий входных или выходных параметров, что в конечном итоги и является причиной невозможности вызвать его явно, непосредственно, его вызывает только сама база данных Oracle. Выходные данные триггера должны быть также применимы к БД, а не возвращены вызывающей программе или отображены на экране.

Таким образом, целостность базы данных может быть рассмотрена на трех уровнях:

- На уровне типа данных (т.е. соответствия типов данных)
- На уровне ключей (к примеру, соответствие первичных и внешних ключей)
- На уровне триггеров, процедур и /или функций(к примеру, триггера отвечают только за свои области данных).

Вопросы к лекции

1. Что такое словарь данных?
2. Какова структура словаря данных?
3. Как создается учётная запись пользователя базы данных Oracle?
4. Какие типы ограничений целостности можно задавать для таблиц?

Лекция №17 **Администрирование базы данных. Основные задачи администратора базы данных. Работа Enterprice Manager. Хранение данных.**

План

1. Администрирование базы данных
2. Основные задачи администратора базы данных
3. Основные типы администраторов БД

Администрирование базы данных – это функция управления базой данных (БД). Лицо ответственное за администрирование БД называется “Администратор базы данных” (АБД) или “Database Administrator” (DBA).

Функция “администрирования данных” стала активно рассматриваться и определяться как вполне самостоятельная с конца 60-х годов. Практическое значение это имело для предприятий, использующих вычислительную технику в системах информационного обеспечения для своей ежедневной деятельности. Специализация этой функции с течением времени совершенствовалась, но качественные изменения в этой области стали происходить с началом использования так называемых интегрированных баз данных. Одна такая база данных могла использоваться для решения многих задач.

Таким образом, сформировалось определение БД как общего информационного ресурса предприятия, которое должно находиться всегда в работоспособном состоянии. И как для каждого общего ресурса значительной важности, БД стала требовать отдельного управления. Во многих случаях это было необходимо для обеспечения её повседневной эксплуатации, её развития в соответствии с растущими потребностями предприятия. К тому же БД и технология её разработки постоянно совершенствовались и уже требовались специальные знания высокого уровня для довольно сложного объекта, которым стала база данных. Отсюда функция управления базой данных и получила название “Администрирование базы данных”, а лицо ею управляющее стали называть “Администратор баз данных”.

Администратор базы данных (DBA)

Администратор базы данных (АБД) или **Database Administrator** (DBA) – это лицо, отвечающее за выработку требований к базе данных, её проектирование, реализацию, эффективное использование и сопровождение, включая управление учётными записями пользователей БД и защиту от несанкционированного доступа. Не менее важной функцией администратора БД является поддержка целостности базы данных.

АБД имеет код специальности по общероссийскому классификатору профессий рабочих, должностей служащих и тарифных разрядов (ОКПДТР) — 40064 и код 2139 по Общероссийскому классификатору занятий (ОКЗ). Код 2139 ОКЗ расшифровывается следующим образом: 2 - СПЕЦИАЛИСТЫ ВЫСШЕГО УРОВНЯ КВАЛИФИКАЦИИ, 21 - Специалисты в области естественных* и инженерных наук, 213 - Специалисты по компьютерам, 2139 - Специалисты по компьютерам, не вошедшие в другие группы.

Классические подходы к наполнению содержанием понятия "АБД" стали формироваться после издания рабочего отчета группы по базам данных Американского Национального Института Стандартов ANSI/X3/SPARC в 1975 года. В этом отчете была описана трехуровневая архитектура СУБД, в которой выделялся уровень внешних схем данных, уровень концептуальной схемы данных и уровень схемы физического хранения данных. В соответствии с этой архитектурой определялись три роли АБД: администратор концептуальной схемы, администратор внешних схем и администратор хранения данных. Эти роли в случае очень маленькой системы мог играть один человек, в большой системе для выполнения каждой роли могла назначаться группа людей. Каждой роли соответствовал набор функций, а все эти функции вместе составляли функции АБД.

В 1980 - 1981 г. в американской литературе стало принятым включать в функции АБД:

- организационное и техническое планирование БД,

- проектирование БД,
- обеспечение поддержки разработок прикладных программ,
- управление эксплуатацией БД.

В нашей стране в это же время первое определение АБД в ГОСТ-ах задало слишком узкий состав функций АБД:

- подготовка вычислительного комплекса к установке СУБД, участие в установке и приемке СУБД и самой БД с комплексом прикладных программ,
- управление эксплуатацией БД,
- подготовка словарей и другой НСИ - нормативно-справочной информации - к моменту начала испытания БД.

Предполагалось, что функции АБД будут ориентированы только на эксплуатацию БД, а её разработка будет вестись силами специализированной организации.

К середине 90-х годов сложились еще не завершенные, но уже достаточно устойчивые и полные методологии разработки систем с базами данных. Основная работа по планированию информационных потребностей предприятия, проектированию концептуальной и логической схемы БД, внешних схем, используемых в отдельных процессах обработки информации, ложится теперь на группу проектирования Автоматизированной Системы (АС). Становиться и более определённым объем функций АБД. Это обеспечение надежной и эффективной работы пользователей и программ с БД, поддержка разработчиков в их доступе к БД и средствам разработки.

Основные задачи администратора базы данных

Задачи АБД могут незначительно отличаться в зависимости от вида применяемой СУБД, но в основные задачи входит:

- Проектирование базы данных.
- Оптимизация производительности базы данных.
- Обеспечение и контроль доступа к базе данных.
- Обеспечение безопасности в базе данных.
- Резервирование и восстановление базы данных.
- Обеспечение целостности баз данных.
- Обеспечение перехода на новую версию СУБД.

Основные типы администраторов БД

Среди АБД нет строгого документального разграничения по типам. Но можно выделить несколько общих видов АБД, в зависимости от возложенных на них обязанностей:

- Системный администратор.
- Архитектор БД.
- Аналитик БД.
- Разработчик моделей данных.
- Администратор приложении.
- Проблемно-ориентированный администратор БД.
- Аналитик производительности.
- Администратор хранилища данных.

База данных

База данных - совокупность связанных данных, организованных по определенным правилам, предусматривающим общие принципы описания, хранения и манипулирования, независимая от прикладных программ. База данных является информационной моделью предметной области. Обращение к базам данных осуществляется с помощью системы управления базами данных (СУБД).

Базой данных часто ошибочно называют систему управления базами данных. Необходимо различать хранимые данные (собственно БД) и программное обеспечение, предназначенное для организации и ведения базы данных (СУБД).

Система управления базами данных

Система управления базами данных (СУБД) — специализированная программа (чаще комплекс программ), предназначенная для организации и ведения базы данных. Используется для упорядоченного хранения и обработки больших объемов информации. В процессе упорядочения информации СУБД генерирует базу данных, а в процессе обработки сортирует информацию и осуществляют ее поиск.

На данный момент времени существуют следующие СУБД:

- **Oracle Database** - объектно-реляционная система управления базами данных, разработанная корпорацией Oracle.
- **Microsoft SQL Server** - система управления реляционными базами данных (СУБД), разработанная корпорацией Microsoft.
- **PostgreSQL** - это объектно-реляционная система управления базами данных основанная на POSTGRES, версии 4.2, которая была разработана в Научном Компьютерном Департаменте Беркли Калифорнийского Университета.
- **MySQL** - свободная система управления базами данных. Является собственностью компании Sun Microsystems, осуществляющей разработку и поддержку приложения.
- **ЛИНТЕР** - российская СУБД, реализующая стандарт SQL-92 и поддерживающая большинство операционных систем, в том числе семейство Windows (включая Windows CE), различные версии UNIX, ОС реального времени (включая QNX).
- **IBM DB2** - семейство программных продуктов в области управления информацией компании IBM.
- **Informix** - семейство систем управления реляционными базами данных (СУБД) выпускаемых компанией IBM. Informix позиционируется как флагманский корабль IBM для онлайновой обработки транзакций (OLTP), также как и для интегрированных решений.
- **Sybase** - система управления базами данных (СУБД) от одноимённой компании.
- **Ingres** - коммерчески поддерживаемая реляционная СУБД с открытыми исходными текстами от компании Ingres Corporation.
- **MariaDB** - открытая СУБД. Является ответвлением от MySQL и развивается компанией Monty Program Ab, созданной Майклом Видениусом после его ухода из Sun Microsystems.

Вопросы к лекции

1. Что значит администрирование базы данных **?**
2. Что значит администратор базы данных ?
3. Перечислите основные типы администраторов БД ?
4. Перечислите основные задачи администратора базы данных?

Лекция №18. **Импорт и экспорт данных. Обслуживание, обновление и дополнение СУБД. Создание резервной копии базы данных. Повышение эффективности базы данных.**

План

1. Об импорте, экспорте, загрузке и выгрузке данных
2. Выбор подходящего варианта импорта/экспорта/загрузки/выгрузки
3. Выгрузка и загрузка данных при помощи мастеров

Об импорте, экспорте, загрузке и выгрузке данных

Oracle Database XE позволяет копировать данные между базами данных Oracle, а также обмениваться данными с внешними файлами. Копирование данных осуществляется посредством *экспорта* и *импорта*, а также посредством *выгрузки* и *загрузки*. В следующей таблице имеются определения перечисленных терминов.

Термин	**Определение**
Экспорт	Копирование данных во внешние файлы только для импорта в другую базу данных Oracle. Такие файлы имеют собственный бинарный формат.
Импорт	Копирование данных в базу из внешних файлов, которые были созданы экспортом из другой базы данных Oracle.
Выгрузка	Копирование данных базы во внешние текстовые файлы для использования в другой базе данных Oracle или в другом приложении (например в табличном процессоре). Текстовые файлы имеют общепринятый формат, например с разделением табуляцией или запятыми (CSV).
Загрузка	Копирование данных в базу из внешних текстовых файлов, которые сохранены в формате с разделением табуляцией, запятыми или в любом другом формате, поддерживаемом утилитой SQL*Loader.

Данные, экспортированные из любой версии Oracle Database (Express Edition, Standard Edition, and Enterprise Edition) можно импортировать в любую другую версию базы данных.

Выбор подходящего варианта импорта/экспорта/загрузки/выгрузки

Oracle Database Express Edition (Oracle Database XE) предоставляет на выбор несколько полноценных вариантов для импорта, экспорта, загрузки и выгрузки данных. В Таблице 10-1 перечислены эти возможные варианты.

Таблица 10-1 Сводка вариантов выполнения импорта/экспорта Oracle Database XE

Функционал или утилита	**Описание**
Мастера загрузки/выгрузки данных графического пользовательского интерфейса Oracle Database XE	• Простой в использовании графический интерфейс • Загружает/выгружает из и во внешние текстовые файлы (в формате с разделителями) или XML-файлы • Загружает/выгружает только таблицы, по одной за раз • Доступ только к схеме подключившегося

Функционал или утилита	Описание
	пользователя • Нет фильтрации данных
Утилита SQL*Loader	• Интерфейс командной строки, вызываемый командой sqlldr • Массовая загрузка данных из внешних файлов в базу данных • Поддерживает множество форматов ввода, включая формат с ограничителями, формат с фиксированной длиной записи, формат с переменной длиной записи, потоковый формат • Загружает одновременно несколько таблиц • Полноценные возможности для фильтрации данных
Утилиты "Помпа данных экспорта" и "Помпа данных импорта"	• Интерфейс командной строки, вызываемый командами expdp и impdp • Выполняют экспорт и импорт из одной базы данных Oracle в другую (собственный бинарный формат) • Импорт/экспорт все типов объектов схемы • Импорт/экспорт всей базы данных, всей схемы, нескольких схем, нескольких табличных пространств и нескольких таблиц • Полноценные возможности для фильтрации данных • Высокая скорость выполнения операций • Отсутствует поддержка XMLType
Export and Import utilities	• Интерфейс командной строки, вызываемый командами exp и imp • Выполняют экспорт и импорт из одной базы данных Oracle в другую (собственный бинарный формат) • Имеется поддержка XMLType • Отсутствует поддержка типов данных FLOAT и DOUBLE • Возможности аналогичны помпе данных. Предпочтительно использовать помпу данных (Data Pump), если только вам не надо выполнить импорт или экспорт данных XMLType

Таблица 10-2 содержит несколько сценариев загрузки/выгрузки/импорта/экспорта и рекомендует подходящий вариант использования для каждого.

Таблица 10-2 Сценарии импорта/экспорта и рекомендуемые варианты

Сценарий импорта/экспорта	Рекомендуемый вариант

Сценарий импорта/экспорта	Рекомендуемый вариант
У вас меньше 10 таблиц для импорта, данные находятся в электронных таблицах или в текстовых файлах с разделителями (табулятором или запятой) и нет сложных типов данных (таких как объекты или многозначные поля).	Мастера загрузки/выгрузки данных графического пользовательского интерфейса Oracle Database XE
Вы должны импортировать данные, не имеющие разделителей. Записи имеют фиксированную длину, и определения полей зависят от позиции колонок.	SQL*Loader
Вы должны импортировать текстовые данные, разделенные табулятором, и у вас более 10 таблиц.	SQL*Loader
Вы должны импортировать текстовые данные, и вы хотите импортировать только записи, удовлетворяющие некоторому критерию отбора (например, только записи для сотрудников отдела 3001).	SQL*Loader
Вы хотите импортировать или экспортировать схему полностью из или в другую базу данных Oracle. Среди данных нет типа XMLType.	Помпа данных экспорта и помпа данных импорта
Вы хотите импортировать или экспортировать данные из или в другую базу данных Oracle. Среди данных имеются данные типа XMLType, и нет типов данных FLOAT или DOUBLE.	Импорт (imp) и экспорт (exp)

Выгрузка и загрузка данных

Вы можете выгрузить и загрузить данные следующими способами:

- Выгрузить и загрузить данные при помощи мастеров Выгрузки/Загрузки данных пользовательского графического интерфейса Oracle Database XE

  Мастера могут использовать только текстовые файлы в формате с разделителями.

- Загрузить данные с помощью утилиты Oracle SQL*Loader

  SQL*Loader поддерживает множество форматов текстовых файлов и предоставляет очень широкие возможности.

Выгрузка и загрузка данных при помощи мастеров

Мастера загрузки/выгрузки данных графического пользовательского интерфейса Oracle Database XE позволяют Вам легко загружать и выгружать текстовые данные в формате с разделителями в/из базы данных. Пошаговые мастера имеют следующие возможности:

- Вы можете импортировать или экспортировать XML файлы или текстовые файлы с разделителями (такие, как файлы с разделителем-запятой (.csv) или табулятором).
- Вы можете загружать данные копированием и вставкой из электронных таблиц.
- Вы можете не включать (пропускать) столбцы при загрузке или выгрузке.
- Вы можете загружать в уже существующую таблицу или создать новую таблицу из загружаемых данных.
- При загрузке в новую таблицу, первичный ключ может быть взят из данных или сгенерирован из новой, или существующей последовательности Oracle.
- При загрузке в новую таблицу, наименования столбцов могут быть взяты из загружаемых данных.
- При каждой загрузке из файлов, они сохраняются в архиве загрузки текстовых данных. Вы можете получить к ним доступ из архива в любой момент.

Ограничения заключаются в следующем:

- Мастера загружают и выгружают только табличные данные. Они не могут загрузить или выгрузить другие типы объектов схемы.
- Вы можете загружать или выгружать только в/из принадлежащей вам схемы. Это относится и к пользователям с административными привилегиями.
- Вы можете загрузить или выгрузить только одну таблицу за раз.
- Нет ограничений на типы данных для выгрузки в текст, выгрузки в XML или загрузки XML данных. Тем не менее, загрузка текстовых данных и загрузка данных электронных таблиц (через копирование и вставку) поддерживают только следующие типы данных: NUMBER, DATE, VARCHAR2, CLOB, BINARY_FLOAT и BINARY_DOUBLE.

В этом разделе имеются следующие примеры выгрузки и загрузки данных при помощи мастеров:

- Пример: Выгрузка данных при помощи мастера выгрузки
- Пример: Загрузка данных при помощи мастера загрузки

Пример: Выгрузка данных при помощи мастера выгрузки

Предположим, вы хотите выгрузить таблицу REGIONS, которая является частью пробной схемы HR, для того, чтобы ее можно было использовать в другом приложении. Предположим также, что вы хотите создать текстовый файл с разделителем-табулятором, и вы хотите сохранить данные в файл под именем regions.txt.

Для выгрузки таблицы REGIONS:

1. Подключитесь к графическому пользовательскому интерфейсу Oracle Database XE в качестве пользователя HR.
2. На домашней странице базы данных, щелкните иконку **Utilities**, и далее иконку Data Load/Unload.

3. На странице Data Load/Unload щелкните иконку **Unload**, и далее иконку **Unload to Text**.
   Откроется страница Unload to Text, показывая шаг мастера Schema. На этом шаге мастер отображает раскрывающийся список Schema, в котором выбрана HR. Так как вы можете экспортировать только из своей собственной схемы, то нельзя изменить этот выбор.
4. Нажмите **Next**.
   Откроется шаг Table Name мастера.
5. Из выпадающего списка Table выберите **REGIONS** и нажмите **Next**.
   Откроется шаг Columns мастера.

Описание иллюстрации exp_columns.gif

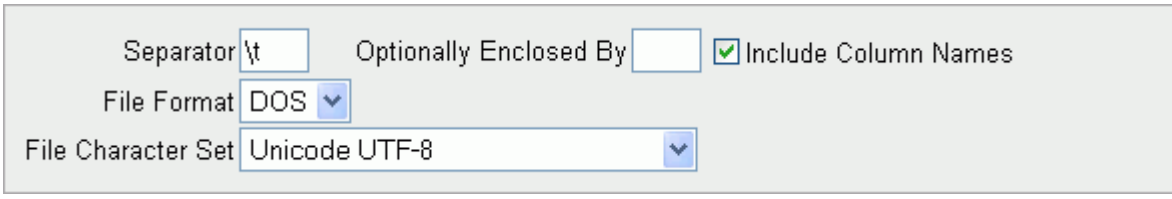
6. Выберите все столбцы щелчком-перетягиванием или shift-щелчком, а затем нажмите Next. (Вам не обязательно выбирать все столбцы. Неотмеченные столбцы будут исключены из операции выгрузки.)

   Откроется шаг Options мастера.

7. Выполните следующие действия:
   a. В поле Separator удалите запятую (если есть) и введите обратный слэш и символ T в нижнем регистре (\t), чтобы в качестве разделителя использовать символ табуляции. (Вы можете использовать любой символ в качестве разделителя.)
   b. Установите флажок Include Column Names.

      Установка этого флажка приведет к тому, что первой выгруженной строкой будут являться наименования столбцов, а не данные. Позднее вы сможете использовать эту первую строку для установки наименований столбцов при загрузке.

   c. В выпадающем списке File Character Set выберите Unicode UTF-8.



Описание иллюстрации exp_options.gif

8. Нажмите Unload Data.

   Появится окно Save As с именем файла regions.txt. В зависимости от используемого браузера этому окну может предшествовать другое окно с запросом, хотите ли вы сохранить или открыть файл. В этом случае выберите вариант для сохранения файла на диск.

9. Сохраните файл regions.txt на рабочем столе или в каталоге по своему усмотрению.
10. (Не обязательно) Откройте файл regions.txt в текстовом редакторе или в табличном процессоре и убедитесь, что таблица REGIONS выгружена должным образом.

Пример: Загрузка данных при помощи мастера загрузки

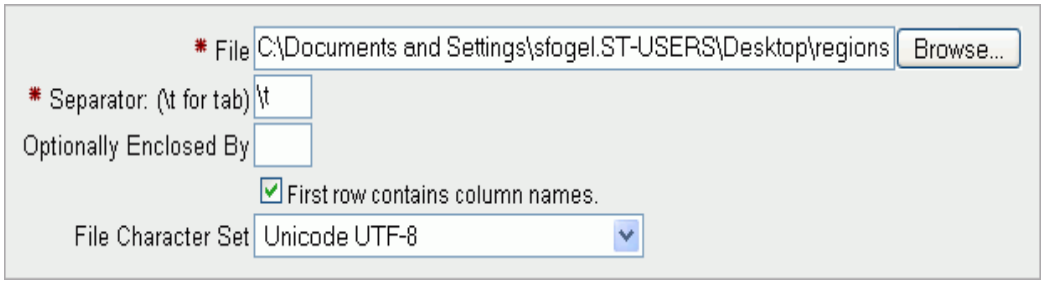
Предположим, что вашему приложению требуется таблица REGIONS, каждая строка которой состоит из кода и названия части света. Также предположим, что Вы выгрузили данные о частях света из настольной базы данных в размеченный табуляцией текстовый файл, который называется regions.txt.

Код региона Вы хотите использовать в качестве бизнес-ключа, а не в качестве первичного ключа, поэтому решаете использовать мастер загрузки так же и для генерации числового первичного ключа для каждой загружаемой записи.

Для загрузки таблицы REGIONS:

1. Войдите в графический интерфейс Oracle Database XE под любой учетной записью, кроме SYSTEM SYSTEM и HR.
   Для того, чтобы предварительно выйти, нажмите кнопку Logout в верхнем правом углу страницы. Смотрите "Доступ к домашней странице базы данных" чтобы получить инструкции по входу. Если не существует других пользователей кроме SYSTEM и HR, то создайте одного. Смотрите "Создание пользователей" о том как это сделать.
2. На домашней странице базы данных выберите иконку Utilities, после чего - Data Load/Unload.
3. На странице Data Load/Unload щелкните по иконке Load, а затем по Load Text Data.
   Откроется страница Load Data на шаге Target and Method мастера.
4. Под заголовком Load To выберите New table, а под заголовком Load From выберите Upload file (comma separated or tab delimited).
5. Нажмите Next.
   Откроется шаг File Details мастера.
6. Выполните следующие действия:
   a. Нажмите Browse, выберите файл regions.txt и нажмите Open.
   b. В поле Separator замените запятую на обратный слэш и T в нижнем регистре (\t) для указания того, что разделителем полей является символ табуляции.

c. В выпадающем списке File Character Set выберите Unicode UTF-8.

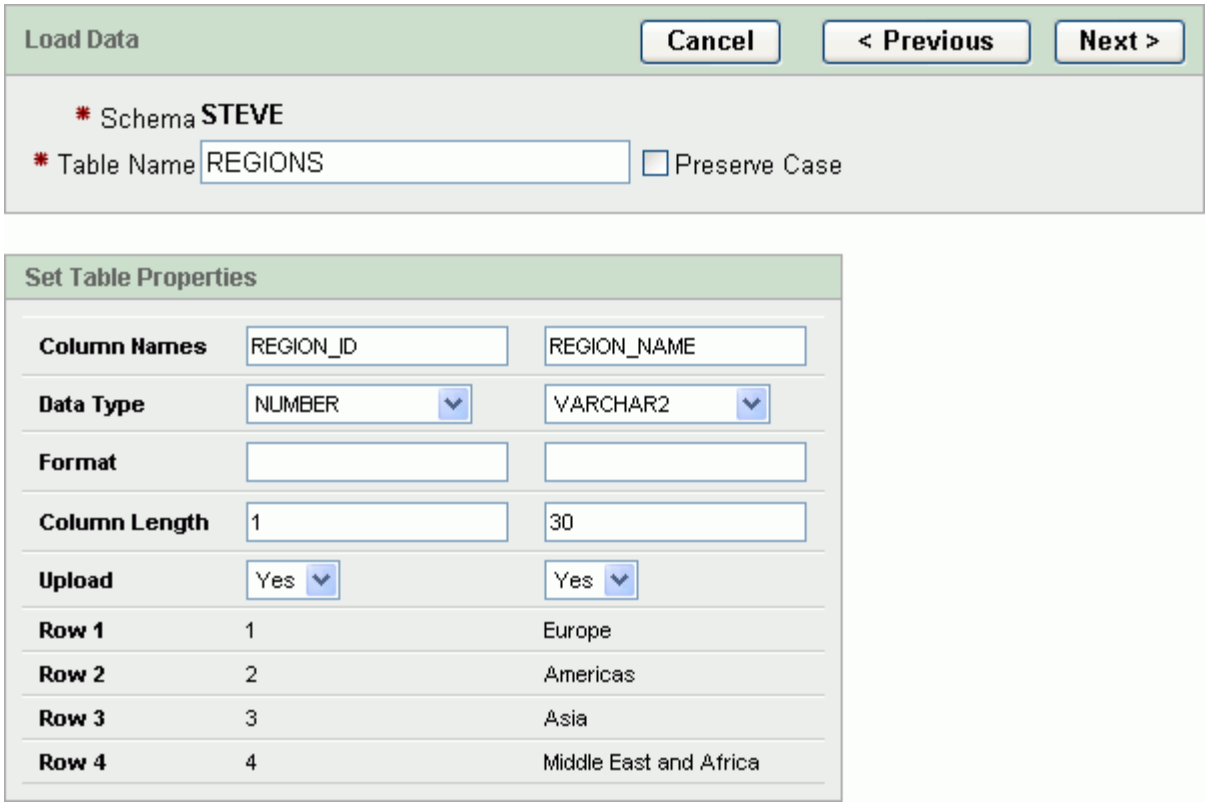


Описание иллюстрации imp_fileinfo.gif

d. Нажмите Next.

Откроется шаг Table Properties мастера.

7. Выполните следующие действия:
   a. В поле Table Name введите REGIONS.



Описание иллюстрации imp_tableinfo.gif

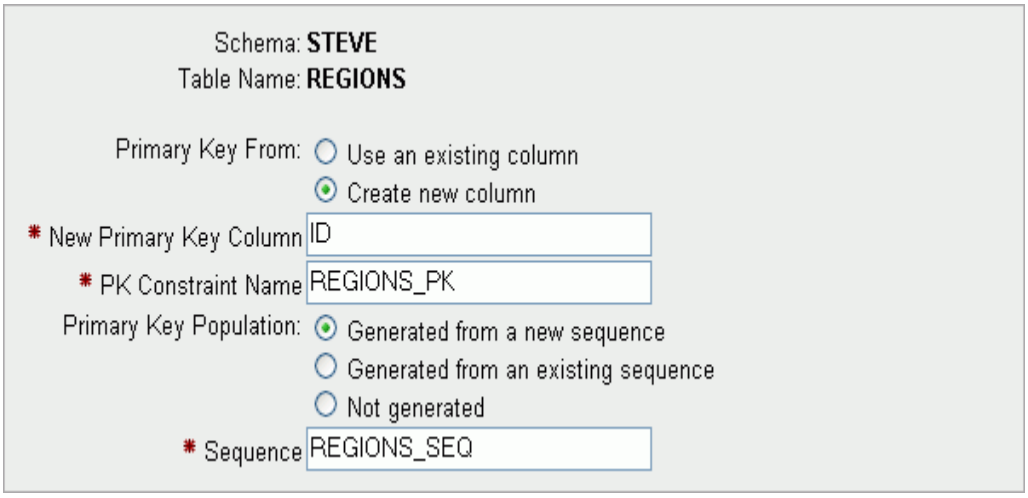
   b. Примите умолчания (Yes) во всех выпадающих списках Upload. Установкой Upload в No можно исключить столбец из операции загрузки.
   c. Нажмите Next.

Откроется шаг Primary Key мастера.

8. Выполните следующие действия:
   a. В Primary Key From выберите Create new column.

b. В Primary Key Population выберите Generated from a new sequence.

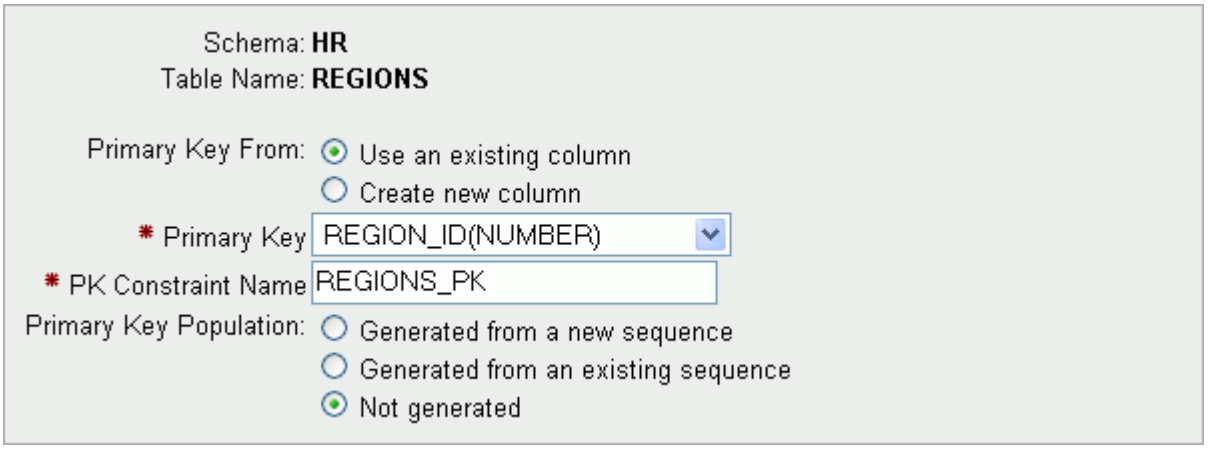
9. Сделанный выбор приведет к следующим действиям Oracle Database XE:
   - Создание в таблице дополнительного столбца ID, который будет использован в качестве первичного ключа для REGIONS.



   - Создание новой последовательности REGIONS_SEQ.
   - Использование значений из последовательности при заполнении поля ID для каждой добавляемой новой строки в таблице.

   Если вы не хотите создавать новый первичный ключ, а хотите использовать существующее поле REGION_ID в качестве первичного ключа, выполните следующие действия:

   - Выберите Use an existing column.
   - В выпадающем списке Primary Key выберите REGION_ID(NUMBER).
   - Выберите Not generated.



Описание иллюстрации imp_pk_alt.gif

10. Нажмите кнопку Load Data.

    Загрузка начнется, а когда она закончится, то появится страница Text Data Load Repository, показывающая файл regions.txt вверху списка загруженных файлов.

11. Проверьте статус загрузки файла regions.txt в колонках Succeeded и Failed.

    Числа в этих колонках показывают количество успешно или неудачно импортированных строк.

Загрузка данных при помощи SQL*Loader

SQL*Loader загружает данные из внешних файлов данных в таблицы базы данных Oracle. Отдельный файл данных может быть в формате с фиксированной длиной записей, в формате с переменной длиной записей или потоковом формате (по умолчанию).

На вход сессии SQL*Loader обычно подается управляющий файл, в котором обозначен способ действия SQL*Loader, и собственно данные, расположенные в конце самого управляющего файла или в отдельном файле данных.

На выходе сессии SQL*Loader находится база данных Oracle database (куда данные загружаются), файл журнала, "bad"-файл и, возможно, файл отмены. В файле журнала помещаются результаты выполнения загрузки, включая описание любых возникших в процессе загрузки ошибок. В плохом файле располагаются записи, которые были отклонены утилитой SQL*Loader или самой базой данных Oracle. В файле отмены располагаются записи, которые не прошли загрузку по фильтру, т.е. не соответствовали некоторому критерию, указанному в управляющем файле.

Методы загрузки данных через SQL*Loader

SQL*Loader применяет три различных метода загрузки данных, в зависимости от ситуации: обычный метод, прямой метод, и внешние таблицы.

Обычный метод

По умолчанию используется обычный метод загрузки. Он заключается в выполнении операторов SQL INSERT для заполнения таблиц базы данных Oracle. Иногда этот метод может оказаться медленнее других, поскольку требуются дополнительные ресурсы для генерации операторов SQL, передачи их серверу и выполнения. Помимо этих причин замедления есть и другая - при выполнении загрузки обычным методом, оперативная память используется в конкуренции со всеми другими процессами.

Прямой метод

При загрузке прямым методом отсутствует конкуренция с другими пользователями за ресурсы базы данных. Этим методом устраняются большинство причин замедления посредством форматирования блоков данных Oracle и прямой записью их в файлы данных, пропуская большую часть обработки данных, происходящую в обычном режиме работы. Таким образом, загрузка прямым методом обычно может быть выполнена быстрее, чем обычным методом. Однако есть несколько

ограничений на выполнение загрузки прямым методом, из-за чего этот метод может вам не подойти. К примеру, прямой метод загрузки не может быть использован для кластерных таблиц или для таблиц с незавершенными транзакциями.

Смотрите *Утилиты Oracle Database* для получения полной информации о ситуациях, в которых может или не может быть применен прямой метод загрузки.

Внешние таблицы

Загрузка во внешнюю таблицу - это создание внешней таблицы для тех данных, которые содержатся в файле данных. Загрузка выполняет операторы INSERT для вставки данных из файла данных в целевую таблицу. Загрузка во внешнюю таблицу позволяет модифицировать загружаемые данные с помощью SQL и PL/SQL функций в составе оператора INSERT, используемого для создания внешней таблицы.

Смотрите *Руководство администратора базы данных Oracle* для получения дополнительной информации о внешних таблицах.

Возможности SQL*Loader

Вы можете использовать SQL*Loader для:

- Загрузки данных по сети. Иными словами, вы можете запустить SQL*Loader не на сервере, а на любом другом клиенте.
- Загрузки данных из нескольких файлов данных в течение одной операции загрузки.
- Загрузки данных одновременно в несколько таблиц одной операцией загрузки.
- Указания символьного набора данных.
- Выборочной загрузки данных (вы можете загружать записи в зависимости от значений отдельных полей).
- Изменения данных перед их загрузкой, используя функции SQL.
- Генерации уникальных последовательных значений ключа в указанных столбцах.
- Доступа к файлам данных средствами операционной системы.
- Загрузки данных с диска, кассеты или именованного шлюза (named pipe).
- Генерации замысловатых отчетов об ошибках, которые могут значительно помочь в локализации ошибок.
- Загрузки сколь угодно сложных объектно-реляционных данных.
- Для загрузки данных типа LOB и коллекций из вспомогательных файлов данных.

Пример: Использование SQL*Loader

В следующем примере будет создана новая таблица dependents в пробной схеме HR. В ней будет содержаться информация о подчиненных, перечисленных в таблице employees схемы HR. После создания таблицы будет использован SQL*Loader для загрузки данных о подчиненных из текстового файла данных в таблицу dependents.

Для этого примера требуются файл данных и управляющий файл SQL*Loader, которые мы создадим на первых двух этапах.

1. Создайте файл данных dependents.dat в своем текущем рабочем каталоге. Вы может создать этот файл множеством способов, например: с помощью табличного процессора, или, просто, набрав текст в редакторе. Содержимое файла должно быть следующим:
2. 100,"Susan, Susie",Kochhar,17-JUN-1997,daughter,101,NULL,
3. 102,David,Kochhar,02-APR-1999,son,101,NULL,
4. 104,Jill,Colmenares,10-FEB-1992,daughter,119,NULL,
5. 106,"Victoria, Vicki",Chen,17-JUN-1997,daughter,110,NULL,
6. 108,"Donald, Donnie",Weiss,24-OCT-1989,son,120,NULL,
7.

   Такой файл называют CSV-файлом (comma-separated values) - в нем запятые играют роль разделителей между отдельными полями. Поле, в котором содержится имя подчиненного, заключено в двойные кавычки на тот случай, если в официальном имени вдруг встретится запятая.

8. Создайте управляющий файл SQL*Loader dependents.ctl в своем текущем рабочем каталоге. Вы можете создать этот файл в любом текстовом редакторе. Содержимое должно быть следующим:
9. LOAD DATA
10. INFILE dependents.dat
11. INTO TABLE dependents
12. REPLACE
13. FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
14. (
15. dep_id,
16. first_name,
17. last_name,
18. birthdate,
19. relation,
20. relative_id,
21. benefits
22.)
23.
24. Выполните одно из:
    - Под Linux: Запустите терминальную сессию и зарегистрируйтесь под учетной записью oracle.
    - Под Windows: Войдите в систему на сервере Oracle Database XE под тем пользователем, который произвел установку Oracle Database XE и откройте командное окно.
25. Под Linux, убедитесь, что переменные окружения установлены в соответствии с инструкциями в "Установка переменных окружения на платформе Linux".
26. Запустите командную строку SQL (SQL*Plus) и подключитесь под пользователем hr посредством команды:
27. sqlplus hr/hr
28.
29. Через командную строку SQL создайте таблицу dependents командой:
30. CREATE TABLE dependents (

```
31.     dep_id      NUMBER(6),
32.     first_name  VARCHAR2(20),
33.     last_name   VARCHAR2(25) CONSTRAINT dep_last_name_nn NOT NULL,
34.     birthdate   DATE,
35.     relation    VARCHAR2(25),
36.     relative_id  NUMBER(6) CONSTRAINT emp_dep_rel_id_fk REFERENCES
    employees
37.       (employee_id),
38.     benefits    CLOB
39.   )
40.   /
```

41. Ограничение (constraint) для столбца last_name указывает, что значение обязательно должно присутствовать. Ограничение для столбца relative_id указывает, что значению должно соответствовать одно из значений столбца employee_id таблицы employees. Столбец benefits имеет тип данных CLOB, поэтому мы можем сохранять здесь большие блоки текстовых данных. (В данном примере никаких льгот не предусмотрено, поэтому в файле данных dependents.dat проставлено NULL в соответствующих полях.)

    После получения сообщения Table created введите exit для выхода из командной строки SQL.

42. Из своего текущего рабочего каталога (в котором вы создали управляющий файл и файл данных) выполните следующую команду SQL*Loader в приглашении командной строки:

```
43.   sqlldr hr/hr DATA=dependents.dat CONTROL=dependents.ctl LOG=dependents.log
44.
```

    Данные из файла dependents.dat загрузятся в таблицу dependents и отобразится следующее сообщение:

```
Commit point reached - logical record count 5
```

    Информация о загрузке будет записана в файл журнала dependents.log. Содержимое файла журнала будет аналогично следующему:

```
Copyright (c) 1982, 2005, Oracle.  All rights reserved.

Control File:   dependents.ctl
Data File:      dependents.dat
  Bad File:     dependents.bad
  Discard File:  none specified

 (Allow all discards)

Number to load: ALL
Number to skip: 0
Errors allowed: 50
Bind array:     64 rows, maximum of 256000 bytes
Continuation:    none specified
```

```
Path used:      Conventional

Table DEPENDENTS, loaded from every logical record.
Insert option in effect for this table: REPLACE

  Column Name                  Position   Len  Term Encl Datatype
------------------------------ ---------- ----- ---- ---- ---------------------
DEP_ID                         FIRST     *   ,  O(") CHARACTER
FIRST_NAME                        NEXT     *   ,  O(") CHARACTER
LAST_NAME                         NEXT     *   ,  O(") CHARACTER
BIRTHDATE                         NEXT     *   ,  O(") CHARACTER
RELATION                          NEXT     *   ,  O(") CHARACTER
RELATIVE_ID                       NEXT     *   ,  O(") CHARACTER
BENEFITS                          NEXT     *   ,  O(") CHARACTER


Table DEPENDENTS:
  5 Rows successfully loaded.
  0 Rows not loaded due to data errors.
  0 Rows not loaded because all WHEN clauses were failed.
  0 Rows not loaded because all fields were null.


Space allocated for bind array:                115584 bytes(64 rows)
Read   buffer bytes: 1048576

Total logical records skipped:          0
Total logical records read:             5
Total logical records rejected:         0
Total logical records discarded:        0

Run began on Mon Dec 05 16:16:29 2005
Run ended on Mon Dec 05 16:16:42 2005

Elapsed time was:     00:00:12.22
CPU time was:         00:00:00.09
```

Теперь вы можете работать с таблицей dependents как с любой другой.

Экспорт и импорт данных

Oracle Database XE предоставляет следующие утилиты командной строки для экспорта и импорта данных:

- Data Pump Export и Data Pump Import
- Export и Import

В следующих раздела дан обзор каждой утилиты. Ситуации, в которых может понадобится та или иная утилита, приведены в Таблице 10-2.

Экспорт и импорт с помощью помпы данных экспорта и помпы данных импорта

Помпа данных экспорта позволяет экспортировать данные и метаданные в набор файлов операционной системы, называемый набор файлов дампа. Помпа данных импорта позваляет импортировать набор файлов дампа в целевую базу данных Oracle.

Набор файлов дампа состоит из одного или нескольких файлов, содержащих табличные данные, метаданные объектов базы данных и управляющую информацию. Эти файлы записываются в специальном бинарном формате, поэтому они могут быть импортированы только помпой данных импорта. Набор файлов дампа может быть импортирован в ту же самую базу данных, откуда был произведен, или в другую базу данных Oracle на другой системе.

Так как файлы дампа записываются сервером базы данных, а не самой помпой данных (клиентом), то необходимо создать объекты с типом *directory*, для тех каталогов, в которые эти файлы будут записаны. Объект directory - это объект базы данных, который является синонимом соответствующего каталога в файловой системе сервера.

Помпы данных экспорта и импорта позволяют вам перемещать подмножество данных и метаданных. Это достигатся с помощью применения параметров помпы данных и режимов импорта, а также различных критериев фильтрации.

Вы можете также осуществлять экспорт и импорт по сети. При экспорте по сети, данные из экземпляра базы данных записываются в набор файлов дампа на сервере. При импорте по сети, данные загружаются в целевую базу данных из базы-источника прямо по сети, минуя стадию файлов дампа. С помощью этого механизма можно запускать экспорт и импорт параллельно, минимизируя время, необходимое для всей этой операции.

Помпа данных экспорта и импорта также имеют набор интерактивных команд, с помощью которых можно просматривать и изменять выполняющиеся задания экспорта и импорта.

Пример: Использование помпы данных экспорта и помпы данных импорта

В этом примере предположим, что вы хотите внести несколько изменений в пробную схему HR, и протестировать эти изменения, но не хотите испортить текущую схему HR. Вы могли бы сделать экспорт схемы HR и импортировать ее в новую схему HRDEV, в которой можно заниматься задачами разработки и тестирования. Чтобы это осуществить, выполните следующее:

1. Сделайте одно из:

- Под Windows: Войдите на сервер Oracle Database XE под тем пользователем, который выполнил установку Oracle Database XE, а затем откройте окно командной строки.
- Под Linux: Запустите терминальную сессию и войдите на сервер Oracle Database XE под учетной записью oracle.

2. Под Linux убедитесь, что переменные окружения настроены в соответствии с инструкциями в "Установка переменных окружения на платформе Linux".
3. В приглашении командной строки задайте команду в соответствии с операционной системой для создания каталога, в который будут помещены файлы экспорта:

   Под Windows:

   MKDIR c:\oraclexe\app\tmp

   Под Linux:

   mkdir /usr/lib/oracle/xe/tmp

4. Запустите командную строку SQL (SQL*Plus) и войдите под пользователем SYSTEM с помощью следующей команды:
5. sqlplus SYSTEM/*password*
6.

   где *password* - это пароль, который был задан для учетных записей SYS и SYSTEM во время установки (Windows) или конфигурирования (Linux) Oracle Database XE.

7. В приглашении командной строки SQL введите следующие команды для создания объекта каталога, названного dmpdir для каталога tmp, который вы только что создали, и для выдачи прав для чтения и записи на него для пользователя HR.

   Под Windows:

   CREATE OR REPLACE DIRECTORY dmpdir AS 'c:\oraclexe\app\tmp';
   GRANT READ,WRITE ON DIRECTORY dmpdir TO hr;

   Под Linux:

   CREATE OR REPLACE DIRECTORY dmpdir AS '/usr/lib/oracle/xe/tmp';
   GRANT READ,WRITE ON DIRECTORY dmpdir TO hr;

8. Выполните экспорт схемы HR в файл дампа, названный schema.dmp с помощью следующей команды в приглашении командной строки операционной системы:
9. expdp SYSTEM/*password* SCHEMAS=hr DIRECTORY=dmpdir DUMPFILE=schema.dmp LOGFILE=expschema.log

10. где *password* - это пароль пользователя SYSTEM.

Во время выполнения операции экспорта будут появляться сообщения, аналогичные следующим:

```
Export: Release 10.2.0.1.0 - Production on Tuesday, 13 December, 2005 11:48:01

Copyright (c) 2003, 2005, Oracle.  All rights reserved.

Connected to: Oracle Database 10g Express Edition Release 10.2.0.1.0 - Production
Starting "SYSTEM"."SYS_EXPORT_SCHEMA_01":  SYSTEM/******** SCHEMAS=hr
  DIRECTORY=dmpdir DUMPFILE=schema.dmp LOGFILE=expschema.log
Estimate in progress using BLOCKS method...
Processing object type SCHEMA_EXPORT/TABLE/TABLE_DATA
Total estimation using BLOCKS method: 448 KB
Processing object type SCHEMA_EXPORT/USER
Processing object type SCHEMA_EXPORT/SYSTEM_GRANT
Processing object type SCHEMA_EXPORT/ROLE_GRANT
Processing object type SCHEMA_EXPORT/DEFAULT_ROLE
Processing object type SCHEMA_EXPORT/TABLESPACE_QUOTA
Processing object type SCHEMA_EXPORT/PRE_SCHEMA/PROCACT_SCHEMA
Processing object type SCHEMA_EXPORT/SEQUENCE/SEQUENCE
Processing object type SCHEMA_EXPORT/TABLE/TABLE
Processing object type SCHEMA_EXPORT/TABLE/INDEX/INDEX
Processing object type SCHEMA_EXPORT/TABLE/CONSTRAINT/CONSTRAINT
Processing object type
SCHEMA_EXPORT/TABLE/INDEX/STATISTICS/INDEX_STATISTICS
Processing object type SCHEMA_EXPORT/TABLE/COMMENT
Processing object type SCHEMA_EXPORT/PROCEDURE/PROCEDURE
Processing object type SCHEMA_EXPORT/PROCEDURE/ALTER_PROCEDURE
Processing object type SCHEMA_EXPORT/VIEW/VIEW
Processing object type
SCHEMA_EXPORT/TABLE/CONSTRAINT/REF_CONSTRAINT
Processing object type SCHEMA_EXPORT/TABLE/TRIGGER
Processing object type
SCHEMA_EXPORT/TABLE/STATISTICS/TABLE_STATISTICS
. . exported "HR"."COUNTRIES"                    6.093 KB      25 rows
. . exported "HR"."DEPARTMENTS"                  6.640 KB      27 rows
. . exported "HR"."EMPLOYEES"                    15.77 KB     107 rows
. . exported "HR"."JOBS"                         6.609 KB      19 rows
. . exported "HR"."JOB_HISTORY"                  6.585 KB      10 rows
. . exported "HR"."LOCATIONS"                    7.710 KB      23 rows
. . exported "HR"."REGIONS"                      5.296 KB       4 rows
Master table "SYSTEM"."SYS_EXPORT_SCHEMA_01" successfully loaded/unloaded
******************************************************************************
*******
Dump file set for SYSTEM.SYS_EXPORT_SCHEMA_01 is:
  C:\ORACLEXE\APP\TMP\SCHEMA.DMP
Job "SYSTEM"."SYS_EXPORT_SCHEMA_01" successfully completed at 11:48:46
```

В каталог dmpdir будут записаны файлы schema.dmp и expschema.log.

11. Выполните импорт файла дампа schema.dmp в другую схему, на этот раз в HRDEV. Вы воспользуетесь параметром REMAP_SCHEMA для указания того, что объекты должны быть импортированы в схему, отличную от исходной. Так как учетной записи HRDEV еще не существует, то в процессе импорта она будет создана автоматически. В этом примере вы импортируете все за исключением ограничений, ссылочных ограничений и индексов. Если таблица уже существует, то она будет заменена таблицей из файла экспорта.

    В приглашении командной строки операционной системы введите следующую команду:

    ```
    impdp SYSTEM/password SCHEMAS=hr DIRECTORY=dmpdir
    DUMPFILE=schema.dmp
      REMAP_SCHEMA=hr:hrdev EXCLUDE=constraint, ref_constraint, index
      TABLE_EXISTS_ACTION=replace LOGFILE=impschema.log
    ```

    где *password* - это пароль пользователя SYSTEM.

    В процессе операции импорта будут появляться сообщения, аналогичные следующим (эти сообщения также будут сохранены в файле impschema.log в каталоге dmpdir):

    ```
    Import: Release 10.2.0.1.0 - Production on Tuesday, 13 December, 2005 11:49:29

    Copyright (c) 2003, 2005, Oracle.  All rights reserved.

    Connected to: Oracle Database 10g Express Edition Release 10.2.0.1.0 - Production
    Master table "SYSTEM"."SYS_IMPORT_SCHEMA_01" successfully loaded/unloaded
    Starting "SYSTEM"."SYS_IMPORT_SCHEMA_01":  SYSTEM/******** SCHEMAS=hr
      DIRECTORY=dmpdir DUMPFILE=schema.dmp REMAP_SCHEMA=hr:hrdev
      EXCLUDE=constraint, ref_constraint, index TABLE_EXISTS_ACTION=replace
    LOGFILE=impschema.log
    Processing object type SCHEMA_EXPORT/USER
    Processing object type SCHEMA_EXPORT/SYSTEM_GRANT
    Processing object type SCHEMA_EXPORT/ROLE_GRANT
    Processing object type SCHEMA_EXPORT/DEFAULT_ROLE
    Processing object type SCHEMA_EXPORT/TABLESPACE_QUOTA
    Processing object type SCHEMA_EXPORT/PRE_SCHEMA/PROCACT_SCHEMA
    Processing object type SCHEMA_EXPORT/SEQUENCE/SEQUENCE
    Processing object type SCHEMA_EXPORT/TABLE/TABLE
    Processing object type SCHEMA_EXPORT/TABLE/TABLE_DATA
    . . imported "HRDEV"."COUNTRIES"                    6.093 KB      25 rows
    . . imported "HRDEV"."DEPARTMENTS"                  6.640 KB      27 rows
    . . imported "HRDEV"."EMPLOYEES"                    15.77 KB     107 rows
    . . imported "HRDEV"."JOBS"                         6.609 KB      19 rows
    . . imported "HRDEV"."JOB_HISTORY"                  6.585 KB      10 rows
    . . imported "HRDEV"."LOCATIONS"                    7.710 KB      23 rows
    . . imported "HRDEV"."REGIONS"                      5.296 KB       4 rows
    ```

Processing object type SCHEMA_EXPORT/TABLE/COMMENT
Processing object type SCHEMA_EXPORT/PROCEDURE/PROCEDURE
Processing object type SCHEMA_EXPORT/PROCEDURE/ALTER_PROCEDURE
Processing object type SCHEMA_EXPORT/VIEW/VIEW
Processing object type SCHEMA_EXPORT/TABLE/TRIGGER
Processing object type
SCHEMA_EXPORT/TABLE/STATISTICS/TABLE_STATISTICS
Job "SYSTEM"."SYS_IMPORT_SCHEMA_01" successfully completed at 11:49:49

Теперь схема HRDEV пополнится данными из схемы HR.

12. Присвойте пароль только что созданной пользовательской учетной записи HRDEV. Чтобы сделать это, запустите командную строку SQL и подключитесь под пользователем SYSTEM (как вы это делали на этапе 4), после чего в приглашении SQL введите следующую команду ALTER USER:
13. ALTER USER hrdev IDENTIFIED BY hrdev;
14.

Этой командой будет назначен пароль hrdev.

Теперь вы можете работать в схеме HRDEV, не затрагивая данных в рабочей схеме HR.