**Тема 1 — Введение**

**1-2 Общие сведения о базах данных**

o Различия между данными и информацией

*Данными* называют собранные факты по какому-то вопросу или теме. А *информацией* является результат объединения, сравнения и выполнения расчетов с данными. Например, мы вводим в базу данных данные о бюджете, что $1 000 000 в 2015г., а из базы данных мы получаем информацию: «$1 000 000 бюджет на 2015 год».

o Определение базы данных

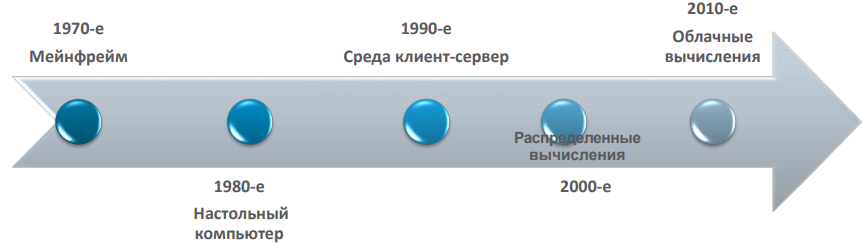
*База данных* – это централизованный структурированный набор данных, хранящихся в компьютерной системе. Она предоставляет средства для извлечения, добавления, изменения и удаления данных по требованию. База данных предоставляет средства для преобразования извлеченных данных в полезную информацию.

o Описание элементов системы управления базами данных (DBMS)

Система управления базами данных (DBMS) – это программное обеспечение, которое контролирует хранение, организацию и извлечение данных. Элементами DBMS являются:

* Управление памятью и хранилищем;
* Словарь данных («центральное хранилище информации о данных»);
* Язык запросов (искусственный язык, на котором делаются запросы к базам данных и информационно-поисковым системам).

o Преобразования в сфере компьютерных технологий



*1970-е: мейнфреймы (централизованная обработка):*

В 1970-е годы совершались попытки построить системы баз данных, интегрируя аппаратное и программное обеспечение. Небольшие компьютеры или «терминалы ввода-вывода» использовались для доступа к мощным мейнфреймам и выполнения команд. Эти терминалы зависели от мейнфрейма и отображали результаты только тогда, когда мейнфрейм завершал обработку. Сами они не обладали значимыми вычислительными мощностями для обработки данных.

*1980-е: настольные компьютеры (локализованная обработка):*

Обработка данных переместилась с мейнфреймов на клиентские компьютеры. ПК, имеющие собственное программное обеспечение и самостоятельно выполняющие некоторую обработку, стали известны как «смарт-клиенты» или «рабочие станции». Вычислительная мощность клиентской машины привела к появлению приложений с графическим интерфейсом пользователя. В эту эпоху были созданы многие приложения, широко распространенные сегодня (Word, Excel, PowerPoint).

*1990-е: среда «клиент-сервер» (централизованная и локальная обработка):*

В среде «клиент-сервер» используется Интернет и быстродействующие серверы для удовлетворения потребностей организаций в хранении данных и получении информации. Программное обеспечении, которое управляет данными, находится на сервере баз данных и выполняет операции по хранению и извлечению данных. Приложения для бизнес-операций находится на сервере приложений и служат для создания документов, разработки, взаимодействия или выполнения операций с данными. Клиенты могут иметь собственные приложения, но доступ к основным бизнес-приложениям осуществляется с клиентом с помощью интернет-браузера.

*2000-е: распределенные вычисления (совместная обработка):*

В модели распределенных вычислений все компьютеры в организации, находящиеся в разных местах, могли использоваться как пул вычислительных ресурсов. В грид-системах создается программная инфраструктура, которая может работать на большом числе сетевых серверов. Пользователи со своих рабочих станций отправляют запросы, чтобы получить информацию или выполнить вычисления, и эти запросы максимально эффективно обрабатываются на компьютерах в гриде-системе.

*2010-е: облачные вычисления (обработка в Интернете):*

Облачные вычисления позволяют предоставлять вычислительные услуги через Интернет. Три основные категории облачных услуг:

* IaaS – позволяет вам арендовать серверы, хранилища, операционные системе и т.д. в облачной среде.
* PaaS – предоставление доступа к онлайн-среде для разработки и тестирования ПО без затрат на установку или управление.
* SaaS – предоставление ПО напрямую из Интернета. Пользователи обычно получают к нему доступ, используя веб-браузер.

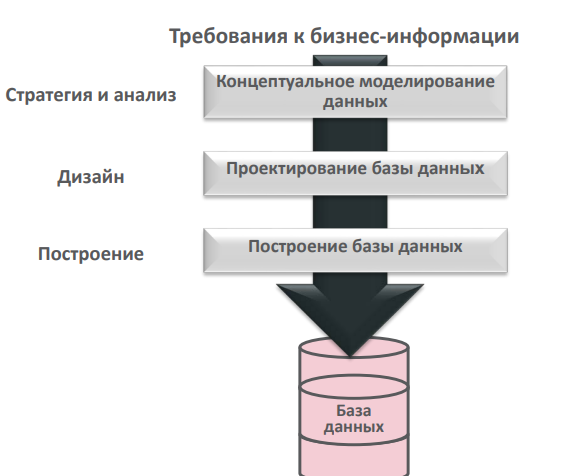
o Определение бизнес-сценариев и отраслевых примеров применения баз данных

Школы и колледжи используют базы данных, чтобы хранить информацию об учебных курсах, учащихся и преподавателях. Банки используют базы данных для хранения информации о клиентах, счетах, займах и транзакциях. Авиакомпании и железные дороги используют онлайн-базы данных для бронирования билетов и для отображения информации о расписании. Телекоммуникационные подразделения хранят в базах данных информацию о коммуникационных сетях, номера телефонов, сведения о вызовах и месячных счетах. В финансовой сфере и торговле базы данных используются для хранения информации, касающейся продаж и покупки акций и облигаций или интернет-трейдинга. Организации используют базы данных для хранения информации о своих сотрудниках, зарплатах, премиях, налогах и создания платежных чеков.

**1-3 Типы моделей баз данных**

o Описание процесса разработки базы данных

Процесс разработки базы данных:



o Описание основных типов моделей баз данных:

* Модель плоских файлов

Базы данных на основе плоских файлов проектируются при использовании одной таблицы. Базы данных на основе плоских файлов, как правило, имеют текстовый формат, где в каждой стоке содержится только одна запись. Поля в записи отделяются друг от друга разделителями, например знаками табуляции и запятыми.

* Иерархическая модель

В иерархической модели данные организованы в виде древовидной структуры. Данные хранятся как записи, и между ними существуют связи. Запись является набором полей. Запись в иерархической модели БД соответствует строке в реляционной модели БД.

* Сетевая модель

Сетевая модель – это модель БД, которая обеспечивает гибкий способ представления объектов и их связей. Сетевая база данных состоит из набора записей, между которыми существуют связи (прямоугольники = поля, линии = связи). Каждая запись представляет собой набор полей, каждое из полей содержит только одно значение данных. Связь указывает на ассоциацию двух записей.

* Объектно-ориентированная модель

Объект моделируется в виде объекта. Для каждого объекта определено состояние и поведение. Связь между объектами осуществляется путем предоставления доступа. Объект должен принадлежать только к одному классу, как экземпляр этого класса. Вы можете наследовать новый класс (подкласс) из существующего класса (суперкласса).

* Реляционная модель

Данные предоставлены в виде набора таблиц. Каждый столбец представляет атрибуты, относящиеся к таблице. Каждая строка представляет собой экземпляр таблицы. Каждая таблица может быть визуально представлена в виде столбцов и строк. Каждая таблица имеет поле или набор полей, которые однозначно идентифицируют строку. Порядок строк и столбцов не имеет значения. Каждая строка уникальна. Каждое поле может содержать только одно значение. Значения в столбце или поле из одного и того же домена (типа данных). Имена таблиц должны быть уникальными. Имена столбцов в каждой таблице должны быть уникальными.

**Тема 2 — Базы данных и моделирование данных**

**2-1 Реляционные базы данных**

o Описание функций отдельной таблицы

Основной и ключевой объект реляционной базы данных это – *таблица*. Базы данных на основе плоских файлов представляют собой тип баз данных, где данные хранятся в одной таблице. БД на основе плоских файлов, как правило, имеют текстовый формат, где в каждой строке содержится только одна запись. У таких баз данных есть свои преимущества – это простота понимания, простота реализации, простота извлечения информации, все записи хранятся в одном месте, простота сортировки и фильтрации записей, низкие требования к аппаратному и программному обеспечению. Но при всем этом есть и недостатки: низкая безопасность, несогласованность данных, избыточность данных, обременительное предоставление информации, медленные базы данных огромного объема.

o Описание функций и правил реляционной базы данных

Реляционная база данных представляет информацию в таблицах со строками и столбцами. Каждый столбец представляет конкретный тип информации, и каждая строка содержит одну запись. Таблицы ссылаются друг на друга с помощью общего поля. Уникальное поле, называемое *ключом*, используется для идентификации каждой записи в реляционной базе данных.

o Описание преимуществ и недостатков типов баз данных

Если говорить про преимущества – это меньше избыточность, мы можем один раз описать объект, и там в действительности будет это отображено. Мы исключаем несогласованность данных, увеличиваем эффективность за счет того, что данные у нас хранятся один раз. Гарантируется также целостность данных, и вопрос о конфиденциальности существенно выше. Но у нее есть и недостатки – это медленный доступ к данным и трудоемкость разработки.

o Описание реляционных таблиц и основных понятий

Реляционная таблица представляет собой простую структуру, в которой осуществляется организация и хранение данных. Каждая таблица должна иметь особое имя, значение для однозначной идентификации строк, и таблица может содержать несколько строк. Каждый столбец должен иметь уникальное имя. А записи в столбцах должны быть однозначными и должны быть одинакового типа. Порядок строк и столбцов не имеет значения.

Таблица – базовая структура для хранения данных. Столбец – атрибут, который описывает информацию в таблице. Первичный ключ – уникальный идентификатор для каждой строки. Внешний ключ – столбец, который ссылается на столбец первичного ключа в другой таблице. Строка – данные для одного экземпляра таблицы. Поле – одно значение на пересечении строки и столбца.

**2-2 Концептуальные и физические модели данных**

o Описание концептуальной модели данных

Концептуальная модель отражает функциональные и информационные потребности бизнеса, также она удовлетворяет потребности бизнеса, но не отвечает их реализации. Концептуальная модель основана на текущих потребностях, но может отражать будущие потребности. Она определяет важные объекты, которые станут таблицами в БД, и связи между объектами. Но она не определяет атрибуты, которые станут столбцами или полями в БД, и уникальные идентификаторы – атрибуты, которые станут первичными ключами в БД.

o Описание логической модели данных

*Логическая модель* – это модель связей между объектами (ERM), ведь она содержит все объекты и связи между ними. Она иллюстрируется ERD-диаграммой. Логическая модель указывает все атрибуты и уникальные идентификаторы (UID) для каждого объекта, определяет обязательность атрибутов и обязательность и кардинальность связей.

o Описание физической модели данных

*Физическая модель* является расширением логической модели данных. Она указывает определения таблиц, типы данных и точность, определяет представления, индексы и прочие объекты БД. Физическая модель описывает, как должны быть реализованы объекты в конкретной базе данных. Она показывает все структуры таблиц, включая столбцы, первичные ключи и внешние ключи.

Шаги для создания физической модели данных:

1. Моделирование объектов как таблиц
2. Моделирование связей как внешних ключей
3. Моделирование атрибутов как столбцов
4. Изменение модели данных с учетом физических ограничений и требований

o Анализ сходств и различий между концептуальными и физическими моделями данных

*Моделирование данных* – это процесс сбора важных концепций и правил, которые характеризуют бизнес, и их визуальное представление на диаграмме.

Чтобы примерно показать разницу у концептуальной и физической модели, рассмотрим следующее предложение. Мечты клиента (концептуальная модель) становятся физической реальностью (физическая модель).

В процессе моделирования физической модели из концептуальной, объекты сменяются на таблицы и связи на внешние ключи.

**2-3 Объекты и атрибуты**

o Определение объектов

*Объектом* можно назвать информацию, которую требуется отслеживать, а также можно назвать наименование предметов, которые вы можете перечислить. Существует несколько типов объектов:

1. Первичный (существует независимо);
2. Характеристика (существует благодаря другому объекту);
3. Пересечение (существует благодаря двум или более объектам).

Объекты содержат экземпляры. Экземпляр объекта – одно вхождение объекта. Объекты представляют набор экземпляров, представляющих интерес для конкретной организации.

o Определение атрибутов

Атрибуты описывают сущности и представляют собой конкретную информацию, которая должна быть известна. Атрибуты служат однозначными характеристиками объекта. Атрибуты изображаются в ERD-диаграмме в пределах прямоугольника, определяющего объект. Имена атрибутов должны быть существительными в единственном числе, содержащими строчные и прописные буквы. В большинстве случаев имя атрибута не должно содержать в себе имя объекта, потому что атрибуты указываются с именем объекта.

o Определение обязательных, необязательных, временных и постоянных атрибутов

Атрибуты классифицируются на два типа: обязательные (не допускается значение null; обозначаются символом \*) и необязательные (допускается значение null; обозначаются строчной буквой o).

Бывают также атрибуты временные и постоянные. Временные атрибуты – нестабильные (пример: Age), а постоянные – стабильные (пример: Birth Date).

Еще атрибуты делятся на простые (атомарные) и составные. Простые атрибуты – это атрибуты, которые нельзя разделить на компоненты. А составные атрибуты – это атрибуты, которые можно разделить на более мелкие компоненты, представляющие собой базовые атрибуты с самостоятельными значениями.

Также атрибуты делятся на однозначные и многозначные. Однозначные атрибуты могут иметь единственное значение в определенный момент времени (пример: Student Last Name). Многозначные атрибуты могут иметь более одного значения в определенный момент времени (пример: Address).

o Описание обозначений Баркера, Бахмана и обозначений информационной инженерии

Система обозначения Баркера:

1. Правила представления объектов:

Объект представляется в виде прямоугольника с закругленными углами. Следует присвоить объекту имя и поместить это имя в левом верхнем углу прямоугольника данного объекта. Имя объекта должно быть в верхнем регистре и должно быть существительным в единственном числе.

1. Правила представления атрибутов:

Имена атрибутов должны быть понятны всем, а не только разработчикам. Первая буква каждого слова атрибута должна быть прописной, а остальные буквы – строчными. Рядом с каждым атрибутом следует указать символ, представляющий тип атрибута (\* - обязательный; о – необязательный; # - UID).

1. Правила представления связей:

Связь может существовать максимум между двумя объектами. Может существовать связь объекта с самим собой (рекурсивная). Связь имеет два направления. Оба направления связи должны быть отмечены.

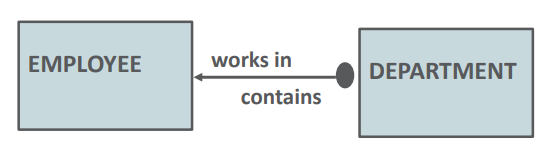
Система обозначения Бахмана:

1. Объект представлен прямоугольником
2. Атрибуты

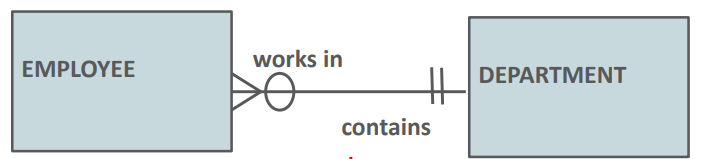
Изображение выглядит как текст

Автоматически созданное описание

1. Линии связей

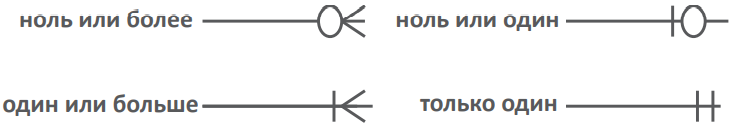


Система обозначений в информационной инженерии:



Любой сотрудник (EMPLOYEE) работает только в одном отделе (DEPARTMENT).

Любой отдел (DEPARTMENT) содержит ноль или больше сотрудников (EMPLOYEES).



Обозначения в модели данных:

Изображение выглядит как стол

Автоматически созданное описание

**2-4 Уникальные идентификаторы**

o Определение уникальных идентификаторов (UID)

*Уникальный идентификатор (UID)* — это атрибут объекта, который соответствует следующим правилам:

– Он уникален среди всех экземпляров данного объекта.

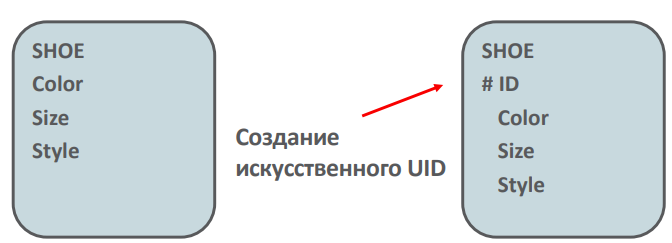
– Его значение не равно NULL для каждого экземпляра объекта в течение времени существования экземпляра.

– Его значение не изменяется в течение времени существования экземпляра.

*UID* — специальный атрибут или группа атрибутов, которая однозначно идентифицирует конкретный экземпляр объекта.

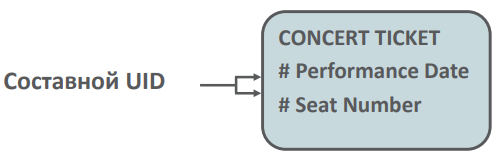
o Определение искусственных уникальных идентификаторов

Искусственный UID создается из данных, которые присваивает или генерирует система. Искусственные UID не встречаются в реальном мире, а создаются для идентификации в системе.



o Определение составных уникальных идентификаторов

UID из одного атрибута является *простым UID*. Иногда одного атрибута недостаточно, чтобы однозначно идентифицировать экземпляр объекта. Если UID представляет собой комбинацию атрибутов, его называют *составным UID*.



o Определение потенциальных и дополнительных уникальных идентификаторов

Объект может иметь более одного UID.

Изображение выглядит как текст

Автоматически созданное описание

Потенциальные уникальные идентификаторы на примере:

– Номер бейджа (Badge number)

– Номер платежной ведомости (Payroll number)

Только один из потенциальных UID можно выбрать в качестве первичного UID. Остальные потенциальные UID называются вторичными UID.

o Определение первичных ключей

UID становится первичным ключом, когда логическая модель преобразуется в физическую базу данных. *Первичный ключ (PK)* — это столбец или набор столбцов, которые однозначно идентифицируют каждую строку в таблице. Он не может содержать значения null. *PK* — это либо существующий столбец таблицы, либо столбец, который специально генерируется базой данных в соответствии с определенной последовательностью. Он должен содержать уникальное значение для каждой строки данных.

**2-5 Связи**

o Определение и распознавание примеров связей и соответствующих внешних ключей

Связь представляет собой двунаправленную значимую ассоциацию между двумя объектами или объекта с самим собой. Связи представляют собой ассоциацию между двумя и более объектами. Линия связи на диаграмме обозначается либо сплошной (обязательная), либо пунктирной линией (необязательная). Эти линии заканчиваются «черточкой» (один экземпляр) или «птичьей лапкой» (один или более экземпляров). Связи должны иметь имена, которые помогают описать связь между объектами.

Связи в концептуальной модели данных сопоставляются с внешними ключами в таблице физической базы данных. *Внешний ключ (FK)* — это столбец или комбинация столбцов в одной таблице, которая ссылается на первичный ключ в той же таблице или в другой таблице.

Компонентами связи являются:

– Имя: метка, которая появляется рядом с объектом, для которого она назначена. Имена связи должны быть указаны в нижнем регистре.

– Кардинальность: минимальное и максимальное число значений в связи.

– Обязательность: должна ли существовать связь.

o Определение обязательности связей

Обязательность связей заключается в том, что определяют должна ли существовать связь. Связи бывают необязательные и обязательные. В системе обозначений Баркера обязательность помечается сплошной линией, а необязательность пунктирной линией. В прочтении бизнес-правил используется следующие фразы: необязательно – «может быть» или «может»; обязательно – «должно быть» или «должно».

o Определение кардинальности связей

Кардинальность связей служит мерой количества чего-либо. В связи она определяет степень, с которой один объект связан с другим, отвечая на вопрос «Сколько?». Бывают либо одна и только одна соответствующая запись, либо одна или более соответствующих записей. В системе обозначений Баркера минимальное значение отображается простой линией, а максимальное – птичьей лапкой. В прочтении бизнес-правил используется следующие фразы: линия – «один и только один»; птичья лапка – «один или более».

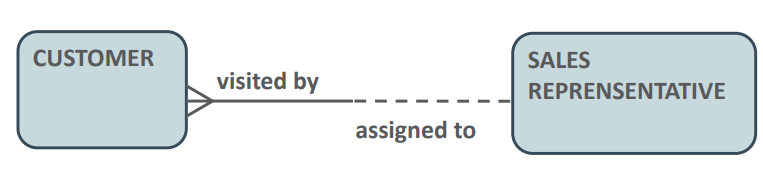
И общий синтаксис бизнес-правил выглядит следующим образом: Каждый объект entity1 {должен быть или может быть} имя связи {«один или более» или «один и только один»} объект entity2.

o Типы связей

Все связи представляют собой информационные требования и правила ведения бизнеса. Типы связей:

* «Многие к одному» (M:1) или «один ко многим» (1:M)

Связи «многие к одному» и «один ко многим» (M:1 и 1:M) имеют кардинальность «один или более» и «один и только один» в другом направлении.



Бизнес-правила:

– Каждого покупатель (CUSTOMER) должен посетить один и только один продавец-консультант (SALES REPRESENTATIVE).

– Каждый продавец-консультант (SALES REPRESENTATIVE) может быть назначен одному или более покупателям (CUSTOMER)

* «Многие ко многим» (M:M)

Связи «многие ко многим» (M:M) имеют кардинальность «один или более» в обоих направлениях.



Бизнес-правила:

– Каждому сотруднику (EMPLOYEE) может быть назначена одна или более работ (JOB).

– Каждую работу (JOB) может выполнять один или более сотрудников (EMPLOYEE).

* «Один к одному» (1:1)

Связи «один к одному» (1:1) имеют кардинальность «один и только один» в обоих направлениях.

Изображение выглядит как стол

Автоматически созданное описание

Бизнес-правила:

– Каждый компьютер (COMPUTER) должен содержать одну и только одну материнскую плату (MOTHERBOARD).

– Каждая материнская плата (MOTHERBOARD) должна содержаться в одном и только одном компьютере (COMPUTER).

* Рекурсивные

*Рекурсивные связи* — это связи объекта с самим собой.

Изображение выглядит как текст

Автоматически созданное описание

Бизнес-правила:

– Каждый сотрудник (EMPLOYEE) может руководить одним или более сотрудниками (EMPLOYEE).

– Каждым сотрудником (EMPLOYEE) должен руководить один и только один сотрудник (EMPLOYEE).

o Матрица связей

Матрицу связей можно использовать для сбора первоначальной информации о связях между объектами. Матрица связей имеет следующие характеристики:

– Матрица связей показывает, состоит ли объект строки, расположенный слева, в каких-либо отношениях с объектом столбца в верхней части матрицы.

– Все объекты перечислены в левой и верхней частях матрицы.

– Если объект строки связан с объектом столбца, имя этой связи отображается в области пересечения.

– Если объект строки не связан с объектом столбца, область пересечения пуста.

– Каждая связь выше диагональной линии является обратной (служит зеркальным отражением) связи ниже диагональной линии.

– Рекурсивные связи представлены полями на диагонали.

**2-6 Моделирование связей между объектами (ERD-диаграммы)**

o Описание моделирования данных

*Моделирование данных* – это процесс сбора важных концепций и правил, которые характеризуют бизнес, и их визуальное представление на диаграмме.

Концептуальная модель базы данных может использоваться для дальнейших обсуждений с разработчиками, администраторами баз данных и разработчиками приложений. Важность концептуальной модели для бизнеса обусловлена тем, что она:

– Точно описывает информационные потребности бизнеса

– Способствует обсуждению

– Предотвращает ошибки и неправильное понимание

– Формирует прочную основу для разработки физической базы данных

– Документирует процессы (также называемые бизнес-правилами) деловой деятельности

– Учитывает правила и законы, регулирующие эту отрасль

Концептуальная модель представляет собой формальную модель, которая:

– Описывает предметы, которые важны для организации (объекты).

– Определяет связи самого высокого уровня между различными объектами, но может содержать или не содержать сведения о кардинальности и допустимости неопределенного значения (null).

– Не определяет атрибуты или уникальный идентификатор для каждого объекта.

o Объяснение понятия «без реализации» (implementation-free) в отношении разработки моделей данных и

проектов баз данных

Логическая модель данных:

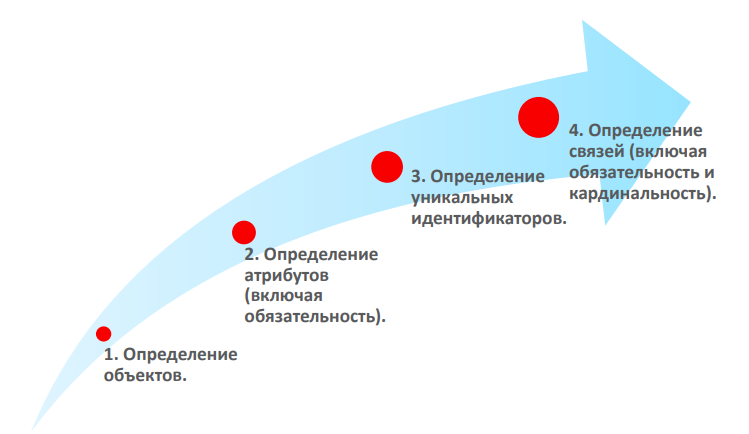
– Описывает данные максимально подробно, независимо от того, как это будет физически реализовано в базе данных.

– Обычно создается на базе концептуальной модели данных.

– Содержит все объекты, атрибуты, UID и связи, а также определяет обязательность и кардинальность этих элементов.

Логическая модель иллюстрируется с помощью ERD.

Этапы создания:



Хорошая логическая модель данных остается неизменной независимо от типа системы базы данных, где в итоге она будет реализована. Именно это означает термин «модель, не зависящая от реализации» (implementation-free).

o Перечисление четырех целей моделирования связей между объектами

Четыре цели моделирования связей между объектами:

• Собрать всю необходимую информацию

• Убедиться, что информация отображается только один раз

• Не тратить время на моделирование информации, производной от других, уже смоделированных данных

• Представить информацию в предсказуемой логической форме

o Определение диаграммы «объект-связь» (ERD)

*ERD* — модель, которая определяет понятия или объекты, существующие в системе, а также связи между этими объектами. Она преследует несколько целей:

– Аналитик/проектировщик базы данных в процессе построения ERD лучше понимает, какая информация должна содержаться в базе данных.

– Служит средством документирования.

– Используется для информирования пользователей о логической структуре базы данных. В частности, она эффективно информирует пользователей о логике базы данных.

ERD может использоваться для представления требований организации к данным независимо от типа используемой базы данных и даже при ее отсутствии.

o Сопоставление связей с помощью ERD-инструкций

*ERD-инструкции* — это словарь, который используется для четкого описания бизнес-правил, которые указаны в ERD. Используйте язык ERD-инструкций, чтобы установить связи между объектами в ERD.

o Создание компонентов ERD, представляющих собой объекты и атрибуты, в соответствии с правилами

построения диаграмм

Просто разбейте каждую ERD-инструкцию на ее компоненты.

Компоненты ERD-инструкций:

• EACH

• Entity A

• OPTIONALITY (должен быть/может быть)

• RELATIONSHIP NAME

• CARDINALITY (один и только один/один или более)

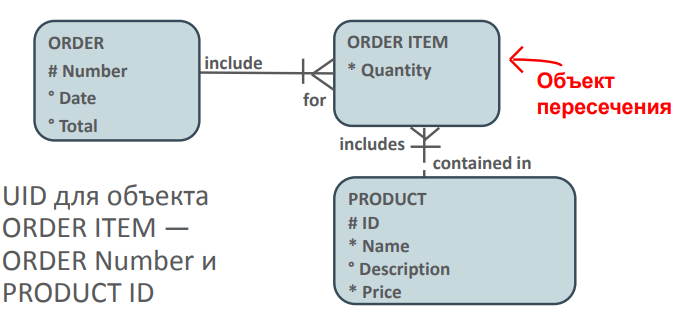
• Entity B

**Тема 3 — Уточнение модели данных**

**3-1 Дополнительные действия со связями**

o Решение связей типа M:M

Атрибуты описывают только объекты. Если атрибуты описывают связь, значит, эту связь необходимо решить. Решение связи M:M с новым объектом пересечения и двумя ассоциативными связями 1:M.



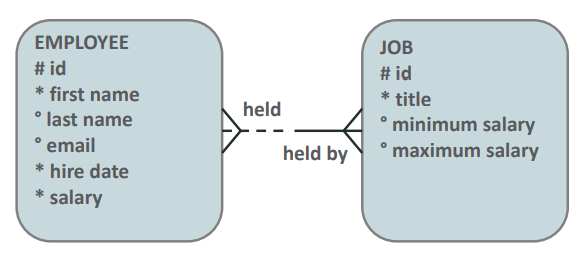
*Характеристики объекта пересечения:*

• Связи из объекта пересечения всегда обязательные.

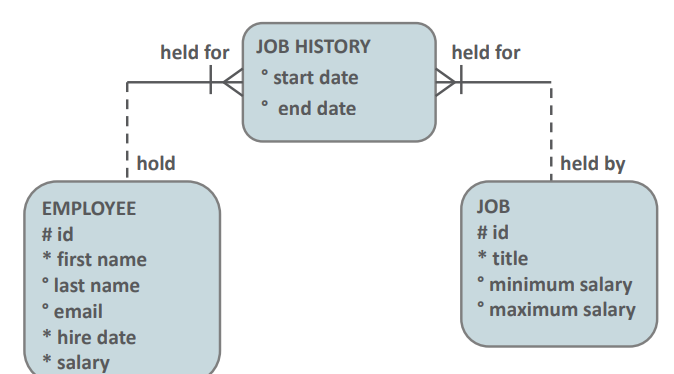
• Объекты пересечения обычно содержат расходные материалы, например использованное количество и даты. Обычно это временные объекты большого объема.

• Объект пересечения определяется по своим двум исходным связям (определяющим связям).

Решение связей M:M с помощью объекта пересечения с ассоциативными связями

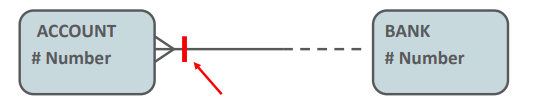


Объекты EMPLOYEE и JOB не хранят историю работ сотрудника.



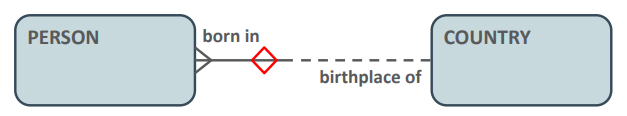
o Определение ассоциативных связей

У ассоциативных связей один объект заимствует как дополнительный уникальный идентификатор один из UID у другого объекта. Ассоциативная связь отображается как параллельная черточка на связи перед заимствующим объектом, и связь должна быть обязательная. Пример:



o Определение и примеры непередаваемых связей

*Перемещаемость* — это возможность связи между двумя экземплярами объекта меняться со временем. Неперемещаемую связь невозможно переместить между объектами, которые она соединяет. Неперемещаемая связь обозначается ромбом. Неперемещаемые связи могут быть только обязательными. Например, страна рождения для человека представляет собой неперемещаемую связь.



o Определение и построение объектов с супертипом и подтипом

Супертип имеет связь «родитель-потомок» с одним или несколькими подтипами. *Подтип* — это подгруппировка объекта в типе объекта, имеющая атрибуты, которые отличаются от атрибутов в других подгруппировках. Каждый подтип является конкретизацией супертипа, и поэтому должен быть заключен внутри объекта. Общие атрибуты и связи для всех подтипов должны быть перечислены только в супертипе, но они наследуются в каждом подтипе. Подтип может и, как правило, имеет собственные атрибуты и связи. Не может быть только одного подтипа; необходимо создать другой подтип, содержащий все остальное.

Пример:Изображение выглядит как текст

Автоматически созданное описание

Характеристика подтипа:

– Наследует все атрибуты супертипа.

– Наследует все связи супертипа.

– Обычно имеет собственные атрибуты или связи.

– Извлекается внутри супертипа.

– Никогда не существует один.

– Может иметь собственные подтипы.

– Имеет одинаковые первичные ключи супертипа и подтипа.

*Правильное определение подтипов:*

• Является ли этот подтип разновидностью супертипа?

• Учтены все возможные случаи? (исчерпывающий)

Все экземпляры супертипа также являются экземплярами одного из подтипов. Для категоризации экземпляров, которые не определяются одним из существующих подтипов необходимо добавить подтип OTHER.

• Каждый экземпляр соответствует только одному подтипу? (взаимоисключающий)

Каждый экземпляр супертипа принадлежит только одному подтипу.

o Определение иерархических, рекурсивных и дуговых связей

Моделирование иерархических данных:

Изображение выглядит как текст, внешний, знак, снимок экрана

Автоматически созданное описание Один объект включает в себя один объект, а другой – в этот объект и т.д. Связь между такими должна быть от одной обязательная и с птичьей лапкой. У каждого такого объекта должна быть связь намертво с предыдущим объектом.

*Иерархическая модель данных* – это модель данных, где используется представление базы данных в виде древовидной структуры, состоящей из объектов различных уровней.

*Рекурсивные связи* – это связь, при которой экземпляр объекта связан с другим экземпляром в этом же объекте. Она всегда моделируется циклом.

Изображение выглядит как текст

Автоматически созданное описание

**Дуговые связи:**

Дугой обводится группа эксклюзивных связей — в ней может существовать только одна из связей для любого экземпляра объекта. Дуга, нарисованная между двумя связями, соединяет их и демонстрирует взаимоисключение. Такая связь предполагает наличие условия «или». Дуга означает, что любой экземпляр этого объекта в заданный момент времени может иметь только одну действительную связь внутри дуги. Связи, включенные в дугу, обозначены кружком на линии дуги связи.

Характеристики дуговой связи:

– Связи в дуге часто имеют одинаковое название.

– Связи в дуге должны быть либо все обязательные, либо все необязательные, и должны иметь одинаковую кардинальность.

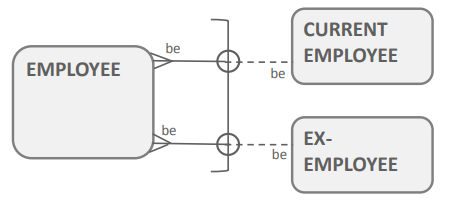
– Дуга принадлежит одному объекту и должна содержать только те связи, которые исходят из него.

– Объект может иметь несколько дуг, но каждая связь может быть частью только одной дуги.

– Дуговая связь представлена дугообразной линией, соединяющей две или более линий связи.

Объект-супертип и его подтипы можно моделировать как дуговую связь.

Пример: объект EMPLOYEE является либо CURRENT EMPLOYEE, либо EX-EMPLOYEE, но не может быть и тем, и другим одновременно (это также можно смоделировать как супертип/подтип).



**3-2 Отслеживание изменений данных**

o Отслеживание изменений данных с течением времени

Моделирование данных на протяжении времени:

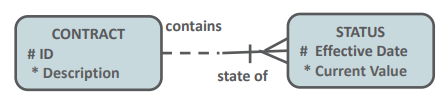
* Добавление исторических данных

Большинству компаний необходимо отслеживать исторические данные, чтобы выявлять тенденции и шаблоны, которые могут послужить основой для бизнес-инноваций или улучшения процессов. Каждое обновление атрибута или перемещение связи означает потерю информации. Часто информация уже бесполезна, но некоторые системы должны продолжать отслеживать некоторые или все исторические значения атрибута. Кроме того, некоторые системы должны хранить журнал аудита для каждой транзакции. Нужно добавить в модель объекты и связи для размещения исторических данных.

* Отслеживание значений атрибута

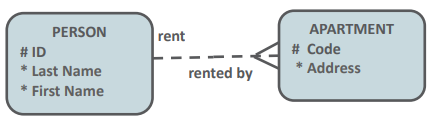
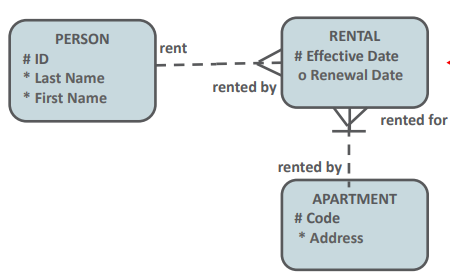
Нужно создать дополнительные объекты, чтобы отслеживать значения атрибутов на протяжение времени.

Изображение выглядит как текст

Автоматически созданное описание> 

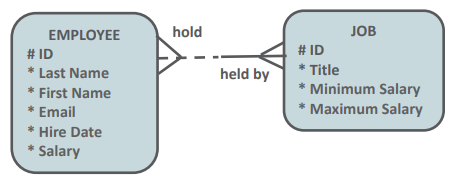
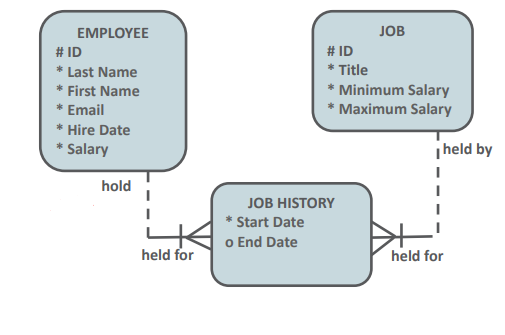
* Изменение связи

Нужно добавить объект для добавления связи, которая может со временем измениться.

> 

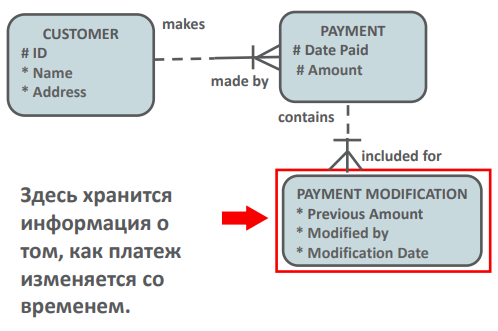
* Создание объектов пересечения

Решение связей M:M с помощью объекта пересечения часто используется для отслеживания информации о связи, которая меняется со временем.

> 

* Ведение журнала измененной информации

Дополнительные объекты и атрибуты хранят измененную информацию.



**3-3 Нормализация и бизнес-правила**

o Объяснение нормализации

*Нормализация* — концепция реляционной базы данных, но ее принципы применимы к моделированию данных. Целью является нормализация данных до 3NF перед преобразованием модели в реляционный проект.

o Описание нормальных форм

Основные типы нормальных форм по Э.Ф. Кодду:

• Первая нормальная форма (1NF)

Для первой нормальной формы необходимо наличие атрибутов без нескольких значений. Чтобы проверить пригодность для 1NF, нужно проверить, что каждый атрибут имеет только одно значение для каждого экземпляра объекта.

• Вторая нормальная форма (2NF)

Вторая нормальная форма (2NF) требует, чтобы любой атрибут, не являющийся UID, зависел (был свойством или характеристикой) от всего UID. Если UID составной, то каждый атрибут должен зависеть от всех частей составного UID.

• Третья нормальная форма (3NF)

Правило третьей нормальной формы (3NF) гласит, что никакой атрибут, не являющийся UID, не может зависеть от другого атрибута, не являющегося UID. Третья нормальная форма запрещает транзитивные зависимости. Транзитивная зависимость существует в том случае, когда какой-либо атрибут в объекте зависит от другого атрибута, не являющегося UID, в этом объекте.

o Проверка данных с использованием нормализации

Первая нормальная форма (1NF) - Если атрибут имеет несколько значений, создайте дополнительный объект и свяжите его с исходным с помощью связи 1:M. Каждый атрибут должен иметь только одно значение для каждого экземпляра объекта.

Вторая нормальная форма (2NF) - Если атрибут не зависит от всего UID, нужно создать дополнительный объект с частичным UID. Атрибут должен зависеть от полного UID его объекта.

Третья нормальная форма (3NF) - Необходимо переместить в новый объект все атрибуты, не являющиеся UID, которые зависят от другого атрибута, не являющегося UID. Каждый атрибут зависит только от UID своего объекта. Нужно переместить в новый объект все атрибуты, не являющиеся UID, которые зависят от другого атрибута, не являющегося UID.

o Описание бизнес-правил

*Бизнес-правило* — это высказывание, которое определяет или ограничивает какие-либо аспекты бизнеса. Оно используется для определения объектов, атрибутов, связей и ограничений. Оно содержит правила и политики, определяющие ведение бизнеса, и определяет повседневные операции. Существует два типа бизнес-правил:

– Структурные

– Процедурные

*Источники бизнес-правил:*

• Руководство и руководители высшего звена

• Письменная документация (процедуры, стандарты, руководства по эксплуатации)

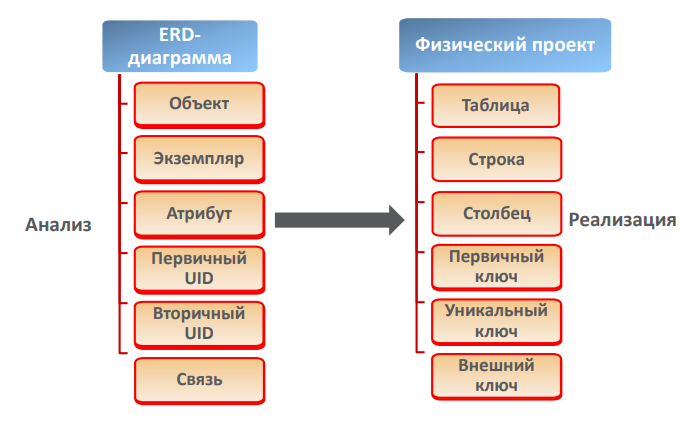
• Прямые собеседования с конечными пользователями

Структурные бизнес-правила обозначают тип информации, которая должна храниться, и взаимодействие элементов информации. Структурное бизнес-правило определяет конкретный статичный аспект бизнеса. Эти правила всегда можно представить в виде диаграммы ERD.

Процедурные правила определяют предварительные требования, шаги, процессы или требования к рабочим процессам. Многие процедурные бизнес-правила связаны со временем: событие A должно произойти до события Б. Некоторые процедурные правила невозможно изобразить в виде диаграммы, но их необходимо задокументировать, чтобы впоследствии запрограммировать.

**3-4 Терминология моделирования данных и сопоставление**

o Применение сопоставления терминологии между логическими и физическими моделями



o Изучение и применение правил именования Oracle для таблиц и столбцов, используемых в физических

моделях

*Диаграмма таблицы* — это дополнительная документация, которая часто используется для более подробного объяснения ключей и столбцов в физической базе данных.

**Правила именования таблиц:**

*Имя таблицы* — это множественное число от имени объекта. Имена столбцов аналогичны именам атрибутов, за тем исключением, что специальные символы и пробелы заменяются на подчеркивания.

Уникальное краткое имя для каждой таблицы полезно при именовании столбцов внешних ключей.

Создайте краткие имена на основе следующего:

– Имена объектов, содержащие более одного слова

– Имена объектов, содержащие одно слово, состоящее более чем из одного слога

– Имена объектов, содержащие один слог, состоящий более чем из одного символа

**Правила именования столбцов:**

Имена столбцов аналогичны именам атрибутов, за тем исключением, что специальные символы и пробелы заменяются на подчеркивания, что было сказано ранее. В именах столбцов часто используется больше сокращений, чем в именах атрибутов.

**Ограничения именования в Oracle:**

Имена таблиц и столбцов:

• Должны начинаться с буквы

• Могут содержать до 30 символов

• Не могут содержать пробелы или специальные символы, например «!» (но «$», «#» и «\_» разрешены)

• Не могут быть «зарезервированными словами» в БД Oracle или SQL

Имена таблиц должны быть уникальными в пределах одной учетной записи пользователя в базе данных Oracle. Имена столбцов в таблице должны быть уникальными.

o Применение правил сопоставления связей для правильного преобразования отношений

Связи сопоставляются между первичными ключами и внешними ключами, чтобы одна таблица могла ссылаться на другую. Связь создает в таблице один или несколько столбцов внешнего ключа на стороне связи «многие». Для именования столбца внешнего ключа используется краткое имя таблицы.

Ассоциативная связь сопоставлена со столбцом внешнего ключа на стороне «многие», как любая другая связь M:1. В этом случае столбец внешнего ключа играет двойную роль, так как он также входит в состав первичного ключа.

Связь M:M решается объектом пересечения, который сопоставляется с таблицей пересечения. Эта таблица пересечения будет содержать столбцы внешнего ключа, связанные с исходными таблицами.

При преобразовании связи 1:1 создается внешний ключ и уникальный ключ. Все столбцы этого внешнего ключа также входят в состав уникального ключа. Если связь обязательная с одной стороны, в соответствующей таблице создается внешний ключ.

Объект, имеющий дугу, будет сопоставлен с таблицей, содержащей внешние ключи из таблиц на конце связей «один». Даже хотя связи в дуге обязательные на стороне «многие», итоговые внешние ключи должны быть необязательными (так как один из них всегда будет пустым). Это может обеспечить ограничение проверки значений, хранящихся в базе данных.

Объекты-супертипы и объекты-подтипы можно сопоставлять несколькими способами:

– Реализация в одной таблице: создается одна таблица, независимо от количества подтипов. Используется, когда большинство атрибутов и связей общие и поэтому находятся на уровне супертипа.

Одна таблица содержит по одному столбцу для каждого атрибута супертипа, а также исходный необязательный характер атрибута. Также таблица содержит столбец для каждого атрибута, принадлежащего подтипу, но все столбцы становятся необязательными. Кроме того, необходимо создать обязательный столбец, который будет выступать столбцом-дискриминатором для различения разных типов подтипов объекта.

– Реализация в двух таблицах: для каждого подтипа создается по таблице (поэтому может быть больше двух таблиц). Используется, когда у подтипов мало общего, мало общих атрибутов и связей.

Одна таблица предназначена для одного подтипа первого уровня. Каждая таблица содержит по одному столбцу для каждого атрибута супертипа, а также его исходный необязательный характер. Каждая таблица также содержит по одному столбцу для каждого атрибута подтипа и его исходный необязательный характер.

**Тема 5 — Сопоставление с физической моделью**

**5-1 Сопоставление объектов и атрибутов**

o Объяснение правил именования, используемых в реляционной базе данных

*Необходимость создания физической модели:*

• Является прототипом фактической реализации базы данных.

• Может использоваться в качестве основы для реализации любых типов DBMS.

• Идеальную модель можно адаптировать для модели RDBMS.

В Oracle SQL Developer Data Modeler физическая модель называется *реляционной моделью*.

*Правила именования:*

– В качестве имени таблицы, как правило, используется имя соответствующего объекта во множественном числе.

– Имена столбцов часто совпадают с именами атрибутов, но есть некоторые исключения. Заменяем специальные символы (%, \*, #, -, пробел, …) нижним подчеркиванием. Имена столбцов часто сокращаются.

– Уникальное сокращение для каждой таблицы очень удобно использовать при именовании столбцов внешнего ключа или ограничений внешнего ключа.

*Ограничения именования в базе данных Oracle:*

Имена таблиц и столбцов:

– Должны начинаться с буквы

– Могут содержать до 30 буквенно-числовых символов

– Не должны содержать пробелы и некоторые специальные символы

– Не должны содержать зарезервированные слова

Имена таблиц в схеме и имена столбцов в таблице должны быть уникальными.

▪ Сопоставление объектов и имен таблиц

При применении стандартов именования из глоссария имена объектов логической модели в единственном числе сопоставляются с именами таблиц реляционной модели во множественном числе.

▪ Сопоставление атрибутов и имен столбцов

Если имена атрибутов объекта включают в себя название объекта, то после применения стандартов именования из глоссария к именам столбцов применяются сокращения, заданные для данного объекта.

**5-2 Сопоставление первичных и внешних ключей**

o Выбор правил именования для следующих элементов:

Ограничения обеспечивают применение правил, согласованность и целостность базы данных. Ограничениям следует давать понятные имена, чтобы на них было проще ссылаться. По умолчанию имена ограничений создаются из полного имени таблицы в Oracle SQL Developer Data Modeler. По этой причине имена ограничений могут быть очень длинными, сложными и могут превышать максимально допустимое число символов для SQL.

▪ Имена ограничений первичного ключа

Имя ограничения первичного ключа состоит из краткого имени таблицы, нижнего подчеркивания и суффикса PK.

▪ Имена ограничений внешнего ключа

Имя ограничения внешнего ключа состоит из кратких имен обоих таблиц, нижнего подчеркивания и суффикса FK.

▪ Имена столбцов внешнего ключа

Имя столбца внешнего ключа состоит из краткого имени таблицы и имени столбца в таблице, на который ссылается этот внешний ключ.

*Создание сокращенных имен:*

Чтобы применить стандарты именования к именам ограничений, создайте файл .csv в приложении для работы с электронными таблицами. • В первом столбце укажите полные имена таблиц, а во втором столбце — используемые сокращения. Сохраните файл с расширением .csv и запомните его расположение.

Чтобы применить сокращения к модели, необходимо задать образец в шаблоне именования. Для этого необходимо нажать правой кнопкой мыши на проект в обозревателе объектов, выбрать Properties, развернуть Settings и Naming Standard и выбрать Templates. Там можно отредактировать шаблон, чтобы использовать сокращенные имена для первичных и внешних ключей и внешнего ключа столбца. Далее нужно добавить в шаблон элемент abbr.

Чтобы применить созданные сокращенные имена, нужно выбрать Tools, далее выбрать Name Abbreviations, перейти к файлу .csv, в котором хранятся сокращения, и снять флажок Tables, чтобы сохранить существующие имена из глоссария. Теперь в реляционной модели есть имена ограничений, созданные в соответствии со стандартами именования.

**Тема 6 — Основы языка SQL**

**6-2 Структурированный язык запросов (SQL)**

o Описание структуры данных в реляционной базе данных

Для хранения данных используется двухмерная матрица, известная как таблица. Для управления чтением данных и манипуляциями с данными используется программное обеспечение DBMS.

o Описание различных терминологий реляционных баз данных

Терминология реляционных баз данных:

*Строка* – определяет экземпляр объекта, хранит информацию;

*Уникальный идентификатор* – первичный ключ, однозначно определяет запись

*Поля таблицы* - хранят информацию

*Внешний ключ* – поле ссылается на другую таблицу

*Ячейка* – на пересечении строки и столбца, содержит одно значение, может быть пустым (null)

o Определение структурированного языка запросов и его функций

В реляционной базе данных не указывается маршрут для доступа к таблицам, и вам не нужно знать физическую структуру данных. Чтобы получить доступ к базе данных, необходимо выполнить инструкцию SQL. *SQL* — это язык стандарта American National Standards Institute (ANSI), используемый для работы с реляционными базами данных. Структурированный язык запросов (SQL) — это декларативный язык на основе наборов, используемый для доступа к базе данных Oracle. SQL предоставляет интерфейс для реляционной базы данных и инструкции для работы с базой данных. У нее есть свои преимущества – это эффективность, простота в освоении и использовании, функциональность.

*Функции SQL:*

* Создание, замена, изменение и удаление объектов базы данных.
* Вставка, обновление и удаление строк в таблице.
* Запрос данных, хранящихся в базе данных.
* Контроль доступа к базе данных и объектам базы данных.
* Обеспечение согласованности и целостности базы данных.

*Типы команд SQL:*

• DDL (язык описания данных) — определяет структуры баз данных

• DML (язык манипулирования данными) — позволяет манипулировать данными (INSERT, UPDATE, DELETE)

• DQL (язык запросов данных) — позволяет выбирать данные (SELECT)

• DCL (язык управления данными) — позволяет контролировать доступ пользователей

• TCL (язык управления транзакциями) — позволяет управлять транзакциями базы данных

o Описание процесса обработки SQL

*Этапы обработки SQL:*

1. Разбор
2. Оптимизация
3. Создание источника строк
4. Выполнение

o Определение средств, используемых для доступа к реляционной базе данных

Пользователи могут получать доступ к данным на сервере баз данных Oracle с помощью специализированных программ, установленных на компьютере. Эти программы называются *клиентами* и используются для отправки инструкций SQL (команд) на сервер.

*Программы*: Oracle Application Express, SQL\*Plus, Oracle SQL Developer

**6-3 Язык описания данных (DDL)**

o Определение действий для создания таблиц базы данных

Таблица - (Представляет собой базовую единицу хранения; состоит из строк);

Представление - (Логически представляет подмножества данных из одной или нескольких таблиц);

Последовательность - (Создает числовые значения);

Индекс - (Повышает эффективность некоторых запросов);

Синоним - (Присваивает альтернативное имя объекту).

*Имена таблиц и столбцов должны:*

• Начинаться с буквы

• Содержать от 1 до 30 символов

• Содержать только символы A–Z, a–z, 0–9, \_, $ и #

• Не дублировать имена других объектов, созданных тем же пользователем

• Не являться словами, зарезервированными для сервера Oracle

Чтобы создать таблицу базы данных, нужно использовать команду CREATE TABLE. Ее синтаксис:

CREATE TABLE [schema.]table

(column datatype [DEFAULT expr][, ...]);

Чтобы проверить, создана ли таблица, выполните команду DESCRIBE.

*Пример создания таблицы:*

CREATE TABLE dept

(deptno NUMBER(2),

dname VARCHAR2(14),

loc VARCHAR2(13),

create\_date DATE DEFAULT SYSDATE);

*Пример подтверждения создания таблицы:*

DESCRIBE dept;

Параметр DEFAULT указывает значение по умолчанию для столбца во время выполнения инструкции CREATE TABLE. Этот параметр предотвращает добавление значений null в столбцы, когда при вставке строки не указано значение столбца. Имя другого столбца или псевдостолбца — недопустимые значения. Тип данных по умолчанию должен совпадать с типом данных столбца.

*Пример:*

CREATE TABLE hire\_dates

(id NUMBER(8),

hire\_date DATE DEFAULT SYSDATE);

*Добавление ограничений:*

Ограничения применяют правила на уровне таблицы. Ограничения обеспечивают согласованность и целостность базы данных. Допускаются следующие типы ограничений:

– NOT NULL (Столбец не может содержать значение null)

– UNIQUE (Значения столбца или комбинации столбцов должны быть уникальными для всех строк в таблице.)

– PRIMARY KEY (Столбец (или комбинация столбцов) должны содержать уникальное значение AND IS NOT NULL для всех строк.)

– FOREIGN KEY (Столбец (или комбинация столбцов) должны создавать и применять ссылки на столбец или комбинацию столбцов в другой (или той же) таблице)

– CHECK (Условие должно иметь значение true.)

Ограничению можно присвоить имя. Рекомендуется давать ограничениям понятные имена, чтобы на них было проще ссылаться. Определять ограничение нужно на уровне столбца или таблицы. Ограничения на уровне столбца включаются при создании столбца. Ограничения на уровне таблицы определяются на последнем этапе создания таблицы. Эти ограничения должны ссылаться на столбец (или столбцы), которым они принадлежат.

o Описание цели языка описания данных (DDL)

Создание таблиц относится к языку описания данных SQL.

Язык определения данных (Data Definition Language) предназначен для создания, изменения и удаления объектов базы данных. *Инструкции DDL:*

– ALTER: изменение структуры объекта

– DROP: удаление объекта из базы данных

– RENAME: переименование объекта базы данных

o Перечисление операций DDL для создания и обслуживания таблиц базы данных

Инструкция ALTER TABLE позволяет изменить структуру таблицы:

• Добавить столбец

• Изменить определение существующего столбца

• Указать значение по умолчанию для нового столбца

• Удалить столбец

• Переименовать столбец

• Присвоить таблице статус «только чтение»

С помощью инструкции ALTER TABLE можно добавить, изменить или удалить столбцы:

ALTER TABLE table

ADD (column data type [DEFAULT expr]

[, column data type]...);

ALTER TABLE table

MODIFY (column data type [DEFAULT expr]

[, column data type]...);

ALTER TABLE table

DROP (column [, column] …);

Для добавления столбцов используется фраза ADD:

ALTER TABLE dept

ADD job\_id VARCHAR2(9);

Можно изменить тип данных, размер данных и значение по умолчанию для столбца:

ALTER TABLE dept

MODIFY dname VARCHAR2(30);

Измененное значение по умолчанию применяется только при последующих вставках в таблицу.

С помощью фразы DROP COLUMN можно удалить ненужные столбцы:

ALTER TABLE dept

DROP (job\_id);

Параметр SET UNUSED позволяет пометить столбец (или столбцы) как неиспользуемые. Эти столбцы можно удалить, когда уменьшится потребность в ресурсах системы. Параметр SET UNUSED позволяет пометить один или несколько столбцов как неиспользуемые. Параметр DROP UNUSED COLUMNS позволяет удалить столбцы, помеченные как неиспользуемые.

ALTER TABLE <table\_name>

SET UNUSED(<column\_name> [ , <column\_name>]);

С помощью синтаксиса ALTER TABLE можно:

• Перевести таблицу в режим «только чтение». Этот режим не позволяет вносить изменения DDL или DML во время обслуживания таблицы.

• Перевести таблицу обратно в режим «чтение/запись».

ALTER TABLE dept READ ONLY;

ALTER TABLE dept READ WRITE;

Инструкция DROP перемещает таблицу в корзину, удаляет таблицу вместе с данными, если указана фраза PURGE и делает недействительными зависимые объекты и аннулирует полномочия объектов для таблицы.

DROP TABLE dept;

**6-4 Язык манипулирования данными (DML)**

o Описание цели языка манипулирования данными (DDL)

Инструкция DML выполняется в следующих случаях:

– Добавление новых строк в таблицу (INSERT)

– Изменение существующих строк в таблице (UPDATE)

– Удаление существующих строк из таблицы (DELETE)

o Объяснение операций DML для управления данными в таблице базы данных:

▪ Вставка

Добавьте строки в таблицу с помощью инструкции INSERT. Этот синтаксис позволяет вставлять строки только по одной.

INSERT INTO table [(column [, column...])]

VALUES (value [, value...]);

Прежде чем выполнять любые операции DML с таблицей, создайте копию таблицы (копируется только ограничение NOT NULL):

CREATE TABLE copy\_departments

AS (SELECT \* FROM departments);

Если вы вставляете строку, содержащую значения для каждого столбца, во фразе INSERT не нужно указывать список столбцов. Нужно перечислять значения в том же порядке, в каком по умолчанию располагаются столбцы таблицы. Для каждого столбца необходимо указать значение.

INSERT INTO copy\_departments

VALUES (40, 'Advertising', 201, 1800);

Также можно указать список столбцов во фразе INSERT. А символы и значения дат заключаются в одинарные кавычки.

Вставка строк со значениями Null:

• Явный метод: пропустить один столбец в списке столбцов.

INSERT INTO copy\_departments(department\_id,department\_name)

VALUES(30,'Purchasing');

• Неявный метод: указать ключевое слово NULL во фразе VALUES.

INSERT INTO copy\_departments

VALUES (100, 'Finance', NULL, NULL);

Для ввода специальных значений в таблице можно использовать функции. Функция SYSDATE регистрирует текущую дату и время.

INSERT INTO copy\_employees (hire\_date)

VALUES (SYSDATE);

Вставка определенных значений даты и времени

INSERT INTO copy\_employees

VALUES (TO\_DATE('Dec 7, 2002', 'MON DD, YYYY'));

▪ Обновление

Изменить существующие значения в таблице можно с помощью инструкции UPDATE:

UPDATE table

SET column = value [, column = value, ...]

[WHERE condition];

Как правило, столбец первичного ключа во фразе WHERE используется как идентификатор одной обновляемой строки.

Если указана фраза WHERE, изменяются значения в определенной строке или строках:

UPDATE copy\_employees

SET department\_id = 50

WHERE employee\_id = 113;

Если фраза WHERE пропущена, изменяются значения во всех строках таблицы:

UPDATE copy\_employees

SET department\_id = 110;

Укажите SET column\_name= NULL, чтобы изменить значение столбца на NULL.

▪ Удаление

Удалить существующие строки из таблицы можно с помощью инструкции DELETE:

DELETE [FROM] table

[WHERE condition];

Если указана фраза WHERE, удаляются определенные строки:

DELETE FROM copy\_departments

WHERE department\_name = 'Purchasing';

Если фраза WHERE пропущена, удаляются все строки таблицы:

DELETE FROM copy\_departments;

Нельзя удалить строку, содержащую первичный ключ, который используется в качестве внешнего ключа в другой таблице.

Инструкция TRUNCATE удаляет все строки из таблицы, оставляя таблицу пустой, а ее структуру без изменений. Эта инструкция относится к языку DDL, а не DML, и ее непросто отменить.

Синтаксис: TRUNCATE TABLE table\_name;

**6-5 Язык управления транзакциями (TCL)**

o Описание цели языка управления транзакциями (TCL)

Транзакции состоят из инструкций DML, представляющих собой одно согласованное изменение данных. Сервер Oracle обеспечивает согласованность данных на основе транзакций. Транзакции обеспечивают дополнительную гибкость и возможности управления при изменении данных, а также гарантируют согласованность данных в случае сбоя обработки пользователем или системного сбоя.

Транзакция начинается при выполнении первой SQL-инструкции DML. Она завершается одним из следующих событий:

– Выполнение инструкции COMMIT или ROLLBACK.

– Выполнение инструкции DDL или TCL (автоматическая фиксация).

– Пользователь выполняет выход из используемого ПО SQL.

– Сбой системы.

o Описание операций TCL для управления следующими транзакциями:

▪ COMMIT

COMMIT завершает текущую транзакцию, делая все отложенные изменения данных постоянными.

COMMIT;

Состояние данных после COMMIT:

• Изменения данных сохранены в базе данных.

• Предыдущее состояние данных перезаписано.

• Все сеансы могут просматривать результаты.

• Блокировка затронутых строк снята; доступна обработка данных строк другими сеансами.

• Все точки сохранения удалены.

▪ SAVEPOINT

SAVEPOINT помечает точку сохранения в текущей транзакции.

UPDATE…

SAVEPOINT update\_done;

▪ ROLLBACK

ROLLBACK завершает текущую транзакцию, отменяя все отложенные изменения данных.

Состояние данных после ROLLBACK:

• Отмените все отложенные изменения с помощью инструкции ROLLBACK.

• Изменения данных отменены.

• Предыдущее состояние данных восстановлено.

• Блокировка затронутых строк снята.

DELETE FROM copy\_employees;

ROLLBACK ;

ROLLBACK TO SAVEPOINT name – выполняет откат текущей транзакции до указанной точки сохранения, тем самым отменяя все изменения и/или точки сохранения, созданные после точки сохранения, с которой выполняется откат.

INSERT…

ROLLBACK TO update\_done;

Если во время выполнения происходит сбой отдельной инструкции DML, выполняется откат только этой инструкции. Сервер Oracle реализует неявную точку сохранения. Все остальные изменения сохраняются. Пользователь должен явным образом прервать транзакции, выполнив инструкцию COMMIT или ROLLBACK.

o Описание необходимости согласованности чтения

Согласованность чтения гарантирует согласованный просмотр данных в любое время — каждый пользователь видит данные в том виде, в котором они существовали на момент последнего выполнения команды COMMIT. Изменения, выполненные одним пользователем, не конфликтуют с изменениями, выполненными другим пользователем.

**6-6 Извлечение данных с помощью SELECT**

o Список возможностей инструкций SQL SELECT

SELECT определяет столбцы для отображения. FROM определяет таблицу, содержащую данные столбцы.

SELECT {\*|[DISTINCT] column|expression [alias],...}

FROM table;

o Создание и выполнение инструкции SELECT, которая:

▪ Возвращает все строки и столбцы таблицы

Можно отобразить все столбцы таблицы с помощью ввода \* после SELECT

SELECT \*

FROM departments;

▪ Возвращает определенные столбцы таблицы

Можно использовать инструкцию SELECT для отображения определенных столбцов таблицы, указав имена столбцов в необходимом порядке отображения и разделив их запятой.

SELECT department\_id, location\_id

FROM departments;

▪ Использует псевдонимы столбцов для отображения описательных заголовков столбцов

*Псевдоним столбца:*

• переименовывает заголовок столбца;

• полезен при вычислениях;

• следует за именем столбца (между именем столбца и псевдонимом может использоваться дополнительное ключевое слово AS);

• требует использования двойных кавычек, если содержит пробелы или специальные символы или чувствителен к регистру; по умолчанию указывается в верхнем регистре.

SELECT last\_name AS name, commission\_pct comm

FROM employees;

Ключевое слово AS является необязательным.

▪ Использует арифметические операторы и операторы соединения

Создать выражения с числовыми данными и данными дат с помощью арифметических операторов. В арифметическом выражении можно использовать имена столбцов, числовые константы и арифметические операторы. Арифметические операторы можно использовать в любой фразе инструкции SQL, кроме FROM.

SELECT last\_name, salary, salary + 300 FROM employees;

Здесь оператор сложения используется для вычисления повышения заработной платы на $300 для всех сотрудников. В качестве заголовка столбца отображается SALARY + 300.

Определение значения null:

*Null* — это значение, которое недоступно, не назначено, неизвестно или неприменимо. Null не равно нулю и не является пробелом. Любое арифметическое выражение, содержащее значение null, будет принимать значение null.

Оператор объединения:

• Связывает столбцы или строки символов с другими столбцами

• Представлен двумя вертикальными полосами (||)

• Создает столбец, являющийся символьным выражением

• Объединение NULL с символом приводит к созданию строки символов

SELECT last\_name||job\_id

AS "Employees"

FROM employees;

▪ Использует строки литеральных символов

*Литерал* — это символ, число или дата, заключенные в инструкцию SELECT. Литеральные значения даты и символа должны быть заключены в одинарные кавычки. Вывод каждой строки выполняется один раз для каждой возвращаемой строки.

SELECT last\_name ||' is a '||job\_id

AS "Employee Details"

FROM employees;

В этом примере фамилия и job\_id каждого сотрудника объединены с литералом, чтобы придать возвращаемым строкам дополнительное значение.

Альтернативный оператор кавычки (q):

SELECT department\_name || q'[ Department's Manager Id: ]' || manager\_id

AS "Department and Manager"

FROM departments;

Многие инструкции SQL используют в выражениях или условиях символьные литералы. Если литерал включает в себя одинарную кавычку, можно использовать оператор кавычки (q) и выбрать собственный разделитель кавычки — в данном случае скобки [].

▪ Устраняет дублирование строк

По умолчанию в запросах отображаются все строки, включая повторяющиеся. Чтобы удалить из результатов повторяющиеся строки, включите ключевое слово DISTINCT во фразу SELECT сразу после ключевого слова SELECT.

SELECT DISTINCT department\_id

FROM employees;

o Описание структуры таблицы

Чтобы отобразить структуру таблицы, включая имя столбца, тип данных и допустимость неопределенного значения, используйте команду DESCRIBE. Или выберите Object Browser APEX в SQL Workshop для просмотра структуры таблицы. Для просмотра структур таблиц можно также нажать кнопку «Find Tables» в SQL Commands APEX.

DESC[RIBE] tablename

DESCRIBE employees

**6-7 Ограничение данных с помощью WHERE**

o Ограничение строк с помощью:

* Фразы WHERE

Ограничение возвращенных строк с помощью фразы WHERE:

SELECT \*|{[DISTINCT] column|expression [alias],...}

FROM table

[WHERE logical expression(s)];

Если логическое выражение выдает значение true, возвращается строка, отвечающая условию. Фраза WHERE следует за фразой FROM.

*Пример:*

SELECT employee\_id,last\_name,job\_id, department\_id

FROM employees

WHERE department\_id = 90;

* операторов сравнения, использующих условия =, <=, >=, <>, >, <, !=, ^=, BETWEEN, IN, LIKE и NULL

Строки символов и значения дат заключены в одинарные кавычки. Значения символов чувствительны к регистру; значения даты чувствительны к формату.

SELECT last\_name, job\_id, department\_id

FROM employees

WHERE last\_name = 'Whalen' ;

Форматом отображения даты по умолчанию является DD-Mon-YYYY.

SELECT last\_name

FROM employees

WHERE hire\_date = '29-Jan-2000' ;

Операторы сравнения:

= (Равно);

> (Больше);

>= (Больше или равно);

< (Меньше);

<= (Меньше или равно);

<> (Нe равно);

BETWEEN…AND… (Между двумя значениями (включительно));

IN (set) (Соответствует любому значению в списке);

LIKE (Соответствует шаблону символа);

IS NULL (Является значением null)

Извлеките записи из таблицы EMPLOYEES, где заработная плата меньше или равна $3,000:

SELECT last\_name, salary

FROM employees

WHERE salary <= 3000 ;

Используйте оператор BETWEEN для отображения строк на основе диапазона значений:

SELECT last\_name, salary

FROM employees

WHERE salary BETWEEN 2500 AND 3500 ;

Используйте оператор IN для тестирования значений в списке:

SELECT employee\_id, last\_name, salary, manager\_id

FROM employees

WHERE manager\_id IN (100, 101, 201) ;

Используйте оператор NOT IN для тестирования значений в списке:

SELECT employee\_id, last\_name, salary, manager\_id

FROM employees

WHERE department\_id NOT IN (60, 90, 100) ;

Используйте оператор LIKE для поиска с подстановочными знаками допустимых значений строки поиска. Условия поиска могут содержать символы или числа литерала:

– % обозначает ноль или более символов.

– \_ обозначает один символ.

SELECT first\_name

FROM employees

WHERE first\_name LIKE 'S%' ;

Можно объединить два подстановочных знака (%, \_) с символами литерала для соответствия шаблону:

SELECT last\_name

FROM employees

WHERE last\_name LIKE '\_o%' ;

Проверьте наличие значений null с помощью операторов IS NULL или IS NOT NULL:

SELECT last\_name, manager\_id

FROM employees

WHERE manager\_id IS NULL ;

Проверка с помощью = недоступна, так как значение null не может быть равно или не равно любому значению.

* логических условий с операторами AND, OR и NOT

AND (Возвращает значение TRUE, если оба компонентных условия имеют значение TRUE);

OR (Возвращает значение TRUE, если любое из компонентных условий имеют значение TRUE);

NOT (Возвращает значение TRUE, если условие имеет значение FALSE; Возвращает значение FALSE, если условие имеет значение TRUE)

AND требует, чтобы выполнялись оба компонентных условия:

SELECT employee\_id, last\_name, job\_id, salary

FROM employees

WHERE salary >= 10000

AND job\_id LIKE '%MAN%' ;

OR требует, чтобы одно из компонентных условий имело значение true:

SELECT employee\_id, last\_name, job\_id, salary

FROM employees

WHERE salary >= 10000

OR job\_id LIKE '%MAN%' ;

NOT изменяет значение условия на противоположное

SELECT last\_name, job\_id

FROM employees

WHERE job\_id

NOT IN ('IT\_PROG', 'ST\_CLERK', 'SA\_REP');

o Описание правил приоритета для операторов в выражении

Правила приоритета:

1 Арифметические операторы

2 Оператор объединения

3 Условия сравнения

4 IS [NOT] NULL, LIKE, [NOT] IN

5 [NOT] BETWEEN

6 Не равно

7 Логический оператор NOT

8 Логический оператор AND

9 Логический оператор OR

Чтобы переопределить правила приоритета, необходимо использовать круглые скобки.

**6-8 Сортировка данных с помощью ORDER BY**

o Применение фразы ORDER BY для сортировки результатов SQL-запроса

Сортировка извлеченных строк с помощью фразы ORDER BY :

– ASC: по возрастанию (по умолчанию)

– DESC: по убыванию

Фраза ORDER BY расположена в инструкции SELECT последней:

SELECT last\_name, job\_id, department\_id, hire\_date

FROM employees

ORDER BY hire\_date ;

o Определение правильного положения для фразы ORDER BY в инструкции SELECT

Числовые значения отображаются в порядке убывания или возрастания. Отображение значений дат начинается с наиболее ранней даты. Значения символов отображаются в алфавитном порядке. Значения null отображаются последними в порядке возрастания и первыми в порядке убывания. NULLS FIRST указывает, что значения NULL должны быть возвращены перед значениями, не являющимися значениями NULL. NULLS LAST указывает, что значения NULL должны быть возвращены после значений, не являющихся значениями NULL.

*Сортировка в порядке убывания:*

SELECT last\_name, job\_id, department\_id, hire\_date

FROM employees

ORDER BY hire\_date DESC ;

*Сортировка по псевдонимам столбцов:*

SELECT employee\_id, last\_name, salary\*12 annsal

FROM employees

ORDER BY annsal;

*Сортировка с помощью номера позиции столбца:*

SELECT last\_name, job\_id, department\_id, hire\_date

FROM employees

ORDER BY 3;

*Сортировка по нескольким столбцам:*

SELECT last\_name, department\_id, salary

FROM employees

ORDER BY department\_id, salary DESC;

Предусмотрен следующий порядок выполнения инструкции SELECT:

– Фраза FROM: находит таблицу с данными

– Фраза WHERE: ограничивает возвращаемые строки

– Фраза SELECT: выбирает запрашиваемые столбцы в уменьшенном наборе данных

– Фраза ORDER BY: упорядочивает набор результатов

o Применение ROWNUM для Top-N-Analysis

Top-n-analysis представляет собой операцию SQL, выполняемую для ранжирования результатов. В APEX запросы Top-N выполняются с помощью ROWNUM.

SELECT ROWNUM As "Longest employed",last\_name,hire\_date

FROM (SELECT last\_name, hire\_date

FROM employees

ORDER BY hire\_date)

WHERE ROWNUM <=3;

o Применение подстановочных переменных во фразе WHERE

При создании отчета пользователям часто требуется динамическое ограничение возвращаемых данных. Подстановочные переменные позволяют создать отчеты с запросами на указание пользователями собственных значений, чтобы ограничить диапазон возвращаемых данных. Можно встроить подстановочные переменные в файл команд или в отдельную инструкцию SQL. Переменная может быть представлена как контейнер, в котором временно хранятся значения. При выполнении инструкции подставляется сохраненное значение. Используют подстановочные переменные для запроса значений. Нужно использовать переменную с префиксом в виде двоеточия (:) для запроса пользовательского значения:

SELECT employee\_id, last\_name, salary, department\_id

FROM employees

WHERE employee\_id = :employee\_num ;

Когда APEX определяет наличие постановочной переменной в инструкции SQL, отобразится запрос на указание ее значения. После ввода значения и нажатия кнопки Submit результаты отобразятся на вкладке Results сеанса APEX.

Значения символов и дат с подстановочными переменными

Можно также использовать значения даты или символов:

SELECT last\_name, department\_id, salary\*12

FROM employees

WHERE job\_id = :job\_title ;

**6-9 Соединение таблиц с помощью JOIN**

o Создание инструкций SELECT для доступа к данным из более чем одной таблицы, используя соединения по эквивалентности и неэквивалентности

Иногда требуется использовать данные из различных таблиц.

*Типы соединений:*

– Естественное соединение с помощью фразы NATURAL JOIN

– Соединение с помощью фразы USING

– Соединение с помощью фразы ON

– Соединения OUTER:

• LEFT OUTER JOIN

• RIGHT OUTER JOIN

• FULL OUTER JOIN

– CROSS JOIN

Соединение для запроса данных из нескольких таблиц:

SELECT table1.column, table2.column

FROM table1

[NATURAL JOIN table2] |

[JOIN table2 USING (column\_name)] |

[JOIN table2

ON (table1.column\_name = table2.column\_name)]|

[LEFT|RIGHT|FULL OUTER JOIN table2

ON (table1.column\_name = table2.column\_name)]|

[CROSS JOIN table2];

Используйте псевдонимы таблиц для различия столбцов с одинаковыми именами, содержащихся в различных таблицах:

SELECT e.first\_name,d.department\_name,d.manager\_id,

FROM employees e JOIN departments d

USING(deparment\_id)

Фраза NATURAL JOIN основана на всех столбцах в двух таблицах, имеющих одинаковое имя и тип данных. Она выбирает из двух таблиц строки с одинаковыми значениями во всех совпадающих столбцах.

SELECT department\_id, department\_name,location\_id, city

FROM departments NATURAL JOIN locations;

Если соединяемые таблицы совместно используют несколько столбцов, все общие поля используются в соединении. Используйте фразу USING для указания отдельного столбца для JOIN вместо NATURAL JOIN. Фразу USING также можно использовать для сопоставления столбцов с одинаковым именем, но разными типами данных. Фразы NATURAL JOIN и USING являются взаимоисключающими.

SELECT employee\_id, last\_name, location\_id, department\_id

FROM employees JOIN departments

USING (department\_id) ;

NATURAL JOIN создает соединение по эквивалентности для всех столбцов с одинаковым именем и типом данных. Используйте фразу ON для указания произвольных условий или указания столбцов для соединения. Условие соединения отделяется от других условий поиска. Фраза USING создает соединение по эквивалентности между двумя таблицами с помощью одного столбца с одинаковым именем, независимо от типа данных. Фраза ON создает соединение по эквивалентности между двумя таблицами с помощью одного столбца из каждой таблицы, независимо от имени или типа данных.

Можно также использовать фразу ON для соединения столбцов с различными именами или типами данных.

SELECT e.employee\_id, e.last\_name, e.department\_id, d.department\_id, d.location\_id

FROM employees e JOIN departments d

ON (e.department\_id = d.department\_id);

При соединении 3 таблиц необходимо наличие 2 инструкций соединения:

SELECT employee\_id, city, department\_name

FROM employees e

JOIN departments d

ON d.department\_id = e.department\_id

JOIN locations l

ON d.location\_id = l.location\_id;

В данном примере выполняется создание соединения не по эквивалентности для оценки категории оклада сотрудника. Уровень заработной платы должен находиться в пределах любой пары верхнего и нижнего значений диапазона заработной платы:

SELECT e.last\_name, e.salary, j.grade\_level

FROM employees e JOIN job\_grades j

ON e.salary

BETWEEN j.lowest\_sal AND j.highest\_sal;

o Присоединение таблицы к самой себе путем самосоединения

Фразу ON можно также использовать для соединения столбцов с различными именами, расположенных в одной или различных таблицах.

SELECT worker.last\_name emp, manager.last\_name mgr

FROM employees worker JOIN employees manager

ON (worker.manager\_id = manager.employee\_id);

o Применение внешних соединений (OUTER JOIN) для просмотра данных, которые обычно не соответствуют условию соединения

В SQL:1999 соединение двух таблиц, возвращающих только совпадающие строки, называется соединением INNER. Соединение между двумя таблицами, возвращающее результаты соединения INNER и несовпадающие строки из таблицы слева (или справа), называется соединением LEFT OUTER (или RIGHT OUTER). Соединение между двумя таблицами, возвращающее результаты соединения INNER и результаты соединения OUTER слева и справа, является полным соединением FULL OUTER.

Здесь нам необходимо просмотреть все записи сотрудников (таблица слева), даже если они не назначены подразделению:

SELECT e.last\_name, e.department\_id, d.department\_name

FROM employees e LEFT OUTER JOIN departments d

ON (e.department\_id = d.department\_id) ;

Здесь нам необходимо просмотреть все записи подразделений (таблица справа), даже если в них отсутствуют сотрудники:

SELECT e.last\_name, d.department\_id, d.department\_name

FROM employees e RIGHT OUTER JOIN departments d

ON (e.department\_id = d.department\_id) ;

Здесь нам необходимо просмотреть все записи сотрудников и все записи подразделений:

SELECT e.last\_name, d.department\_id, d.department\_name

FROM employees e FULL OUTER JOIN departments d

ON (e.department\_id = d.department\_id) ;

o Создание декартова произведения (перекрестного соединения) всех строк из двух и более таблиц

В декартовом произведении отображаются все комбинации строк. Все строки в первой таблице связаны со всеми строками во второй таблице.

SELECT last\_name, department\_name

FROM employees, departments

Декартово произведение образуется, когда условие соединения опущено или недопустимо. Чтобы предотвратить образования декартова произведения, всегда указывайте допустимое условие соединения.

SELECT last\_name, department\_name

FROM employees e, departments d

WHERE e.department\_id = d.department\_id

Фраза CROSS JOIN создает векторное произведение двух таблиц. Оно также называется декартовым произведением между двумя таблицами.

SELECT last\_name, department\_name

FROM employees

CROSS JOIN departments ;

**SELECT and WHERE**

**2-1 Columns, Characters, and Rows (столбцы, символы и строки)**

DESCRIBE

•Использование DESCRIBE (DESC) команды для отображения структуры таблицы

DESCRIBE <table name>;

•DESC возвращает имя таблицы, типы данных, первичный и внешний ключи и столбцы с возможностью обнуления, а также другие сведения об объекте. Это важно при вставке новых строк в таблицу, поскольку нужно знать, какой тип данных будет принимать каждый столбец и можно ли оставить столбец пустым.

Оператор конкатенации

• Конкатенация означает соединение. • Символом объединения являются 2 вертикальные полосы ||. •Значения по обе стороны от оператора || объединяются в один выходной столбец • Синтаксис: string1 || string2 || string\_n (||' '|| - создание пробела)

•При объединении значений результирующее значение представляет собой символьную строку. •В SQL оператор конкатенации может связывать столбцы с другими столбцами, арифметическими выражениями или постоянными значениями для создания символьного выражения. • Оператор используется для создания читаемого текстового вывода

Объединение и псевдонимы столбцов

• Псевдонимы столбцов полезны при использовании оператора конкатенации, чтобы строка ВЫБОРА по умолчанию не отображалась в качестве заголовка столбца (… AS “\_”)

Конкатенация и литеральные значения

•Буквальное значение — это фиксированное значение данных, такое как символ, число или дата. •Используя конкатенацию и литеральные значения, вы можете создать выходные данные, которые выглядят как предложение или утверждение

• Литеральные значения могут быть включены в список SELECTс помощью оператора конкатенации • Символы и даты должны быть заключены в одинарные кавычки

SELECT … ' || salary || ' dollars.' AS Pay FROM employees; ‘dollars’ – литерал.

• Вы также можете включать числа в качестве буквенных значений SELECT last\_name ||' has a '|| 1 || ' dollars.' AS Pay FROM employees;

Использование DISTINCT для устранения повторяющихся строк

•В SQL ключевое слово DISTINCT используется для устранения повторяющихся

Строк • Квалификатор DISTINCT влияет на все перечисленные столбцы и возвращает каждую отдельную комбинацию столбцов в предложении SELECT • Ключевое слово DISTINCT должно отображаться непосредственно после ключевого слова SELECT

SELECT DISTINCT department\_id FROM employees;

ВЫПОЛНЕНИЕ, СОХРАНЕНИЕ и РЕДАКТИРОВАНИЕ в приложении Oracle

**2-2 Limit Rows Selected (Ограничение выбранных строк)**

SELECT: Оператор SELECT должен включать, как минимум, предложение SELECT и предложение FROM • Предложение WHERE является необязательным

SELECT\*|{[DISTINCT] column | expression alias]..} FROM table [WHERE condition(s)];

WHERE •При извлечении данных из базы данных может потребоваться ограничить количество отображаемых строк данных • Вы можете сделать это с помощью предложения WHERE •Предложение WHERE содержит условие, которое должно быть выполнено, и оно непосредственно следует за предложением FROM в инструкции SQL

• Синтаксис предложения WHERE следующий: WHERE column\_name comparison\_condition comparison\_value

•Примечание: Псевдоним не может быть использован в предложении WHERE!

SELECT employee\_id, first\_name, last\_name FROM employees WHERE employee\_id = 101;

Операторы сравнения в предложении WHERE

•В дополнение к оператору "равно" (=) для сравнения одного выражения с другим можно использовать другие операторы сравнения: =, >, >=, <, <=, <>

SELECT employee\_id, last\_name, department\_id FROM employees WHERE department\_id = 90;

Строки символов и даты в предложении WHERE

• Символьные строки и даты в предложении WHERE должны быть заключены в одинарные кавычки ' ' •цифры не должны заключаться в ‘’.

SELECT first\_name, last\_name FROM employees WHERE last\_name = 'Taylor';

•Все поисковые запросы символов чувствительны к регистру

Операторы сравнения в предложении WHERE

• Операторы сравнения могут использоваться всеми следующими способами в предложении WHERE:

WHERE hire\_date < '01-Jan-2000' / WHERE salary >= 6000 / WHERE job\_id = 'IT\_PROG'

**2-3 Comparison Operators (Операторы сравнения)**

• В SQL есть другие операторы, которые добавляют функциональность для извлечения определенных наборов данных.

• К ним относятся: BETWEEN…AND −IN −LIKE

BETWEEN...AND

• BETWEEN...AND Оператор AND используется для выбора и отображения строк на основе диапазона значений •При использовании с предложением WHERE BETWEEN...AND условие вернет диапазон значений между указанными нижним и верхним пределами включительно •Значения, указанные с условием BETWEEN, считаются включающими •выходные данные включают значения нижнего и верхнего пределов

SELECT last\_name, salary FROM employees WHERE salary BETWEEN 9000 AND 11000;

• Использование BETWEEN…AND совпадает с использованием

следующего выражения: WHERE salary >= 9000 AND salary <=11000;

IN

•Используется для проверки того, входит ли значение В указанный набор значений

WHERE country\_id IN('UK', 'CA'); •В этом примере предложение WHERE также может быть записано как набор условий ИЛИ: WHERE country\_id = 'UK' OR country\_id = 'CA';

•Как и в случае с BETWEEN...AND условие IN может быть записано с использованием любого синтаксиса так же эффективно

LIKE

• LIKE позволяет выбирать строки, соответствующие символам, датам или шаблонам чисел • Два символа - (%) и символ подчеркивания (\_) – называемые подстановочными знаками, могут использоваться для построения строки поиска.

• Символ процента (%) используется для обозначения любого последовательность из нуля или более символов • Символ подчеркивания (\_) используется для обозначения одного символа. WHERE last\_name LIKE '\_o%';

• Опция ESCAPE может использоваться для указания того, что \_ или % - это часть имени, а не подстановочное значение SELECT last\_name, job\_id FROM EMPLOYEES WHERE job\_id LIKE '%\\_R%' ESCAPE '\'; Без опции ESCAPE все сотрудники , у которых есть R в их JOB\_ID будет возвращен

IS NULL, IS NOT NULL

•Часто желательно иметь возможность проверять значение NULL • Условие IS NULL проверяет наличие недоступных, неназначенных или неизвестных данных • IS NOT NULL возвращает строки, которые имеют значение

SELECT last\_name, manager\_id FROM employees WHERE manager\_id IS NULL;

**3-1 Logical Comparisons and Precedence Rules (Логические Сравнения и Правила приоритета)**

•В SQL часто желательно иметь возможность ограничить количество строк, возвращаемых запросом, на основе двух или более условий

Логические Условия: • Логические условия объединяют результат двух составляющих условий для получения единого результата на их основе AND. •Другое логическое условие объединяет два составных условия с OR

Логические операторы: •Результат возвращается ТОЛЬКО В ТОМ СЛУЧАЕ, ЕСЛИ общий результат условия равен true. • AND -- Возвращает значение TRUE, если оба условия верны • OR -- Возвращает значение TRUE, если выполняется одно из условий •NOT - Возвращает значение TRUE, если условие равно false

AND Operator

•В приведенном ниже запросе возвращаемыми результатами будут строки, удовлетворяющие ОБОИМ условиям, указанным в WHERE

SELECT last\_name, hire\_date, job\_id FROM employees WHERE hire\_date > '01-Jan-1998' AND job\_id LIKE 'SA%';

OR Operator

•Если в предложении WHERE используется условие OR, результаты , возвращаемые из запроса, будут строками, удовлетворяющими одному из условий OR

SELECT department\_name, manager\_id, location\_id FROM departments WHERE location\_id = 2500 OR manager\_id=124;

NOT Operator

• Оператор NOT вернет строки, которые не удовлетворяют условию в предложении WHERE

SELECT department\_name, location\_id FROM departments WHERE location\_id NOT IN (1700,1800);

Правила приоритета

SELECT last\_name||' '||salary\*1.05 As "Employee Raise" FROM employees WHERE department\_id IN(50,80) AND first\_name LIKE 'C%' OR last\_name LIKE '%s%';

•Добавление круглых скобок изменяет способ вычисления предложения Where и возвращаемых строк

**3-2 Sorting Rows (Строки Сортировки)**

ORDER BY

• SQL использует предложение ORDER BY для упорядочивания данных • Предложение ORDER BY может указывать несколько способов упорядочивания строк, возвращаемых в запросе

• Порядок сортировки по умолчанию - по возрастанию ASC •Числовые значения отображаются от самого низкого до самого высокого •Значения даты отображаются с самым ранним значением первым • Значения символов отображаются в алфавитном порядке •Нулевые значения отображаются последними в порядке возрастания и первыми в порядке убывания •NULLS FIRST указывает, что нулевые значения должны быть возвращены перед ненулевыми значениями \*NULLS LAST Указывает, что нулевые значения должны быть возвращены после ненулевых значений

Предложение ORDER BY должно быть последним предложением инструкции SQL. SELECT last\_name, hire\_date FROM employees ORDER BY hire\_date;

Сортировка в порядке убывания

• Вы можете изменить порядок по умолчанию в предложении ORDER BY на порядок убывания, указав DESC, ключевое слово после имени столбца в предложении ORDER BY

ORDER BY hire\_date DESC;

• Вы можете упорядочить данные, используя псевдоним столбца. • Псевдоним, используемый в инструкции SELECT, указан в предложении ORDER BY (SELECT last\_name, hire\_date AS "Date Started" FROM employees ORDER BY "Date Started";)

•Также можно использовать предложение ORDER BY для упорядочения выходных данных по столбцу, которого нет в списке SELECT (SELECT employee\_id, first\_name FROM employees WHERE employee\_id < 105 ORDER BY last\_name;)

• Порядок выполнения инструкции SELECT выглядит следующим образом:

−FROM: находит таблицу, содержащую данные

−WHERE: ограничивает возвращаемые строки

−SELECT: выбирает из сокращенного набора данных столбцы запрошенный

−ORDER BY: упорядочивает результирующий набор

Сортировка по нескольким столбцам

Нет никаких ограничений на количество столбцов, которые можно добавить в предложение ORDER BY (SELECT department\_id, last\_name FROM employees WHERE department\_id <= 50 ORDER BY department\_id, last\_name;)

• Чтобы создать предложение ORDER BY для сортировки по нескольким столбцам, укажите возвращаемые столбцы и разделите имена столбцов запятыми •Если вы хотите изменить порядок сортировки столбца на обратный, добавьте DESC после его названия

**3-3 Introduction to Functions (Введение в функции)**

• Функции - это небольшие программы, которые выполняют действие на значение или столбец и создать что-то другое в качестве выходных данных

Функции

• Функции имеют как ввод, так и вывод. Ввод в функцию называется аргументом

•Oracle имеет два различных типа функций:−Однострочный Single-Row−Многострочный Multiple-Row

Single-Row И Multiple-Row Functions

• Однострочные функции работают только с отдельными строками и возвращают один результат на строку • Существуют различные типы однострочных функций, включая функции символов, чисел, даты и преобразования

•Многострочные функции могут манипулировать группами строк для получения одного результата на группу строк. • Эти функции также известны как групповые функции

Single-Row Functions

• В SQL однострочные функции можно использовать для:− Выполнять вычисления, такие как округление чисел до указанного десятичного знака− Изменение отдельных элементов данных, таких как преобразование значений символов из верхнего регистра в нижний регистр• Форматирование дат и чисел для отображения например, преобразование внутреннего числового формата даты базы данных в стандартный формат• Преобразование типов данных столбцов, таких как преобразование символьной строки в число или дату.

• Однострочные функции принимают один или несколько аргументов и возвращают один результат для каждой строки •Таким образом, однорядные функции выполняют следующее: −Манипулировать элементами данных −Принимать аргументы и возвращать одно значение −Действуйте с каждой возвращенной строкой −Возвращает один результат для каждой строки−Можно изменить тип данных −Может быть вложенным

Multiple-Row Functions

•групповые функции принимают много строк в качестве входных данных и возвращают одно значение в качестве выходных • Вводимые строки могут представлять собой всю таблицу целиком или таблицу, разбитую на более мелкие группы

• Примеры многорядных (групповых) функций включают:−MAX: находит наибольшее значение в группе строк.−MIN: находит наименьшее значение в группе строк.− AVG: находит среднее значение в группе строк

**Single Row Functions**

**4-1 Case and Character Manipulation (Манипуляции с регистром и символами)**

DUAL Table

• DUAL Table содержит одну строку с именем "X" и один столбец с именем ""DUMMY“. • DUAL Table используется для создания инструкций SELECT и выполнения функций, не связанных напрямую с конкретной таблицей базы данных •Запросы, использующие ДВОЙНУЮ таблицу, возвращают в результате одну строку. DUAL может быть полезен для выполнения вычислений, а также для вычисления выражений, которые не являются производными от таблицы •DUAL будет использоваться для изучения многих однострочных функций. •В этом примере ДВОЙНАЯ таблица используется для выполнения SELECT, содержащую вычисление (SELECT (319/29) + 12 FROM DUAL;) • Возвращаемое значение является результатом выполненного вычисления

Однострочные символьные функции

• Функции однорядных символов делятся на две категории: −Функции, преобразующие регистр символьных строк −Функции, которые могут объединять, извлекать, показывать, находить, дополнять и обрезать символьные строки. • Однострочные функции можно использовать в предложениях SELECT, WHERE и ORDER BY. • Функции управления регистром важны, поскольку вы не всегда можете знать, в каком регистре (верхнем, нижнем или смешанном) данные хранятся в базе данных. • Управление регистром позволяет временно преобразовать данные базы данных в регистр по вашему выбору •Устраняются несоответствия между хранилищем обращений к базе данных и запросами обращений к запросам

Функции управления регистром используются для преобразования данных из состояния, в котором они хранятся в таблице, в нижний, верхний или смешанный регистр• Эти преобразования можно использовать для форматирования выходных данных, а также для поиска определенных строк

Case-Manipulation Functions: LOWER UPPER INITCAP /// Character-Manipulation Functions: CONCAT SUBSTR LENGTH INSTR LPAD | RPAD TRIM REPLACE

Функции обработки обращений

• Функции управления регистром часто полезны, когда вы ищете данные и не знаете , написаны ли данные, которые вы ищете, в верхнем или нижнем регистре • С точки зрения базы данных, 'V' и 'v' являются НЕ тот же символ, и, как таковой, вам нужно искать, используя правильный регистр

• LOWER(column | expression) преобразует буквенные символы в нижний регистр (SELECT last\_name FROM employees WHERE LOWER(last\_name) = 'abel';)

• UPPER(column | expression) преобразует буквенные символы в верхний регистр (SELECT last\_name FROM empl oyees WHERE UPPER(last\_name) = 'ABEL';)

• •INITCAP(column | expression) преобразует значения буквенных символов в верхний регистр для первой буквы каждого слова (SELECT last\_name FROM employees WHERE INITCAP(last\_name) = 'Abel';)

Функции манипулирования символами

• Функции обработки символов используются для извлечения, изменения, форматирования или изменения каким-либо образом символьной строки •Один или несколько символов или слов передаются в функцию, и затем функция выполняет свою функциональность с входными символьными строками и возвращает измененное, извлеченное, подсчитанное или измененное значение.

• CONCAT: Объединяет два значения вместе • Принимает 2 символьных строковых аргумента и присоединяет вторую строку к первой. также может быть записан с использованием оператора конкатенации - 'Hello' || 'World' (SELECT CONCAT('Hello', 'World') FROM DUAL;)

• SUBSTR: Извлекает строку определенной длины. • Аргументы следующие (символьная строка, начальная позиция,длина)• Аргумент Length является необязательным, и если он опущен,возвращает все символы до конца строки.

SELECT SUBSTR('HelloWorld', 1, 5) FROM DUAL; Hello

• LENGTH: показывает длину строки в виде числового значения. • Функция принимает символьную строку в качестве аргумента и возвращает количество символов в этой символьной строке SELECT LENGTH('HelloWorld') FROM DUAL; 10

•INSTR: Находит числовую позицию указанного символа (ов). •INSTR выполняет поиск первого вхождения подстроки в символьной строке и возвращает позицию в виде числа •Если подстрока не найдена, возвращается нулевое число. SELECT INSTR('HelloWorld', 'W') FROM DUAL; 6

• LPAD: заполняет левую часть символьной строки, в результате чего значение выравнивается по правому краю. • Для LPAD требуется 3 аргумента: символьная строка, общее количество символов в заполняемой строке и символ, который нужно заполнить SELECT LPAD('HelloWorld', 15, '-') FROM DUAL; -----HelloWorld

•RPAD: заполняет правую часть символьной строки, в результате чего значение выравнивается по левому краю. SELECT RPAD('HelloWorld', 13, '-') FROM DUAL; HelloWorld---

• TRIM: удаляет все указанные символы либо из начала, либо из конца, либо из начала и конца строки одновременно. SELECT TRIM(LEADING 'a' FROM 'abcba') FROM DUAL; bcba

•REPLACE: заменяет последовательность символов в строке другим набором символов. функции REPLACE следующий: REPLACE (string1, string\_to\_replace, [replacement\_string] )

−string1 − это строка, в которой будут заменены символы -string\_to\_replace - это строка, которая будет найдена и извлечена из string1 −[replacement\_string] - это новая строка, которая будет вставлена в string1 SELECT REPLACE('JACK and JUE', 'J') FROM DUAL; ACK and UE

Использование Псевдонимов Столбцов С Функциями

•Все функции работают со значениями, заключенными в круглые скобки, и имя каждой функции обозначает ее назначение, что полезно помнить при построении запроса •Часто псевдоним столбца используется для обозначения функции •Когда используется псевдоним столбца, псевдоним столбца отображается в выходных данных вместо фактического синтаксиса функции (SELECT LOWER(last\_name)|| LOWER(SUBSTR(first\_name,1,1)) AS "User Name" FROM employees;)

Переменные подстановки

• Чтобы использовать их, все, что вам нужно сделать, это заменить жестко заданное значение в вашем операторе на :named\_variable •Затем Oracle Application Express запросит у вас значение при выполнении вашего оператора

SELECT first\_name, last\_name, salary, department\_id FROM employees WHERE department\_id= 10; WHERE department\_id=:enter\_dept\_id;

• Переменные подстановки обрабатываются как символьные строки в Oracle Application Express, что означает, что при передаче значений символов или даты вам не нужны одинарные кавычки, которые обычно используются для заключения строк SELECT \* FROM employees WHERE last\_name = :l\_name;

**4-2 Number Functions (Числовые функции)**

• Три числовые функции являются: −ROUND −TRUNC −MOD

ROUND

• ROUND можно использовать как с числами, так и с датами• Он в основном используется для округления чисел до заданного количества знаков после запятой, но его также можно использовать для округления чисел слева от десятичной точки. Синтаксис: ROUND(column|expression, decimal places) •Если количество знаков после запятой является положительным числом, число округляется до того количества знаков после запятой, которое находится справа от десятичной точки •ROUND(45.926, 2) 45.93 •Если число знаков после запятой является отрицательным числом, число округляется до этого числа знаков после запятой места слева от десятичной точки •ROUND(45.926, -1) 50

TRUNC

• Функция TRUNC может использоваться как с числами , так и с датами. Он в основном используется для завершения столбца, выражения или значения до заданного количества знаков после запятой •При использовании TRUNC, если количество знаков после запятой не указано, то, как и при округлении, указанное число по умолчанию равно нулю. • Синтаксис: TRUNC(column|expression, decimal places) TRUNC (45.926, 2) 45.92 •Как и в случае с ROUND, если выражение TRUNC не указывает количество знаков после запятой или указывает ноль, число усекается до нуля знаков после запятой •TRUNC не округляет число• Он просто завершает число в заданной точке

MOD

• Функция MOD находит остаток после деления одного значения на другое значение • MOD можно использовать для определения того, является ли значение четным или нечетным. SELECT country\_name, MOD(airports,2) AS "Mod Demo" FROM wf\_countries;

**4-3 Date Functions (Функции даты)**

Отображение дат

• Формат отображения и ввода дат по умолчанию -:− DD-Mon-YYYY • Например: 02-Dec-2014 •Однако база данных Oracle хранит даты внутри с числовым форматом, представляющим столетие, год, месяц, день, час, минута и секунда

SYSDATE

• SYSDATE - это функция даты, которая возвращает текущую дату и время сервера базы данных SELECT SYSDATE FROM dual; 6

Тип данных ДАТЫ

• Тип данных ДАТЫ всегда хранит информацию о годе в виде внутреннего четырехзначного числа: две цифры для столетия и две цифры для года. •В предыдущих версиях компонент century по умолчанию не отображался •Однако из-за меняющихся бизнес-требований по всему миру 4-значный год теперь отображается по умолчанию

SELECT last\_name, hire\_date + 60 FROM employees; Adds 60 days to hire\_date ;

Функции даты

• Функции даты, показанные в таблице, работают на Даты оракула •Все функции даты возвращают значение с типом данных DATE, за исключением функции MONTHS\_BETWEEN, которая возвращает значение числового типа данных

MONTHS\_BETWEEN Количество месяцев между двумя датами / ADD\_MONTHS Добавить календарные месяцы к дате / NEXT\_DAY Дата следующего наступления указанного дня недели / LAST\_DAY Последний день месяца / ROUND Круглая дата / TRUNC Усечение даты

•MONTHS\_BETWEEN: принимает 2 аргумента DATE и возвращает количество календарных месяцев между 2 датами •Если первый аргумент является более ранней датой, чем второй, возвращаемое число будет отрицательным WHERE MONTHS\_BETWEEN (SYSDATE, hire\_date) > 240

•ADD\_MONTHS: принимает 2 аргумента, DATE и число. Возвращает значение DATE с номером аргумент, добавленный к компоненту месяца даты •Если заданное число отрицательное, функция вычтет это количество месяцев из аргумента даты SELECT ADD\_MONTHS (SYSDATE, 12) AS "Next Year" FROM dual;

•NEXT\_DAY: принимает 2 аргумента, DATE и день недели и возвращает ДАТУ следующего появления этого дня недели после аргумента DATE. NEXT\_DAY (SYSDATE, 'Saturday')

• LAST\_DAY: принимает аргумент DATE и возвращает DATE последнего дня месяца для аргумента DATE. SELECT LAST\_DAY (SYSDATE) AS "End of the Month" FROM dual;

• •ROUND: возвращает DATE, округленную до единицы измерения, указанной вторым аргументом. ROUND(hire\_date, 'Month')

• TRUNC: возвращает DATE, усеченную до единицы измерения, указанной вторым аргументом

SELECT employee\_id, hire\_date, ROUND(MONTHS\_BETWEEN(SYSDATE, hire\_date)) AS TENURE, ADD\_MONTHS (hire\_date, 6) AS REVIEW, NEXT\_DAY(hire\_date, 'FRIDAY'), LAST\_DAY(hire\_date) FROM employees WHERE MONTHS\_BETWEEN (SYSDATE, hire\_date) > 36;

**5-1 Conversion Functions (Функции преобразования)**

Типы данных

•В SQL существует несколько различных типов данных. Эти типы данных определяют область значений, которые может содержать каждый столбец • −VARCHAR2 −CHAR −NUMBER −DATE

•VARCHAR2: используется для символьных данных переменной длины, включая цифры, тире и специальные символы

• CHAR: используется для текстовых и символьных данных фиксированной длины, включая цифры, тире и специальные символы.

•NUMBER: используется для хранения числовых данных переменной длины. Тире, текст или другие нечисловые данные не допускаются Валюта хранится в виде числового типа данных

•DATE: используется для значений даты и времени. Oracle хранит даты в виде чисел, и по умолчанию информация о DATE отображается как DD-Mon-YYYY

Преобразование типов

• Сервер Oracle может автоматически конвертировать Данные VARCHAR2 и CHAR преобразуются в типы данных NUMBER и DATE • Он может преобразовывать данные NUMBER и DATE обратно в символьный тип данных • Это неявное преобразование данных

• Четыре функции преобразования типов данных:

−Для преобразования типа данных даты в символьный тип данных

−Для преобразования числового типа данных в символьный тип данных

−Для преобразования символьного типа данных в числовой тип данных

−Для преобразования символьного типа данных в типы данных даты

Преобразование даты в Символьные данные

•Часто бывает желательно преобразовать дату из значения по умолчанию Формат DD-Mon-YYYY в другой указанный вами формат • Функция для выполнения этой задачи заключается в: TO\_CHAR (date column name, 'format model you specify') • "Модель формата" должна быть заключена в одинарные кавычки и чувствителен к регистру

•Используйте sp, чтобы произнести число по буквам. •Используйте th, чтобы номер отображался в виде порядкового номера. −(1st, 2nd, 3rd, and so on) •Используйте элемент fm для удаления заполненных пробелов или удаления начальных нулей из выходных данных

SELECT TO\_CHAR(hire\_date, 'Month dd, YYYY') FROM employees; … June 07, 1994 / SELECT TO\_CHAR(hire\_date, 'fmDay ddth Mon, YYYY') FROM employees; Tuesday 7th Jun, 1994 / SELECT TO\_CHAR(SYSDATE, 'hh:mm:ss pm') FROM dual; 02:07:23 am

Преобразование чисел в символьные данные (VARCHAR2)

•Номера, хранящиеся в базе данных, не имеют форматирования • Это означает, что в них нет знаков/символов валюты, запятых, десятичных знаков или другого форматирования • Чтобы добавить форматирование, сначала необходимо преобразовать число в символьный формат: TO\_CHAR(number, 'format model')

SELECT TO\_CHAR(salary, '$99,999') AS "Salary" FROM employees; SELECT TO\_CHAR(3000, '$99999.99') FROM dual; $3000.00

Преобразование символов в число

•Часто бывает желательно преобразовать символьную строку в число. Функция для этого преобразования является: TO\_NUMBER(character string, 'format model') • Модель формата необязательна, но должна быть включена, если преобразуемая символьная строка содержит какие-либо символы, отличные от цифр

Преобразование символов в дату

• Чтобы преобразовать символьную строку в формат даты, используйте: TO\_DATE('character string', 'format model') • Это преобразование принимает символьную строку, не содержащую значения даты и преобразует ее в значение даты• Модель формата сообщает серверу, как "выглядит" символьная строка.: TO\_DATE('November 3, 2001', 'Month dd, yyyy')/ 03-Nov-2001

•При преобразовании символов в дату модификатор fx (format exact) задает точное соответствие для символьный аргумент и модель формата даты. SELECT TO\_DATE('May10,1989', 'fxMonDD,YYYY') AS "Convert" FROM DUAL; 10-May-1989

Правила модификатора fx

−Знаки препинания и текст в кавычках в символьном аргументе должны точно соответствовать соответствующим частям модели формата − Символьный аргумент не может содержать дополнительных пробелов (Без fx сервер Oracle игнорирует дополнительные пробелы)−Числовые данные в символьном аргументе должны содержать то же количество цифр, что и соответствующий элемент в модели формата (Без fx числа в символьном аргументе могут не содержать начальных нулей)

SELECT TO\_DATE('July312004', 'fxMonthDDYYYY') AS "Date" FROM DUAL; 31-Jul-2004

**5-2 NULL Functions (НУЛЕВЫЕ функции)**

• "Вложенность" относится к тому, что одна вещь содержится внутри другой вещи. • Следующий пример представляет собой вложенную функцию • Процесс оценки начинается с самого внутреннего уровня до самого внешнего уровня / SELECT TO\_CHAR(NEXT\_DAY(ADD\_MONTHS(hire\_date, 6), 'FRIDAY'), 'fmDay, Month ddth, YYYY') AS "Next Evaluation" FROM employees WHERE employee\_id = 100;

Функции, относящиеся к нулевым значениям

•Null - это значение, которое недоступно, неназначено, неизвестно или неприменимо • В результате мы не можем проверить, совпадает ли оно с другим значением, потому что мы не знаем, какое значение оно имеет • Он ничему не равен, даже нулю!

•Oracle имеет четыре общие функции, относящиеся к использованию нулевых значений:−NVL −NVL2 −NULLIF −COALESCE

Функция NVL

• Функция NVL преобразует нулевое значение в известное значение фиксированного типа данных, будь то дата, символ или число• Типы данных столбца нулевого значения и нового значения должны быть одинаковыми • Функция NVL является: •NVL (значение или столбец, который может содержать значение null, значение для замены null)

• В следующем запросе используется функция NVL с символьными типами данных: NVL(internet\_extension, 'None')

• Вы можете использовать функцию NVL для преобразования значений столбцов, содержащих нули, в число перед выполнением вычислений•Когда арифметическое вычисление выполняется с null, результат равен null • Функция NVL может преобразовать нулевое значение в число до выполнения арифметических вычислений, чтобы избежать получения нулевого результата

Функция NVL2

• Функция NVL2 вычисляет выражение с тремя значениями. • Если первое значение не равно null, то функция NVL2 возвращает второе выражение• Если первое значение равно null, то возвращается третье выражение.• Значения в выражении 1 могут иметь любой тип данных• Выражение 2 и выражение 3 могут иметь любой тип данных, кроме LONG • Тип данных возвращаемого значения всегда совпадает с типом данных выражения 2, если только выражение 2 не является символьными данными, и в этом случае возвращаемый тип - VARCHAR2

NVL2 (значение выражения 1, которое может содержать значение null, выражение 2 значение для возврата, если выражение 1 не равно null, значение выражения 3 для замены, если выражение 1 равно null)

NVL2(commission\_pct, salary + (salary \* commission\_pct), salary)

Функция NULLIF

• Функция NULLIF сравнивает два выражения•Если они равны, функция возвращает значение null•Если они не равны, функция возвращает первое выражение

• Функция NULLIF является: NULLIF(выражение 1, выражение 2)

NULLIF(LENGTH(first\_name), LENGTH(last\_name))•Если длина обоих имен одинакова, NULLIF возвращает значение NULL, в противном случае возвращается выражение 1 ДЛИНЫ first\_name

Функция COALESCE

• Функция COALESCE является расширением функции NVL , за исключением того, что COALESCE может принимать несколько значений • Слово " coalesce " буквально означает "собраться вместе", и именно это и происходит•Если первое выражение равно null, функция продолжается вниз по строке до тех пор, пока не будет найдено ненулевое выражение •Конечно, если первое выражение имеет значение, функция возвращает первое выражение, и функция останавливается

COALESCE (выражение 1, выражение 2, ...выражение n)

SELECT last\_name, COALESCE(commission\_pct, salary, 10) AS "Comm" FROM employees ORDER BY commission\_pct;

**5-3 Conditional Expressions (Условные Выражения)**

Условные Выражения

• Два условных выражения - CASE and DECODE / NULLIF логически эквивалентен выражению CASE при сравнении двух. •NULIF сравнивает два выражения, и если два выражения равны, то возвращает null; если они не равны, то возвращает первое выражение

• Существует два набора команд или синтаксиса, которые можно использовать для написания инструкций SQL: −Стандартные инструкции, совместимые с ANSI/ISO SQL 99 −Собственные заявления Oracle • CASE - это оператор, совместимый с ANSI/ISO 99 SQL 99 •DECODE - это проприетарное заявление Oracle•Оба оператора возвращают одну и ту же информацию, используя разный синтаксис

CASE выражение

• Выражение CASE в основном выполняет работу оператора IF-THEN-ELSE•Типы данных выражений CASE, WHEN и ELSE должны быть одинаковыми• Синтаксис выражения CASE следующий: CASE expr WHEN comparison\_expr1 THEN return\_expr1 [WHEN comparison\_expr2 THEN return\_expr2 WHEN comparison\_exprn THEN return\_exprn ELSE else\_expr] END

SELECT last\_name, CASE department\_id WHEN 90 THEN 'Management' WHEN 80 THEN 'Sales' WHEN 60 THEN 'It' ELSE 'Other dept.' END AS "Department" FROM employees;

DECODE Выражение

• Функция DECODE вычисляет выражение аналогично логике IF-THEN-ELSE• DECODE сравнивает выражение с каждым из значений поиска.• Синтаксис для DECODE следующий: DECODE(columnl|expression, search1, result1 [, search2, result2,...,] [, default])

• Если значение по умолчанию опущено, возвращается нулевое значение если значение поиска не соответствует ни одному из значений

**JOINs**

**6-1 Cross Joins and Natural Joins (Перекрестные соединения и естественные соединения)**

разделение данных на отдельные таблицы и возможность связывать таблицы друг с другом - это основа проектирования реляционных баз данных.

Join Commands

• Существует два набора команд или синтаксиса, которые можно использовать для установления соединений между таблицами в базе данных:− Oracle proprietary joins− Соединения, совместимые со стандартом ANSI/ISO SQL 99.

ANSI•ANSI расшифровывается как Американский национальный институт стандартов

SQL• Язык структурированных запросов (SQL) - это стандартный для обработки информации отраслевой язык реляционных системы управления базами данных (СУБД)

NATURAL JOIN

•Предложение SQL join объединяет поля из 2 (или более) таблиц в реляционной базе данных•Естественное соединение основано на всех столбцах в двух таблицах с одинаковыми именами и выбирает строки из двух таблиц, которые имеют одинаковые значения во всех совпадающих столбцах.

SELECT first\_name, last\_name, job\_id, job\_title FROM employees NATURAL JOIN jobs WHERE department\_id > 80;

•Как показано в примере кода, при использовании естественного соединения можно объединять таблицы без необходимости явно укажите столбцы в соответствующей таблице•Однако имена и типы данных обоих столбцов должны быть одинаковыми •Обратите внимание, что естественное соединение столбец не обязательно должен отображаться в SELECT

ПЕРЕКРЕСТНОЕ СОЕДИНЕНИЕ CROSS JOIN

• ПЕРЕКРЕСТНОЕ СОЕДИНЕНИЕ ANSI/ISO SQL: 1999 SQL соединяет каждую строку в одной таблице с каждой строкой в другой таблице • Результирующий набор представляет все возможные комбинации строк из двух таблиц. Декартово произведение.

SELECT last\_name, department\_name FROM employees CROSS JOIN departments;

**6-2 Join Clauses (Предложения о присоединении)**

USING

•При естественном соединении, если в таблицах есть столбцы с одинаковыми именами, но разными типами данных, соединение вызывает ошибку • Чтобы избежать этой ситуации, join можно изменить с помощью USING • USING указывает столбцы, которые должны использоваться для объединения• Предложение USING позволяет нам использовать WHERE для ограничения строк из одной или обеих таблиц: SELECT first\_name, last\_name, department\_id, department\_name FROM employees JOIN departments USING (department\_id) WHERE last\_name = 'Higgins';

ON

•Что делать, если столбцы, подлежащие объединению, имеют разные имена или если при объединении используются операторы сравнения с неравенством , такие как <, > или BETWEEN? •Мы не можем использовать USING, поэтому вместо этого мы используем предложение ON• Это позволяет задавать большее разнообразие условий соединения• Предложение ON также позволяет нам использовать WHERE для ограничения строк из одной или обеих таблиц • Предложение join ON требуется, когда общие столбцы имеют разные имена в двух таблицах

SELECT last\_name, job\_title FROM employees e JOIN jobs j ON (e.job\_id = j.job\_id);

•При использовании предложения ON для столбцов с одинаковыми именами в обеих таблицах необходимо добавить определитель (либо имя таблицы, либо псевдоним), в противном случае будет выдана ошибка

• Иногда вам может потребоваться извлечь данные из таблицы, в которой нет соответствующего столбца в другой таблице

SELECT last\_name, salary, grade\_level, lowest\_sal, highest\_sal FROM employees JOIN job\_grades ON(salary BETWEEN lowest\_sal AND highest\_sal);

Объединение Трех Таблиц

•Для объединения трех или более таблиц можно использовать как USING, так и ON.

SELECT last\_name, department\_name AS "Department", city FROM employees JOIN departments USING (department\_id) JOIN locations USING (location\_id);

**6-3 Inner versus Outer Joins (Внутренние и Внешние соединения)**

•В ANSI-99 SQL соединение двух или более таблиц, которое возвращает только совпадающие строки, называется внутренним соединением •Когда соединение возвращает несопоставимые строки, а также совпадающие строки, оно называется внешним соединением •Синтаксис внешнего соединения использует термины "left, full, and right". • Эти имена связаны с порядком имен таблиц в предложении FROM инструкции SELECT

LEFT and RIGHT OUTER Joins

•В приведенном примере левого внешнего соединения обратите внимание, что имя таблицы, указанное слева от слов "левое внешнее соединение", называется "левая таблица".

SELECT e.last\_name, d.department\_id, d.department\_name FROM employees e LEFT OUTER JOIN departments d ON (e.department\_id = d.department\_id);

• Это right outer join вернет все идентификаторы отделов и названия отделов, как те, которым назначены сотрудники, так и те, которые этого не делают. SELECT e.last\_name, d.department\_id, d.department\_name FROM employees e RIGHT OUTER JOIN departments d ON (e.department\_id = d.department\_id);

ПОЛНОЕ ВНЕШНЕЕ соединение

• Результирующий набор полного внешнего соединения включает все строки из левое внешнее соединение и все строки из правого внешнего соединения объединяются вместе без дублирования / SELECT e.last\_name, d.department\_id, d.department\_name FROM employees e FULL OUTER JOIN departments d ON (e.department\_id = d.department\_id);

**6-4 Self-Joins and Hierarchical Queries (Самосоединения и иерархические запросы)**

• Чтобы присоединить таблицу к самой себе, таблице присваиваются два имени или псевдонима. Это заставит базу данных "думать", что есть две таблицы./ SELECT worker.last\_name, worker.manager\_id, manager.last\_name AS "Manager name" FROM employees worker JOIN employees manager ON (worker.manager\_id = manager.employee\_id);

Иерархические запросы

• С самосоединениями тесно связаны иерархические запросы •С помощью такого типа запросов мы можем создать Организацию Диаграмма, показывающая структуру компании или отдела• Представьте себе генеалогическое древо, где старшие члены семьи находятся близко к основанию или стволу дерева, а младшие члены представляют ветви дерева. • Филиалы могут иметь свои собственные филиалы и так далее•Используя иерархические запросы, вы можете извлекать данные на основе естественной иерархической взаимосвязи между строками в таблице. •Иерархический запрос - это метод представления отчетов о ветвях дерева в определенном порядке

•Иерархические запросы имеют свои собственные новые ключевые слова: START WITH, CONNECT BY PRIOR, and LEVEL / • START WITH определяет, какую строку использовать в качестве корня для строящегося дерева, CONNECT BY PRIOR объясняет , как выполнять соединения между строками, а LEVEL указывает , сколько ветвей в глубину будет проходить дерево.

SELECT employee\_id, last\_name, job\_id, manager\_id FROM employees START WITH employee\_id = 100 CONNECT BY PRIOR employee\_id = manager\_id

SELECT last\_name ||' reports to ' || PRIOR last\_name AS "Walk Top Down" FROM employees START WITH last\_name = 'King' CONNECT BY PRIOR employee\_id = manager\_id;

• УРОВЕНЬ - это псевдоколонка, используемый с иерархическими запросы, и он подсчитывает количество шагов, которые он предпринял от корня дерева / SELECT LEVEL, last\_name || ' reports to ' || PRIOR last\_name AS "Walk Top Down" FROM employees START WITH last\_name = 'King' CONNECT BY PRIOR employee\_id = manager\_id;

Отчет по иерархическому запросу: •Если вы хотите создать отчет, отображающий уровни управления компанией, начиная с самого высокого уровня и делая отступы на каждом из следующих уровней, то это было бы легко сделать с помощью псевдоколонки LEVEL и функции LPAD для отступов сотрудников на основе их уровня.

Сокращение иерархических запросов

• Обрезка ветвей из дерева может быть выполнена с использованием либо предложения WHERE, либо предложения CONNECT BY PRIOR•Если используется предложение WHERE, исключается только строка, названная в инструкции; если используется предложение CONNECT BY PRIOR, исключается вся ветвь. / SELECT last\_name FROM employees WHERE last\_name != 'Higgins' START WITH last\_name = 'Kochhar' CONNECT BY PRIOR employee\_id = manager\_id;

•Если, однако, вы хотите исключить одну строку и все строки под ней, вам следует сделать исключение частью инструкции CONNECT BY. •В этом примере, который исключает Хиггинса, мы также исключаем Гитца в результате

SELECT last\_name FROM employees START WITH last\_name = 'Kochhar' CONNECT BY PRIOR employee\_id = manager\_id AND last\_name != 'Higgins';

**7-1 Oracle Equijoin and Cartesian Product (Oracle Equi join и декартово произведение)**

• Два набора команд или синтаксиса, которые могут использоваться для установления соединений между таблицами в базе данных:−Проприетарные соединения Oracle−ANSI/ISO SQL: 99 совместимых стандартных соединений

• Сравнение проприетарных соединений Oracle с ANSI/ISO SQL:

Oracle Proprietary Join ANSI/ISO SQL: 1999 Equivalent

Перекрестное соединение декартовых произведений

Равносоединенный (Equijoin) NATURAL JOIN // JOIN USING clause // JOIN ON

Non-equijoin ON clause

Проприетарные соединения ORACLE

• Для запроса данных из более чем одной таблицы с помощью Проприетарный синтаксис Oracle, используйте условие соединения в WHERE / SELECT table1.column, table2.column FROM table1, table2 WHERE table1.column1 = table2.column2;

• Чтобы упростить чтение инструкции Join и ускорить доступ к базе данных, рекомендуется предварять имя столбца именем таблицы.• Это называется "квалификация ваших столбцов".• Комбинация имени таблицы и имени столбца помогает устранить неоднозначные имена, когда две таблицы содержат столбец с одинаковым именем столбца •Если в обеих таблицах отображается одно и то же имя столбца, имя столбца должно предшествовать имени таблицы

EQUIJOIN - РАВНОСОЕДИНЕННЫЙ

• Иногда называемое "простым" или "внутренним" соединением, equijoin - это объединение таблиц, объединяющее строки, имеющие одинаковые значения для указанных столбцов. •Equijion равнозначен ANSI: −NATURAL JOIN −JOIN USING −JOIN ON (когда условие соединения использует "=")

SELECT определяет имена столбцов для отображения / FROM указывает таблицы, к которым должна обращаться база данных, разделенные запятыми / WHERE определяет, как должны быть соединены таблицы

Псевдонимы

•Работа с длинными именами столбцов и таблиц может быть громоздкой• есть способ сократить синтаксис с помощью псевдонимов• Чтобы различать столбцы с одинаковыми именами, но находящиеся в разных таблицах, используйте псевдонимы таблиц •Псевдоним таблицы аналогичен псевдониму столбца; он переименовывает объект в инструкции•Он создается путем ввода нового имени для таблицы сразу после имени таблицы в предложении from

•Если в предложении FROM используется псевдоним таблицы, то этот псевдоним таблицы должен быть заменен именем таблицы во всей инструкции SELECT•Использование имени таблицы в предложении SELECT, которому присвоен псевдоним в предложении FROM, приведет к ошибке

Соединение декартовых произведений (Cartesian Product Join)

•Если две таблицы в запросе на объединение не имеют условия соединения , указанного в предложении WHERE, или условие соединения недопустимо, сервер Oracle возвращает декартово произведение двух таблиц• Это комбинация каждой строки одной таблицы с каждой строкой другой •Декартово произведение эквивалентно ANSI CROSS JOIN• Чтобы избежать декартова произведения, всегда включайте допустимое условие соединения в WHERE

SELECT employees.last\_name, departments.department\_name FROM employees, departments;

Ограничение Строк В Соединении

•Как и в случае запросов с одной таблицей, предложение WHERE может использоваться для ограничения строк, рассматриваемых в одной или нескольких таблицах соединения• Показанный запрос использует оператор AND для ограничения возвращаемых строк

SELECT employees.last\_name, employees.job\_id, jobs.job\_title FROM employees, jobs WHERE employees.job\_id = jobs.job\_id AND employees.department\_id = 80;

• Чтобы объединить три таблицы, вам необходимо добавить еще одно условие объединения в WHERE с помощью оператора AND

SELECT last\_name, city FROM employees e, departments d, locations l WHERE e.department\_id = d.department\_id AND d.location\_id = l.location\_id;

**7-2 Oracle Nonequijoins and Outer Joins (Неэквивалентные соединения Oracle и внешние соединения)**

Nonequijoins

• Поскольку нет точного совпадения между этими двумя столбцов в каждой таблице, оператор равенства = не может быть использован•Хотя могут использоваться такие условия сравнения, как <= и >=, МЕЖДУ...И является более эффективным способом выполнения неравновесного соединения•Неэквивалентное соединение эквивалентно ANSI JOIN ON (где используется условие, отличное от equals)

SELECT last\_name, salary, grade\_level, lowest\_sal, highest\_sal FROM employees, job\_grades WHERE (salary BETWEEN lowest\_sal AND highest\_sal);

Внешнее соединение Outer Join

•Внешнее соединение используется для просмотра строк, имеющих соответствующее значение в другой таблице, плюс те строки в одной из таблиц, которые не имеют соответствующего значения в другой таблице • Чтобы указать, в какой таблице могут отсутствовать данные, используя Синтаксис соединения Oracle, добавьте знак плюс (+) после имени столбца таблицы в WHERE запроса

SELECT e.last\_name, d.department\_id, d.department\_name FROM employees e, departments d WHERE e.department\_id = d.department\_id(+);

• Этот запрос вернет все фамилии сотрудников , включая те, которые назначены отделу, и те, которые не являются• Те же результаты можно было бы получить, используя ANSI LEFT OUTER JOIN

SELECT e.last\_name, d.department\_id, d.department\_name FROM employees e, departments d WHERE e.department\_id =(+) d.department\_id(+); - ANSI RIGHT OUTER JOIN

•Невозможно получить эквивалент ПОЛНОГО ВНЕШНЕЕ СОЕДИНЕНИЕ путем добавления знака (+) к обоим столбцам в условии соединения •Попытка выполнить это приводит к ошибке

Внешнее соединение и эквиваленты ANSI

• В таблице ниже показаны соединения ANSI/ISO SQL: 99 и эквивалентные им внешние соединения Oracle

**Group Functions**

**8-1 Group Functions (Групповые функции)**

•В SQL следующие групповые функции могут работать со всей таблицей или с определенной группой строк • Каждая функция возвращает один результат

•Групповые функции−AVG −COUNT −MIN −MAX −SUM −VARIANCE −STDDEV

• MIN: используется со столбцами, в которых хранятся данные любого типа, для возврата минимального значения

• MAX: используется со столбцами, в которых хранятся данные любого типа, для возврата максимальное значение SELECT MAX(salary) FROM employees;

• SUM: используется со столбцами, в которых хранятся числовые данные, для нахождения общего числа или суммы значений.

• AVG: используется со столбцами, в которых хранятся числовые данные, для вычисления среднего значения SELECT ROUND(AVG(salary)) FROM employees WHERE department\_id = 90;

• COUNT: Возвращает количество строк

•VARIANCE: используется со столбцами, в которых хранятся числовые данные , для вычисления разброса данных вокруг среднего значения. SELECT ROUND(VARIANCE(life\_expect\_at\_birth),4) FROM wf\_countries;

• STDDEV: Аналогично дисперсии, стандартное отклонение измеряет разброс данных. SELECT ROUND(STDDEV(life\_expect\_at\_birth), 4) FROM wf\_countries;

SELECT column, group\_function(column), .. FROM table WHERE condition GROUP BY column;

Правила для групповых функций: •Групповые функции игнорируют нулевые значения•Групповые функции не могут быть использованы в предложении WHERE•MIN, MAX и COUNT можно использовать с любым типом данных;SUM, AVG, STDDEV и ДИСПЕРСИЯ могут использоваться только с числовыми типами данных

**9-1 Using Group By and Having Clauses (Использование Group By и Having предложения)**

Вы используете предложение GROUP BY, чтобы разделить строки в таблице на более мелкие группы. Затем можно использовать групповые функции для возврата сводной информации для каждой группы. В показанном операторе SELECT строки группируются по ИД\_отдела.

Затем к каждой группе применяется функция AVG. Пример:

SELECT department\_id, AVG(salary) FROM employees GROUP BY department\_id ORDER BY department\_id;

Допустим мы захотели найти максимальную зарплату сотрудников в каждом отделе. Для этого мы используем предложение GROUP BY, указывающее, какой столбец использовать для группировки строк: SELECT MAX(salary) FROM employees GROUP BY department\_id;

Обычно мы хотим включить столбец GROUP BY в список SELECT:

SELECT department\_id, MAX(salary) FROM employees GROUP BY department\_id;

Групповые функции требуют, чтобы любой столбец, указанный в предложении SELECT, который не является частью групповой функции, был указан в предложении GROUP BY.

В следующем примере показано, сколько стран находится в каждом регионе: SELECT COUNT(country\_name), region\_id FROM wf\_countries GROUP BY region\_id ORDER BY region\_id;

Помните, что групповые функции игнорируют нулевые значения, поэтому, если какая-либо страна не имеет названия страны, она не будет включена в COUNT.

Мы также можем использовать предложение WHERE для исключения строк до того, как оставшиеся строки будут объединены в группы: SELECT department\_id, MAX(salary) FROM employees WHERE last\_name != 'King' GROUP BY department\_id;

Важные рекомендации, которые следует помнить при использовании предложения GROUP BY:

− Если вы включаете групповую функцию (AVG, SUM, COUNT, MAX, MIN, STDDEV, VARIANCE) в предложение SELECT вместе с любыми другими отдельными столбцами, каждый отдельный столбец также должен отображаться в предложении GROUP BY.

− Вы не можете использовать псевдоним столбца в предложении GROUP BY.

− Предложение WHERE исключает строки до того, как они будут разделены на группы.

Иногда вам нужно разделить группы на более мелкие группы

Например, вы можете захотеть сгруппировать всех сотрудников по отделам; затем внутри каждого отдела сгруппируйте их по должностям

Этот пример показывает, сколько сотрудников выполняют каждую работу в каждом отделе:

SELECT department\_id, job\_id, count(\*) FROM employees WHERE department\_id > 40 GROUP BY department\_id, job\_id;

Вложенные групповые функции. Групповые функции могут быть вложены на глубину до двух при использовании GROUP BY:

SELECT max(avg(salary)) FROM employees GROUP by department\_id; Запрос найдет среднюю зарплату для каждого отдела, а затем из этого списка выберет единственное наибольшее значение.

Точно так же, как мы использовали предложение WHERE для ограничения выбранных строк, мы можем использовать предложение HAVING для ограничения групп. В запросе с использованием предложения GROUP BY и HAVING строки сначала группируются, применяются групповые функции, а затем отображаются только те группы, которые соответствуют предложению HAVING. Предложение WHERE используется для ограничения строк; предложение HAVING используется для ограничения групп, возвращаемых из предложения GROUP BY:

SELECT department\_id,MAX(salary) FROM employees GROUP BY department\_id HAVING COUNT(\*)>1 ORDER BY department\_id;

Следующий запрос находит среднюю численность населения стран в каждом регионе. Затем он возвращает только группы регионов с наименьшим населением более трехсот тысяч человек:

SELECT region\_id, ROUND(AVG(population)) FROM wf\_countries GROUP BY region\_id HAVING MIN(population)>300000 ORDER BY region\_id;

Хотя предложение HAVING может предшествовать предложению GROUP BY в операторе SELECT, рекомендуется размещать каждое предложение в указанном порядке. Предложение ORDER BY (если используется) всегда последнее!

SELECT column, group\_function FROM table WHERE GROUP BY HAVING ORDER BY

**9-2 Using Rollup and Cube Operations, and Grouping Sets (Использование операций свертки и куба, а также наборов группировок)**

В запросах GROUP BY довольно часто требуется подводить промежуточные итоги и итоги, и операция ROLLUP может сделать это за вас. Без использования оператора ROLLUP такое требование означало бы написание нескольких запросов, а затем ввод результатов, например, в электронную таблицу для расчета и форматирования результатов. ROLLUP создает промежуточные итоги, которые переходят от самого подробного уровня к общему итогу, используя список группировки, указанный в предложении GROUP BY. Действие ROLLUP простое: он создает промежуточные итоги, которые переходят от самого подробного уровня к общему итогу. ROLLUP использует упорядоченный список столбцов группировки в своем списке аргументов. Во-первых, он вычисляет стандартные совокупные значения, указанные в предложении GROUP BY. Затем он создает промежуточные итоги более высокого уровня, двигаясь справа налево по списку группирующих столбцов. Наконец, это создает общую сумму. Пример:

SELECT department\_id, job\_id, SUM(salary) FROM employees WHERE department\_id < 50 GROUP BY ROLLUP (department\_id, job\_id);

Количество столбцов или выражений, отображаемых в списке аргументов ROLLUP, определяет количество группировок. Формула (количество столбцов) + 1, где количество столбцов — это количество столбцов, перечисленных в списке аргументов ROLLUP. В приведенном ниже примере запроса два столбца перечислены в списке аргументов ROLLUP, поэтому мы увидим три значения, сгенерированные автоматически:

SELECT department\_id, job\_id, SUM(salary) FROM employees WHERE department\_id < 50 GROUP BY ROLLUP (department\_id, job\_id);

CUBE, как и ROLLUP, является расширением предложения GROUP BY. Он создает отчеты с перекрестными таблицами. Его можно применять ко всем агрегатным функциям, включая AVG, SUM, MIN, MAX и COUNT. Столбцы, перечисленные в предложении GROUP BY, содержат перекрестные ссылки для создания надмножества групп. Агрегированные функции, указанные в списке SELECT, применяются к этой группе для создания сводных значений для дополнительных строк суперагрегата. Каждая возможная комбинация строк агрегируется CUBE. Если имеется n столбцов в предложении GROUP BY, будет 2n возможных комбинаций суперагрегатов. Математически эти комбинации образуют многомерный куб, поэтому оператор и получил свое название. CUBE часто используется в запросах, в которых используются столбцы из отдельных таблиц, а не отдельные столбцы из одной таблицы. Пример: SELECT department\_id, job\_id, SUM(salary) FROM employees WHERE department\_id < 50 GROUP BY CUBE (department\_id, job\_id);

GROUPING SETS — еще одно расширение предложения GROUP BY. Он используется для указания нескольких групп данных. Это дает нам возможность иметь несколько предложений GROUP BY в одном операторе SELECT, что не допускается в обычном синтаксисе.

Если вы хотите видеть данные из таблицы EMPLOYEES, сгруппированные по (department\_id, job\_id, manager\_id). Но также сгруппированы по (department\_id, manager\_id). А также сгруппированные по (job\_id, manager\_id), тогда вам обычно придется писать три разных оператора выбора, единственная разница между которыми заключается в предложениях GROUP BY. Для базы данных это означает получение одних и тех же данных три раза, а это может привести к довольно большим накладным расходам. Представим, если бы в компании было 3 000 000 сотрудников. Затем нужна база данных, где получаем 9 миллионов строк вместо 3 миллионов строк — довольно большая разница. Таким образом, GROUPING SETS намного эффективнее при написании сложных отчетов. Пример: SELECT department\_id, job\_id, manager\_id, SUM(salary) FROM employees WHERE department\_id < 50 GROUP BY GROUPING SETS ((job\_id, manager\_id),(department\_id, job\_id), (department\_id, manager\_id));

Когда вы используете ROLLUP или CUBE для создания отчетов с промежуточными итогами, вам довольно часто также нужно иметь возможность определить, какие строки в выходных данных являются фактическими строками, возвращенными из базы данных, а какие строки являются вычисленными строками промежуточных итогов, полученными в результате операций ROLLUP или CUBE. Как определить разницу между сохраненным значением NULL, возвращаемым запросом, и значениями NULL, созданными с помощью ROLLUP или CUBE. Функция GROUPING решает эти проблемы. Используя один столбец из запроса в качестве аргумента, функция GROUPING вернет 1 для агрегированной (вычисленной) строки и 0 для неагрегированной (возвращенной) строки. Синтаксис для GROUPING просто: GROUPING (column\_name). Он используется только в предложении SELECT и принимает в качестве аргумента только одно выражение столбца. Пример:

SELECT department\_id, job\_id, SUM(salary), GROUPING(department\_id) AS "Dept sub total", GROUPING(job\_id) AS "Job sub total" FROM employees WHERE department\_id < 50 GROUP BY CUBE (department\_id, job\_id);

**9-3 Using Set Operators (Использование операторов множества)**

Для объяснения операторов SET в этом уроке будут использоваться следующие два списка:

• А = {1, 2, 3, 4, 5}

• В = {4, 5, 6, 7, 8}

При использовании операторов SET следует помнить несколько правил:

− Количество столбцов и типы данных столбцов должны быть одинаковыми во всех операторах SELECT, используемых в запросе.

− Имена столбцов не обязательно должны быть идентичными.

− Имена столбцов в выводе берутся из имен столбцов в первом операторе SELECT.

Таким образом, любые псевдонимы столбцов должны быть введены в первую инструкцию, поскольку вы хотели бы видеть их в готовом отчете.

Оператор UNION возвращает все строки из обеих таблиц после исключения дубликатов. Пример: SELECT a\_id FROM a UNION SELECT b\_id FROM b; Результат перечисления всех элементов в A и B с исключением дубликатов: {1, 2, 3, 4, 5, 6, 7, 8}. Если вы объедините A и B, вы получите только {4, 5}. Вам нужно будет выполнить полное внешнее соединение, чтобы получить тот же список.

Оператор UNION ALL возвращает все строки из обеих таблиц, не удаляя дубликаты. Пример: SELECT a\_id FROM a UNION ALL SELECT b\_id FROM b; Результат перечисления всех элементов в A и B без исключения дубликатов: {1, 2, 3, 4, 5, 4, 5, 6, 7, 8}.

Оператор INTERSECT возвращает все строки, общие для обеих таблиц. Пример: SELECT a\_id FROM a INTERSECT SELECT b\_id FROM b; Результат перечисления всех элементов, найденных как в A, так и в B, равен {4, 5}.

Оператор MINUS возвращает все строки, найденные в одной таблице, но не в другой. Пример: SELECT a\_id FROM a MINUS SELECT b\_id FROM b; Результат перечисления всех элементов, найденных в A, но не в B, равен {1, 2, 3}. Результат B MINUS A даст {6, 7, 8}.

Иногда, если мы выбираем строки из таблиц, которые не имеют общих столбцов, нам может потребоваться создать свои собственные столбцы, чтобы соответствовать количеству столбцов в запросах. Самый простой способ сделать это — включить одно или несколько значений NULL в список выбора.

Главное не забыть дать каждому подходящий псевдоним и соответствующий тип данных. Пример: SELECT hire\_date, employee\_id, job\_id FROM employees UNION SELECT TO\_DATE(NULL),employee\_id, job\_id FROM job\_history;

Ключевое слово NULL может использоваться для сопоставления столбцов в списке SELECT. Один NULL включен для каждого отсутствующего столбца. Кроме того, NULL форматируется в соответствии с типом данных столбца, для которого он используется, поэтому функции TO\_CHAR, TO\_DATE или TO\_NUMBER используются для получения идентичных списков SELECT.

Если мы хотим контролировать порядок возвращаемых строк при использовании операторов SET в вашем запросе, оператор ORDER BY должен использоваться только один раз, в последнем операторе SELECT в запросе. Используя предыдущий пример запроса, мы могли бы ORDER BY employee\_id просмотреть рабочие места, которые занимал каждый сотрудник: SELECT hire\_date, employee\_id, job\_id FROM employees UNION SELECT TO\_DATE(NULL),employee\_id, job\_id FROM job\_history ORDER BY employee\_id;

Мы могли бы улучшить читаемость вывода, включив столбцы даты начала и даты окончания из таблицы истории заданий, для этого нам нужно было бы сопоставить столбцы в обоих запросах, добавив еще два столбца TO\_DATE(NULL) к первому запросу:

SELECT hire\_date, employee\_id, TO\_DATE(null) start\_date, TO\_DATE(null) end\_date, job\_id, department\_id FROM employees UNION SELECT TO\_DATE(null), employee\_id, start\_date, end\_date, job\_id, department\_id FROM job\_history ORDER BY employee\_id;

**Subqueries**

**10-1 Fundamentals of Subqueries (Основы подзапросов)**

Если бы мы захотели написать запрос, но обнаружили, что у нас нет всей информации, необходимой для его построения. Мы можем решить эту проблему, вложив запросы — поместив один запрос внутри другого запроса. Внутренний запрос называется «подзапросом». Подзапрос выполняется, чтобы найти информацию, которую мы не знаем. Внешний запрос использует эту информацию, чтобы выяснить, что нам нужно знать. Возможность объединить два запроса в один может быть очень полезна, когда нам нужно выбрать строки из таблицы с условием, которое зависит от данных в самой таблице.

Подзапрос — это оператор SELECT, встроенный в предложение другого оператора SELECT. Подзапрос выполняется один раз перед основным запросом. Результат подзапроса используется основным или внешним запросом. Подзапросы могут быть помещены в ряд предложений SQL, включая предложение WHERE, предложение HAVING и предложение FROM. Синтаксис подзапроса: SELECT select\_list FROM table WHERE expression operator (SELECT select\_list FROM table);

− Подзапрос заключен в круглые скобки

− Подзапрос размещается справа от условия сравнения

−Внешние и внутренние запросы могут получать данные из разных таблиц.

− Для оператора SELECT можно использовать только одно предложение ORDER BY; если он используется, он должен быть последним предложением во внешнем запросе.

− Подзапрос не может иметь собственного предложения ORDER BY.

− Единственным ограничением на количество подзапросов является размер буфера, который использует запрос.

Существует два типа подзапросов:

− Однострочные подзапросы, использующие однострочные операторы (>, =, >=, <, <>, <=) и возвращающие только одну строку из внутреннего запроса.

− Многострочные подзапросы, использующие многострочные операторы (IN, ANY, ALL) и возвращающие более одной строки из внутреннего запроса.

Пример подзапроса: SELECT first\_name, last\_name, hire\_date FROM employees WHERE hire\_date > (SELECT hire\_date FROM employees WHERE last\_name = 'Vargas');

Если подзапрос возвращает нулевое значение или не возвращает ни одной строки, внешний запрос берет результаты подзапроса (нулевой) и использует этот результат в своем предложении WHERE. Тогда внешний запрос не вернет ни одной строки, потому что сравнение любого значения с нулем всегда дает нулевое значение. Пример: SELECT last\_name FROM employees WHERE department\_id = (SELECT department\_id FROM employees WHERE last\_name = 'Grant');

**10-2 Single-Row Subqueries (Однострочные подзапросы)**

Факты об однострочных подзапросах:

• Они:

− Возвращают только одну строку

− Используют однострочные операторы сравнения (=, >,>=, <, <=, <>)

•Всегда:

− Заключать подзапрос в круглые скобки

− Помещать подзапрос в правую часть условия сравнения.

Внешние и внутренние запросы могут получать данные из разных таблиц. Для оператора SELECT можно использовать только одно предложение ORDER BY, и если оно указано, оно должно быть последним предложением в основном операторе SELECT. Единственным ограничением на количество подзапросов является размер буфера, который использует запрос.

Внешние и внутренние запросы могут получать данные из разных таблиц: SELECT last\_name, job\_id, department\_id FROM employees WHERE department\_id = (SELECT department\_id FROM departments WHERE department\_name = 'Marketing') ORDER BY job\_id;

Более одного подзапроса могут возвращать информацию внешнему запросу: SELECT last\_name, job\_id, salary, department\_id FROM employees WHERE job\_id = (SELECT job\_id FROM employees WHERE employee\_id = 141) AND department\_id = (SELECT department\_id FROM departments WHERE location\_id = 1500);

Групповые функции могут использоваться в подзапросах. Групповая функция без предложения GROUP BY в подзапросе возвращает одну строку. Пример, где подзапрос сначала находит среднюю зарплату для всех сотрудников, затем внешний запрос возвращает сотрудников с зарплатой ниже средней: SELECT last\_name, salary FROM employees WHERE salary < (SELECT AVG(salary) FROM employees);

Подзапросы также могут быть помещены в предложение HAVING. HAVING похоже на предложение WHERE, за исключением того, что предложение HAVING используется для ограничения групп и всегда включает групповую функцию, такую как MIN, MAX или AVG. Поскольку предложение HAVING всегда включает групповую функцию, подзапрос почти всегда также включает групповую функцию. Внешний запрос использует это значение для выбора идентификатора отдела и самой низкой заработной платы всех отделов, чья самая низкая заработная плата больше этого числа. Пример: SELECT department\_id, MIN(salary) FROM employees GROUP BY department\_id HAVING MIN(salary) > (SELECT MIN(salary) FROM employees WHERE department\_id = 50);

**10-3 Multiple-Row Subqueries (Многострочные подзапросы)**

Следующий пример возвращает ошибку, поскольку в отделе 20 существует более одного сотрудника, подзапрос возвращает несколько строк: SELECT first\_name, last\_name FROM employees WHERE salary = (SELECT salary FROM employees WHERE department\_id = 20);

Их называют многострочным подзапросом. Проблема в прирмере заключается в знаке равенства (=) в предложении WHERE внешнего запроса.

Подзапросы, возвращающие более одного значения, называются многострочными подзапросами. Поскольку мы не можем использовать операторы сравнения одной строки (=, < и т. д.), нам нужны разные операторы сравнения для подзапросов с несколькими строками.

Многострочные операторы:

−IN

−ANY

−ALL

Оператор NOT можно использовать с любым из этих трех операторов.

Оператор IN используется во внешнем запросе WHERE для выбора только тех строк, которые находятся в списке значений, возвращаемых из внутреннего запроса. Пример: SELECT last\_name, hire\_date FROM employees WHERE EXTRACT(YEAR FROM hire\_date) IN (SELECT EXTRACT(YEAR FROM hire\_date) FROM employees WHERE department\_id=90);

Оператор ANY используется, когда мы хотим, чтобы предложение WHERE внешнего запроса выбирало строки, соответствующие критериям (<, >, = и т. д.) хотя бы одного значения в результирующем наборе подзапроса. Пример: SELECT last\_name, hire\_date FROM employees WHERE EXTRACT(YEAR FROM hire\_date) < ANY (SELECT EXTRACT(YEAR FROM hire\_date) FROM employees WHERE department\_id=90);

Оператор ALL используется, когда мы хотим, чтобы предложение WHERE внешнего запроса выбирало строки, которые соответствуют критериям (<, >, = и т. д.) всех значений в результирующем наборе подзапроса. Оператор ALL сравнивает значение с каждым значением, возвращаемым внутренним запросом. Пример: SELECT last\_name, hire\_date FROM employees WHERE EXTRACT(YEAR FROM hire\_date) < ALL (SELECT EXTRACT(YEAR FROM hire\_date) FROM employees WHERE department\_id=90);

Предположим, что одно из значений, возвращаемых многострочным подзапросом, равно null, а другие значения — нет. Если используется IN или ANY, внешний запрос вернет строки, соответствующие ненулевым значениям: SELECT last\_name, employee\_id FROM employees WHERE employee\_id IN (SELECT manager\_id FROM employees);

Если используется ALL, внешний запрос не возвращает ни одной строки, поскольку ALL сравнивает строку внешнего запроса с каждым значением, возвращаемым подзапросом, включая null. И сравнение чего-либо с нулем приводит к нулю. Пример: SELECT last\_name, employee\_id FROM employees WHERE employee\_id <= ALL (SELECT manager\_id FROM employees);

Предложение GROUP BY и предложение HAVING также можно использовать с многострочными подзапросами. Пример: SELECT department\_id, MIN(salary) FROM employees GROUP BY department\_id HAVING MIN(salary) < ANY (SELECT salary FROM employees WHERE department\_id IN (10,20)) ORDER BY department\_id;

Подзапросы могут использовать один или несколько столбцов. Если они используют более одного столбца, они называются подзапросами с несколькими столбцами. Подзапрос с несколькими столбцами может быть либо парным сравнением, либо непарным сравнением. Пример: SELECT employee\_id, manager\_id, department\_id FROM employees WHERE(manager\_id,department\_id) IN (SELECT manager\_id,department\_id FROM employees WHERE employee\_id IN (149,174)) AND employee\_id NOT IN (149,174);

Непарный подзапрос с несколькими столбцами также использует более одного столбца в подзапросе, но сравнивает их по одному, поэтому сравнения выполняются в разных подзапросах. Нужно будет написать один подзапрос для каждого столбца, с которым мы хотим сравнить, при выполнении непарных подзапросов из нескольких столбцов: SELECT employee\_id, manager\_id, department\_id FROM employees WHERE manager\_id IN (SELECT manager\_id FROM employees WHERE employee\_id IN (149,174)) AND department\_id IN (SELECT department\_id FROM employees WHERE employee\_id IN (149,174)) AND employee\_id NOT IN(149,174);

Некоторые подзапросы могут возвращать одну или несколько строк в зависимости от значений данных в строках. Если существует даже малейшая возможность возврата нескольких строк, нужно убедится, что написали многострочный подзапрос. Пример: SELECT first\_name, last\_name, job\_id FROM employees WHERE job\_id = (SELECT job\_id FROM employees WHERE last\_name = 'Ernst');

Было бы лучше написать подзапрос из нескольких строк. Синтаксис подзапроса с несколькими строками будет работать, даже если подзапрос возвращает одну строку. Пример: SELECT first\_name, last\_name, job\_id FROM employees WHERE job\_id IN (SELECT job\_id FROM employees WHERE last\_name = 'Ernst');

**10-4 Correlated Subqueries (Коррелированные подзапросы)**

Сервер Oracle выполняет коррелированный подзапрос, когда подзапрос ссылается на столбец из таблицы, указанной в родительском операторе. Коррелированный подзапрос оценивается один раз для каждой строки, обрабатываемой родительским оператором. Родительским оператором может быть оператор SELECT, UPDATE или DELETE.

Коррелированные подзапросы используются для построчной обработки. Каждый подзапрос выполняется один раз для каждой строки внешнего запроса. В обычном подзапросе внутренний запрос SELECT запускается первым и выполняется один раз, возвращая значения, которые будут использоваться внешним запросом. Однако коррелированный подзапрос выполняется один раз для каждой строки, рассматриваемой внешним запросом. Другими словами, внутренний запрос управляется внешним запросом. Пример: SELECT o.first\_name, o.last\_name, o.salary FROM employees o WHERE o.salary > (SELECT AVG(i.salary) FROM employees i WHERE i.department\_id = o.department\_id);

EXISTS и его противоположность NOT EXISTS — это два предложения, которые можно использовать при проверке совпадений в подзапросах. EXISTS проверяет TRUE или совпадающий результат в подзапросе. Пример: SELECT last\_name AS "Not a Manager" FROM employees emp WHERE NOT EXISTS (SELECT \* FROM employees mgr WHERE mgr.manager\_id = emp.employee\_id);

Если тот же запрос выполняется с NOT IN вместо NOT EXISTS, результат будет совсем другим. Пример: SELECT last\_name AS "Not a Manager" FROM employees emp WHERE emp.employee\_id NOT IN (SELECT mgr.manager\_id FROM employees mgr); Подзапросы могут возвращать три значения: TRUE, FALSE и UNKNOWN. Нужно остерегаться NULL в подзапросах при использовании IN или NOT IN. Если не уверены, будет ли подзапрос содержать нулевое значение, либо нужно удалить нулевое значение, либо нужно использовать IS NOT NULL в предложении WHERE.

Если нам нужно написать очень сложный запрос с многократно используемыми объединениями и агрегациями, мы можем написать разные части оператора в виде блоков запроса, а затем использовать те же самые блоки запроса в операторе SELECT. Oracle позволяет нам писать именованные подзапросы в одном операторе, если мы начинаем свой оператор с ключевого слова WITH. Предложение WITH извлекает результаты одного или нескольких блоков запроса и сохраняет эти результаты для пользователя, выполняющего запрос.

Предложение WITH повышает производительность. Предложение WITH упрощает чтение запроса. Синтаксис предложения WITH следующий: WITH subquery-name AS (subquery), subquery-name AS (subquery) SELECT column-list FROM {table | subquery-name | view} WHERE condition is true;

Пример: WITH managers AS (SELECT DISTINCT manager\_id FROM employees WHERE manager\_id IS NOT NULL) SELECT last\_name AS "Not a manager" FROM employees WHERE employee\_id NOT IN (SELECT \* FROM managers);

**11-1 Ensuring Quality Query Results (Обеспечение качества результатов запроса)**

**DML**

**12-1 INSERT Statements (Операторы INSERT)**

Синтаксис создания таблиц: CREATE TABLE copy\_tablename AS (SELECT \* FROM tablename); Пример: CREATE TABLE copy\_ departments AS (SELECT \* FROM departments);

Чтобы проверить и просмотреть копию таблицы, используйют следующие операторы DESCRIBE и SELECT: DESCRIBE copy\_departments; SELECT \* FROM copy\_ departments;

Оператор INSERT используется для добавления новой строки в таблицу. Оператор требует три значения:

− название таблицы

− имена столбцов в таблице для заполнения

− соответствующие значения для каждого столбца

Синтаксис ниже использует INSERT для добавления нового отдела в таблицу copy\_departments. Этот оператор явно перечисляет каждый столбец в том виде, в котором он представлен в таблице. Значения для каждого столбца перечислены в одном и том же порядке. Числовые значения не заключаются в одинарные кавычки. INSERT INTO copy\_departments (department\_id, department\_name, manager\_id, location\_id) VALUES (200,'Human Resources', 205, 1500);

Другой способ вставки значений в таблицу — добавить их неявно, опуская имена столбцов. Однако значения для каждого столбца должны точно соответствовать порядку по умолчанию, в котором они появляются в таблице (как показано в операторе DESCRIBE), и значение должно быть предоставлено для каждого столбца. Пример: INSERT INTO copy\_departments VALUES (210,'Estate Management', 102, 1700);

Оператор INSERT в этом примере был написан без явного указания столбцов. Однако для ясности лучше использовать имена столбцов в предложении INSERT.

Прежде чем вставлять данные в таблицу, нужно проверить несколько деталей таблицы. Оператор DESCRIBE tablename вернет описание структуры таблицы в сводной диаграмме таблицы. Сводка таблицы предоставляет информацию о каждом столбце в таблице, например:

−допуск дублирования значений

− тип разрешенных данных

− разрешенный объем данных

− допуск значений NULL

Где столбец «Тип данных» представляет собой символьный тип данных, столбец «Длина» указывает максимально допустимое количество символов. Для числовых типов данных скобки указывают Precision и Scale. Precision — это общее количество цифр, а Scale — это количество цифр справа от десятичного знака.

Оператор INSERT не обязательно должен указывать каждый столбец — столбцы, допускающие значение NULL, могут быть исключены. Если каждому столбцу, которому требуется значение, присваивается значение, INSERT работает.

Неявная попытка добавить значения в таблицу вызовет ошибку. Пример: INSERT INTO copy\_employees (employee\_id, first\_name, last\_name, phone\_number, hire\_date, job\_id, salary) VALUES (302,'Grigorz','Polanski', '8586667641', '15-Jun-2017', 'IT\_PROG',4200);

Неявная вставка автоматически вставляет нулевое значение в столбцы, допускающие нулевое значение. Чтобы явно добавить нулевое значение в столбец, допускающий нулевое значение, используйте ключевое слово NULL в списке VALUES.

Чтобы указать пустые строки и/или отсутствующие даты, нужно использовать пустые одинарные кавычки для отсутствующих данных. Пример: INSERT INTO copy\_employees (employee\_id, first\_name, last\_name, email, phone\_number, hire\_date, job\_id, salary) VALUES (302,'Grigorz','Polanski', 'gpolanski', '', '15-Jun-2017', 'IT\_PROG',4200);

Специальные значения, такие как SYSDATE и USER, могут быть введены в список VALUES оператора INSERT. SYSDATE поместит текущую дату и время в столбец. USER вставит имя пользователя текущего сеанса, которым является OAE\_PUBLIC\_USER в Oracle Application Express. Пример: INSERT INTO copy\_employees (employee\_id, first\_name, last\_name, email, phone\_number, hire\_date, job\_id, salary) VALUES (304,'Test',USER, 't\_user', 4159982010, SYSDATE, 'ST\_CLERK',2500);

Модель формата по умолчанию для типов данных даты — DDMon-YYYY. В этот формат даты также включено время по умолчанию, равное полуночи (00:00:00). Использовать можно функцию TO\_CHAR для преобразования даты в строку символов, когда мы хотим получить и отобразить значение даты в формате, отличном от используемого по умолчанию. Точно так же, если мы хотим ВСТАВИТЬ строку с форматом, отличным от формата по умолчанию, для столбца даты, мы должны использовать функцию TO\_DATE для преобразования значения даты (строки символов) в дату. Пример: INSERT INTO copy\_employees (employee\_id, first\_name, last\_name, email, phone\_number, hire\_date, job\_id, salary) VALUES (303,'Angelina','Wright', 'awright','4159982010', TO\_DATE('July 10, 2017 17:20', 'Month fmdd, yyyy HH24:MI'), 'MK\_REP', 3600);

Каждый оператор INSERT добавляет в таблицу только одну строку. К счастью, SQL позволяет нам использовать подзапрос внутри инструкции INSERT. Все результаты подзапроса вставляются в таблицу. Таким образом, мы можем скопировать 100 строк — или 1000 строк — с помощью одного многострочного подзапроса в INSERT. Как и следовало ожидать, вам не нужно предложение VALUES при использовании подзапроса для копирования строк, потому что вставленные значения будут точно такими же, как и значения, возвращенные подзапросом. Пример: INSERT INTO sales\_reps(id, name, salary, commission\_pct) SELECT employee\_id, last\_name, salary, commission\_pct FROM employees WHERE job\_id LIKE '%REP%';

Количество столбцов и их типов данных в списке столбцов предложения INSERT должно совпадать с количеством столбцов и их типами данных в подзапросе. Подзапрос не заключен в круглые скобки, как это делается с подзапросами в предложении WHERE оператора SELECT.

Если мы хотим скопировать все данные — все строки и все столбцы — синтаксис еще проще: INSERT INTO sales\_reps SELECT \* FROM employees;

Опять же, это будет работать только в том случае, если обе таблицы имеют одинаковое количество столбцов с соответствующими типами данных и расположены в одном порядке.

**12-2 Updating Column Values and Deleting Rows (Обновление значений столбцов и удаление строк)**

Оператор UPDATE используется для изменения существующих строк в таблице.

UPDATE требует четырех значений:

− название таблицы

− имя столбца (столбцов), значения которого будут изменены

− новое значение для каждого из изменяемых столбцов

− условие, определяющее, какие строки в таблице будут изменены.

Новое значение столбца может быть результатом однострочного подзапроса.

Пример: UPDATE copy\_employees SET phone\_number = '123456' WHERE employee\_id = 303;

Мы можем изменить несколько столбцов и/или несколько строк в одном операторе UPDATE. Пример: UPDATE copy\_employees SET phone\_number = '654321', last\_name = 'Jones' WHERE employee\_id >= 303;

Будьте осторожны при обновлении значений столбцов. Если предложение WHERE опущено, каждая строка в таблице будет обновлена.

Мы можем использовать результат однострочного подзапроса, чтобы предоставить новое значение для обновленного столбца: UPDATE copy\_employees SET salary = (SELECT salary FROM copy\_employees WHERE employee\_id = 100) WHERE employee\_id = 101;

Чтобы обновить несколько столбцов в одном операторе UPDATE, можно написать несколько однострочных подзапросов, по одному для каждого столбца: UPDATE copy\_employees SET salary = (SELECT salary FROM copy\_employees WHERE employee\_id = 205), job\_id = (SELECT job\_id FROM copy\_employees WHERE employee\_id = 205) WHERE employee\_id = 206;

Подзапрос может извлекать информацию из одной таблицы, которая затем используется для обновления другой таблицы. В этом примере была создана копия таблицы сотрудников. Затем данные из исходной таблицы сотрудников были извлечены, скопированы и использованы для заполнения таблицы copy\_employees: UPDATE copy\_employees SET salary = (SELECT salary FROM employees WHERE employee\_id = 205) WHERE employee\_id = 202;

Подзапросы могут быть либо автономными, либо коррелированными. В коррелированном подзапросе вы обновляете строку в таблице на основе выбора из той же таблицы. Пример: ALTER TABLE copy\_employees ADD (department\_name varchar2(30) NOT NULL); UPDATE copy\_employees e SET e.department\_name = (SELECT d.department\_name FROM departments d WHERE e.department\_id = d.department\_id);

Оператор DELETE используется для удаления существующих строк в таблице. Оператор требует два значения:

− название таблицы

− условие, которое идентифицирует удаляемые строки

Пример: DELETE from copy\_employees WHERE employee\_id = 303;

Подзапросы также можно использовать в операторах DELETE. Пример: DELETE FROM copy\_employees WHERE department\_id = (SELECT department\_id FROM departments WHERE department\_name = 'Shipping');

Ограничения целостности гарантируют, что данные соответствуют необходимому набору правил. Ограничения автоматически проверяются всякий раз, когда выполняется оператор DML, который может нарушить правила. Если какое-либо правило нарушается, таблица не обновляется и возвращается ошибка. В этом примере нарушается ограничение NOT NULL, так как last\_name имеет ненулевое ограничение, а id=999 не существует, поэтому подзапрос возвращает нулевой результат: UPDATE copy\_employees SET last\_name = (SELECT last\_name FROM copy\_employees WHERE employee\_id = 999) WHERE employee\_id = 101;

Когда инструкция SELECT выполняется для таблицы базы данных, в базе данных не устанавливаются блокировки для строк, возвращаемых вашим запросом. Большую часть времени мы хотим, чтобы база данных вела себя именно так, чтобы свести количество выдаваемых блокировок к минимуму. Однако иногда мы хотим убедиться, что никто другой не может обновить или удалить записи, которые возвращает наш запрос, пока мы работаем с этими записями. Это когда используется предложение FOR UPDATE. Как только ваш запрос будет выполнен, база данных автоматически установит эксклюзивные блокировки на уровне строк для всех строк, возвращенных вашим оператором SELECT, которые будут удерживаться до тех пор, пока вы не выполните команду COMMIT или ROLLBACK. Если вы используете предложение FOR UPDATE в инструкции SELECT с несколькими таблицами, все строки из всех таблиц будут заблокированы: SELECT e.employee\_id, e.salary, d.department\_name FROM employees e JOIN departments d USING (department\_id) WHERE job\_id = 'ST\_CLERK' AND location\_id = 1500 FOR UPDATE ORDER BY e.employee\_id; Строки теперь заблокированы пользователем, выполнившим оператор SELECT, до тех пор, пока пользователь не выполнит COMMIT или ROLLBACK.

**12-3 DEFAULT Values, MERGE, and Multi-Table Inserts (Значения DEFAULT, MERGE и многотабличные вставки)**

Для каждого столбца в таблице может быть указано значение по умолчанию. Если вставляется новая строка, а значение для столбца не назначается, вместо нулевого значения будет назначено значение по умолчанию. Использование значений по умолчанию позволяет вам контролировать, где и когда следует применять значение по умолчанию. Значением по умолчанию может быть буквальное значение, выражение или функция SQL, такая как SYSDATE и USER, но значение не может быть именем другого столбца. Значение по умолчанию должно соответствовать типу данных столбца. DEFAULT можно указать для столбца при создании или изменении таблицы. Пример: CREATE TABLE my\_employees ( hire\_date DATE DEFAULT SYSDATE, first\_name VARCHAR2(15), last\_name VARCHAR2(15)); Когда в эту таблицу добавляются строки, SYSDATE будет присвоено любой строке, в которой явно не указано значение hire\_date.

Явные значения по умолчанию могут использоваться в операторах INSERT и UPDATE. Пример INSERT с использованием таблицы my\_employees показывает явное использование DEFAULT: INSERT INTO my\_employees (hire\_date, first\_name, last\_name) VALUES (DEFAULT, 'Angelina','Wright');

Неявное использование DEFAULT: INSERT INTO my\_employees (first\_name, last\_name) VALUES ('Angelina','Wright');

Явные значения по умолчанию могут использоваться в операторах INSERT и UPDATE. Пример UPDATE с использованием таблицы my\_employees показывает явное использование DEFAULT: UPDATE my\_employees SET hire\_date = DEFAULT WHERE last\_name = 'Wright'; Если для столбца hire\_date указано значение по умолчанию, столбцу присваивается значение по умолчанию. Однако, если при создании столбца не было указано значение по умолчанию, присваивается нулевое значение.

Использование оператора MERGE выполняет две задачи одновременно. MERGE будет INSERT и UPDATE одновременно. Если значение отсутствует, вставляется новое. Если значение существует, но его нужно изменить, MERGE обновит его. Для внесения подобных изменений в таблицы базы данных вам необходимо иметь привилегии INSERT и UPDATE для целевой таблицы и привилегии SELECT для исходной таблицы. Псевдонимы могут использоваться с оператором MERGE. Одна строка за раз считывается из исходной таблицы и сравнивается со строками в целевой таблице с использованием условия соответствия. Если в целевой таблице существует соответствующая строка, исходная строка используется для обновления одного или нескольких столбцов в соответствующей целевой строке. Если соответствующая строка не существует, значения из исходной строки используются для вставки новой строки в целевую таблицу. Синтаксис: MERGE INTO destination-table USING source-table ON matching-condition WHEN MATCHED THEN UPDATE SET …… WHEN NOT MATCHED THEN INSERT VALUES (……);

Пример: MERGE INTO copy\_emp c USING employees e ON (c.employee\_id = e.employee\_id) WHEN MATCHED THEN UPDATE SET c.last\_name = e.last\_name, c.department\_id = e.department\_id WHEN NOT MATCHED THEN INSERT VALUES (e.employee\_id, e.last\_name, e.department\_id);

Вставки в несколько таблиц используются, когда одни и те же исходные данные должны быть вставлены более чем в одну целевую таблицу. Эта функциональность полезна, когда вы работаете в среде хранилища данных, где обычно регулярно перемещают данные из операционных систем в хранилище данных для аналитической отчетности и анализа. Создание хранилищ данных и управление ими — это один из способов управления иногда очень большим количеством строк, вводимых в операционные системы в течение обычного рабочего дня. Многотабличные вставки могут быть безусловными или условными. При безусловной вставке нескольких таблиц Oracle вставит все строки, возвращенные подзапросом, во все предложения вставки таблицы, найденные в операторе. В условной многотабличной вставке можно указать либо ALL, либо FIRST.

Указание ВСЕХ:

− Если вы укажете ALL, значение по умолчанию, база данных оценивает каждое предложение WHEN независимо от результатов оценки любого другого предложения WHEN.

− Для каждого предложения WHEN, условие которого оценивается как истинное, база данных выполняет соответствующий список предложений INTO.

Указание ПЕРВОГО:

− Если вы укажете FIRST, база данных оценивает каждое предложение WHEN в том порядке, в котором оно появляется в операторе.

− Для первого предложения WHEN, которое оценивается как истинное, база данных выполняет соответствующее предложение INTO и пропускает последующие предложения WHEN для данной строки.

Указание предложения ELSE:

− Для данной строки, если ни одно из предложений WHEN не оценивается как истинное, база данных выполняет список предложений INTO, связанный с предложением ELSE.

− Если вы не указали условие else, база данных не предпринимает никаких действий для этой строки.

Синтаксис оператора вставки нескольких таблиц, следующий: INSERT ALL INTO clause VALUES clause SUBQUERY

Пример оператора вставки нескольких таблиц выглядит следующим образом: INSERT ALL INTO my\_employees VALUES (hire\_date, first\_name, last\_name) INTO copy\_my\_employees VALUES (hire\_date, first\_name, last\_name) SELECT hire\_date, first\_name, last\_name FROM employees;

Другой пример: INSERT ALL WHEN call\_ format IN ('tlk','txt','pic') THEN INTO all\_calls VALUES (caller\_id, call\_timestamp, call\_duration, call\_format) WHEN call\_ format IN ('tlk','txt') THEN INTO police\_record\_calls VALUES (caller\_id, call\_timestamp, recipient\_caller) WHEN call\_duration < 50 AND call\_type = 'tlk' THEN INTO short\_calls VALUES (caller\_id, call\_timestamp, call\_duration) WHEN call\_duration > = 50 AND call\_type = 'tlk' THEN INTO long\_calls VALUES (caller\_id, call\_timestamp, call\_duration) SELECT caller\_id, call\_timestamp, call\_duration, call\_format, recipient\_caller FROM calls WHERE TRUNC(call\_timestamp ) = TRUNC(SYSDATE);

**DDL**

**13-1 Creating Tables (Создание таблиц)**

База данных Oracle может содержать множество различных типов объектов. В этом разделе представлены наиболее часто используемые объекты, а также описано, как сервер Oracle использует информацию, хранящуюся в словаре данных, при выполнении работы в результате введенных вами операторов SQL.

Основные типы объектов базы данных:

−Таблица

−Индекс

−Ограничение

− Просмотр

−Последовательность

−Синоним

Некоторые из этих типов объектов могут существовать независимо, а другие нет.

Некоторые типы объектов занимают место в базе данных, известное как хранилище, а другие — нет. Объекты базы данных, занимающие значительный объем памяти, называются сегментами. Таблицы и индексы являются примерами сегментов, поскольку значения, хранящиеся в столбцах каждой строки, занимают значительное место на физическом диске. Представления, ограничения, последовательности и синонимы также являются объектами, но единственное место, которое им требуется в базе данных, — это определение объекта — ни с одним из них не связаны строки данных.

База данных хранит определения всех объектов базы данных в словаре данных, и эти определения доступны всем пользователям базы данных, а также самой базе данных.

Словарь данных проверяет, существуют ли таблицы, на которые вы ссылаетесь в своем заявлении, в базе данных, проверяет правильность имен столбцов, проверяет, есть ли у вас правильные привилегии для выполнения запрашиваемого вами действия, и, наконец, он использует словарь данных для решения Плана выполнения - как он будет фактически выполнять запрос. Сам словарь данных может запрашиваться всеми пользователями базы данных. В Application Express доступ к нему можно получить с помощью операторов SQL в интерфейсе SQL Workshop > SQL Commands. В окне «Команды SQL нужно знать имена запрашиваемой таблицы, а в интерфейсе «Обозреватель объектов» нужно просто щелкнуть по перечисленным объектам, чтобы увидеть их детали. Используя окно SQL Commands, можно запросить DESCription таблицы. Все дополнительные параметры, предлагаемые Object Browser, недоступны в этом интерфейсе. Пример: DESCRIBE jobs;

Все данные в реляционной базе данных хранятся в таблицах. При создании новой таблицы используйте следующие правила для имен таблиц и имен столбцов:

− Должен начинаться с буквы

− Должно быть от 1 до 30 символов.

−Должен содержать только A–Z, a–z, 0–9, \_ (подчеркивание), $ и #.

− Не должно дублировать имя другого объекта, принадлежащего тому же пользователю

− Не должно быть зарезервированным словом Oracle Server.

Лучше всего использовать описательные имена для таблиц и других объектов базы данных. Если таблица будет хранить информацию о студентах, лучше называть ее STUDENTS, а не PEOPLE или CHILDREN. Имена таблиц не чувствительны к регистру. Имена таблиц должны быть во множественном числе, например STUDENTS, а не student. Создание таблиц является частью языка определения данных SQL (DDL). Другие операторы DDL, используемые для установки, изменения и удаления структур данных из таблиц, включают ALTER, DROP, RENAME и TRUNCATE.

Чтобы создать новую таблицу, вы должны иметь привилегию CREATE TABLE и область хранения для нее. Администратор базы данных использует операторы языка управления данными (DCL), чтобы предоставить эту привилегию пользователям и назначить область хранения. Таблицы, принадлежащие другим пользователям, не входят в вашу схему. Если вы хотите использовать таблицу, которой нет в вашей схеме, используйте имя владельца таблицы в качестве префикса к имени таблицы: SELECT \* FROM mary.students;

Чтобы создать новую таблицу, используйте следующие детали синтаксиса:

−table — имя таблицы

−column — имя столбца

− Тип данных — это тип данных и длина столбца.

- Выражение DEFAULT задает значение по умолчанию, если значение опущено в операторе INSERT.

Синтаксис: CREATE TABLE table (column data type [DEFAULT expression], (column data type [DEFAULT expression], (……[ ] );

Пример: CREATE TABLE my\_cd\_collection (cd\_number NUMBER(3), title VARCHAR2(20), artist VARCHAR2(20), purchase\_date DATE DEFAULT SYSDATE);

Oracle также поддерживает другой тип таблицы: внешняя таблица. Во внешней таблице строки данных не хранятся в файлах базы данных, а находятся в плоском файле, хранящемся вне базы данных. Обычно внешняя таблица используется для хранения данных, перенесенных из старых версий баз данных, используемых компанией. Когда компания внедряет новое приложение и базу данных, ей обычно необходимо импортировать большую часть данных из старых систем в новую систему для нормального доступа для чтения и записи, но могут быть некоторые данные, которые не используются часто и поэтому будут необходимо получить доступ для чтения. Такие данные могут храниться во внешней таблице. Одно из многих преимуществ Oracle заключается в том, что данные, хранящиеся во внешних таблицах, необходимо создать резервную копию только один раз, а затем никогда больше, если содержимое файла не изменится. Синтаксис создания внешней таблицы очень похож на синтаксис создания стандартной таблицы, за исключением того, что в конце есть дополнительный синтаксис.

Пример: CREATE TABLE emp\_load (employee\_number CHAR(5), employee\_dob CHAR(20), employee\_last\_name CHAR(20), employee\_first\_name CHAR(15), employee\_middle\_name CHAR(15), employee\_hire\_date DATE) ORGANIZATION EXTERNAL (TYPE ORACLE\_LOADER DEFAULT DIRECTORY def\_dir1 ACCESS PARAMETERS (RECORDS DELIMITED BY NEWLINE FIELDS (employee\_number CHAR(2), employee\_dob CHAR(20), employee\_last\_name CHAR(18), employee\_first\_name CHAR(11), employee\_middle\_name CHAR(11), employee\_hire\_date CHAR(10) date\_format DATE mask "mm/dd/yyyy")) LOCATION ('info.dat'));

• ORGANIZATION EXTERNAL – указывает Oracle создать внешнюю таблицу.

• TYPE ORACLE\_LOADER – типа Oracle Loader (продукт Oracle)

• DEFAULT DIRECTORY def\_dir1 – имя директории для файла

• ACCESS PARAMETERS – как прочитать файл

• RECORDS DELIMITED BY NEWLINE – как определить начало новой строки

• FIELDS – имя поля и спецификации типа данных.

• LOCATION – имя фактического файла, содержащего данные

В базе данных Oracle существует два типа таблиц: пользовательские таблицы и таблицы словаря данных. Вы можете выполнять операторы SQL для доступа к обоим типам таблиц — вы можете выбирать, вставлять, обновлять и удалять данные в пользовательских таблицах, а также можете выбирать данные в таблицах словаря данных.

Пользовательские таблицы, созданные нами, содержащие наши данные:

− employees, departments, jobs и т. д.

Таблицы словаря данных:

−DICTIONARY, USER\_OBJECTS, USER\_TABLES, USER\_SEGMENTS, USER\_INDEXES и т. д.

Все таблицы словаря данных принадлежат специальному пользователю Oracle с именем SYS, и при работе с любой из этих таблиц следует использовать только операторы SELECT. Чтобы защитить эти таблицы от случайного доступа пользователей, для всех них созданы представления, через которые пользователи базы данных получают доступ к словарю данных. Если какой-либо пользователь Oracle попытается вставить, обновить или удалить какую-либо из таблиц словаря данных, эта операция будет запрещена, так как это может поставить под угрозу целостность всей базы данных.

Когда вы используете представления словаря данных в интерфейсе команд SQL, вам необходимо знать имена представлений словаря, с которыми вы работаете. В Oracle это довольно просто: добавьте к типу объекта, который вы ищете, префикс USER\_xxx или ALL\_xxx, где xxx — тип объекта. Пример: SELECT table\_name, status FROM USER\_TABLES;

Поэтому, если вы хотите исследовать индексы, просто выберите из USER\_INDEXES; если вам нужна информация о последовательностях, то таблица USER\_SEQUENCES и так далее. Пример: SELECT \* FROM user\_indexes; Другой пример: SELECT \* FROM user\_objects WHERE object\_type = 'SEQUENCE';

**13-2 Using Data Types (Использование типов данных)**

Каждое значение, которым управляет Oracle, имеет тип данных. Тип данных значения связывает фиксированный набор свойств со значением. Эти свойства заставляют базу данных обрабатывать значения одного типа данных иначе, чем значения другого типа данных.

Различные типы данных имеют ряд преимуществ:

− Столбцы одного типа дают согласованные результаты.

− Вы не можете вставить неправильный тип данных в столбец.

По этим причинам каждый столбец в реляционной базе данных может содержать данные только одного типа. Нельязя смешивать типы данных в столбце.

Наиболее часто используемые типы данных столбцов для символьных и числовых значений:

Для значений символов:

−CHAR (фиксированный размер, максимум 2000 символов)

−VARCHAR2 (переменный размер, максимум 4000 символов)

−CLOB (переменный размер, максимум 128 терабайт)

Для числовых значений:

−NUMBER (переменный размер, максимальная точность 38 цифр)

Наиболее часто используемые типы данных столбца для даты, времени и двоичных значений:

Для значений даты и времени:

−DATE

−TIMESTAMP ….

−INTERVAL

Для двоичных значений (например, мультимедиа: JPG, WAV, MP3 и т. д.):

−RAW (переменный размер, максимум 2000 байт)

−BLOB (переменный размер, максимум 128 терабайт)

Для символьных значений обычно лучше использовать VARCHAR2 или CLOB, чем CHAR, поскольку это экономит место.

Тип данных DATE хранит значения от столетий до целых секунд, но не может хранить доли секунды.

Тип данных TIMESTAMP является расширением типа данных DATE, которое позволяет использовать доли секунды. Пример: CREATE TABLE time\_ex1 (exact\_time TIMESTAMP); INSERT INTO time\_ex1 VALUES (SYSDATE);

TIMESTAMP WITH TIME ZONE сохраняет значение часового пояса как смещение от универсального скоординированного времени или UCT (ранее известное как среднее время по Гринвичу или GMT). Пример: CREATE TABLE time\_ex2 (time\_with\_offset TIMESTAMP WITH TIME ZONE); INSERT INTO time\_ex2 VALUES ('10-Jun-2017 10:52:29.123456 AM +2:00');

TIMESTAMP WITH LOCAL TIME ZONE аналогичен, но с одним отличием: когда этот столбец выбирается в операторе SQL, время автоматически преобразуется в часовой пояс выбранного пользователя. Пример: CREATE TABLE time\_ex3 ( first\_column TIMESTAMP WITH TIME ZONE, second\_column TIMESTAMP WITH LOCAL TIME ZONE); INSERT INTO time\_ex3 (first\_column, second\_column) VALUES ('15-Jul-2017 08:00:00 AM -07:00', '15-Nov-2007 08:00:00');

INTERVAL хранит прошедшее время или интервал времени между двумя значениями даты и времени.

• INTERVAL YEAR TO MONTH хранит период времени, измеряемый в годах и месяцах.

• INTERVAL DAY TO SECOND сохраняет период времени, измеряемый в днях, часах, минутах и секундах.

Синтаксис: INTERVAL YEAR [(year\_precision)] TO MONTH

year\_precision — максимальное количество цифр в элементе YEAR

Пример INTERVAL YEAR TO MONTH: CREATE TABLE time\_ex4 (loan\_duration1 INTERVAL YEAR(3) TO MONTH, loan\_duration2 INTERVAL YEAR(2) TO MONTH); INSERT INTO time\_ex4 (loan\_duration1, loan\_duration2) VALUES (INTERVAL '120' MONTH(3), INTERVAL '3-6' YEAR TO MONTH);

Используйте INTERVAL DAY…TO SECOND, когда вам нужна более точная разница между двумя значениями даты и времени. Синтаксис: INTERVAL DAY [day\_precision)] TO SECOND [(fractional\_seconds\_precision)]

day\_precision — максимальное количество цифр в элементе DAY

fractional\_seconds\_precision — количество цифр в дробной части поля SECOND даты и времени

Пример: CREATE TABLE time\_ex5 (day\_duration1 INTERVAL DAY(3) TO SECOND, day\_duration2 INTERVAL DAY(3) TO SECOND); INSERT INTO time\_ex5 (day\_duration1, day\_duration2) VALUES (INTERVAL '25' DAY(2), INTERVAL '4 10:30:10' DAY TO SECOND);

**13-3 Modifying a Table (Изменение таблицы)**

Операторы ALTER TABLE используются для:

−Добавить новый столбец

− Изменить существующий столбец

−Определить значение ПО УМОЛЧАНИЮ для столбца

−Удалить столбец

Вы можете добавить или изменить столбец в таблице, но не можете указать, где этот столбец появится. Вновь добавленный столбец всегда становится последним столбцом таблицы. Кроме того, если в таблице уже есть строки данных, и вы добавляете в таблицу новый столбец, новый столбец изначально будет нулевым для всех ранее существовавших строк.

Чтобы добавить новый столбец, используйте показанный синтаксис SQL: ALTER TABLE tablename ADD (column name data type [DEFAULT expression], column name data type [DEFAULT expression], ... Пример: ALTER TABLE my\_cd\_collection ADD (release\_date DATE DEFAULT SYSDATE);

Изменение столбца может включать в себя изменения типа данных столбца, размера и значения по УМОЛЧАНИЮ. Правила и ограничения при изменении столбца:

− Вы можете увеличить ширину или точность числового столбца.

− Вы можете увеличить ширину столбца символов

− Вы можете уменьшить ширину столбца NUMBER, если столбец содержит только нулевые значения или если в таблице нет строк.

− Для типов VARCHAR вы можете уменьшить ширину до наибольшего значения, содержащегося в столбце.

Вы можете изменить тип данных, только если столбец содержит нулевые значения. Вы можете преобразовать столбец CHAR в VARCHAR2 или преобразовать столбец VARCHAR2 в CHAR, только если столбец содержит нулевые значения или если вы не измените размер на что-то меньшее, чем любое значение в столбце. Изменение значения DEFAULT столбца влияет только на последующие вставки в таблицу. Пример: CREATE TABLE mod\_emp (last\_name VARCHAR2(20), salary NUMBER(8,2)); ALTER TABLE mod\_emp MODIFY (last\_name VARCHAR2(30));

При удалении столбца применяются следующие правила:

− Столбец, содержащий данные, может быть удален

− За один раз можно удалить только один столбец.

− Вы не можете удалить все столбцы в таблице; должен остаться хотя бы один столбец

− После удаления столбца значения данных в нем не могут быть восстановлены.

Синтаксис: ALTER TABLE tablename DROP COLUMN column name; Пример: ALTER TABLE my\_cd\_collection DROP COLUMN release\_date;

Удаление столбца из большой таблицы может занять много времени. Более быстрая альтернатива — пометить столбец как непригодный для использования. Значения столбца остаются в базе данных, но к ним нельзя получить доступ, поэтому эффект такой же, как и при удалении столбца. Фактически, вы можете добавить в базу данных новый столбец с тем же именем, что и неиспользуемый столбец. Неиспользуемые столбцы есть, но невидимы! Синтаксис: ALTER TABLE tablename SET UNUSED (column name); Пример: ALTER TABLE copy\_employees SET UNUSED (email);

DROP UNUSED COLUMNS удаляет все столбцы, помеченные в данный момент как неиспользуемые. Вы используете этот оператор, когда хотите освободить дополнительное дисковое пространство от неиспользуемых столбцов в таблице. Пример: ALTER TABLE copy\_employees DROP UNUSED COLUMNS;

Оператор DROP TABLE удаляет определение таблицы Oracle. База данных теряет все данные в таблице и все связанные с ней индексы. Когда выдается оператор DROP TABLE:

−Все данные удаляются из таблицы

−Описание таблицы удалено из словаря данных.

Сервер Oracle не подвергает сомнению ваше решение и немедленно отбрасывает таблицу. Вы увидите, что вы можете восстановить таблицу после ее удаления, но это не гарантируется. Только создатель таблицы или пользователь с привилегией DROP ANY TABLE (обычно только администратор базы данных) может удалить таблицу. Синтаксис: DROP TABLE tablename; Пример: DROP TABLE copy\_employees;

Если вы удалили таблицу по ошибке, вы можете вернуть эту таблицу и ее данные. У каждого пользователя базы данных есть своя корзина, в которую перемещаются удаленные объекты, откуда их можно восстановить с помощью команды FLASHBACK TABLE. Эту команду можно использовать для восстановления таблицы, представления или индекса, которые были удалены по ошибке. Синтаксис: FLASHBACK TABLE tablename TO BEFORE DROP; Пример: FLASHBACK TABLE copy\_employees TO BEFORE DROP;

Можно увидеть, какие объекты можно восстановить, запросив представление словаря данных USER\_RECYCLEBIN. Представление USER\_RECYCLEBIN можно запрашивать, как и все другие представления словаря данных: SELECT original\_name, operation, droptime FROM user\_recyclebin После восстановления таблицы с помощью команды FLASHBACK TABLE она больше не отображается в представлении USER\_RECYCLEBIN. Все индексы, которые были удалены вместе с исходной таблицей, также будут восстановлены. Может быть необходимо (из соображений безопасности) полностью удалить таблицу, минуя корзину. Это можно сделать, добавив ключевое слово PURGE: DROP TABLE copy\_employees PURGE;

Чтобы изменить имя таблицы, используйте оператор RENAME. Это может сделать только владелец объекта или администратор базы данных. Синтаксис: RENAME old\_name to new\_name; Пример: RENAME my\_cd\_collection TO my\_music;

Усечение таблицы удаляет все строки из таблицы и освобождает место для хранения, используемое этой таблицей. При использовании оператора TRUNCATE TABLE:

− Вы не можете отменить удаление строки

− Вы должны быть владельцем таблицы или иметь системные привилегии DROP ANY TABLE.

Синтаксис: TRUNCATE TABLE tablename;

Оператор DELETE также удаляет строки из таблицы, но не освобождает место для хранения. TRUNCATE работает быстрее, чем DELETE, поскольку не генерирует информацию об откате.

Вы можете добавить комментарий длиной до 2000 символов о столбце, таблице или представлении с помощью оператора COMMENT. Синтаксис: COMMENT ON TABLE tablename | COLUMN table.column IS 'place your comment here'; Пример: COMMENT ON TABLE employees IS 'Western Region only';

Если вы хотите удалить ранее сделанный комментарий к таблице или столбцу, используйте пустую строку (''): COMMENT ON TABLE employees IS ' ' ;

Вы можете обнаружить, что данные в таблице каким-то образом были изменены ненадлежащим образом. К счастью, в Oracle есть средство, позволяющее просматривать данные строк в определенные моменты времени, так что вы можете сравнивать различные версии строки во времени. Вы можете использовать средство FLASHBACK QUERY, чтобы проверить, как выглядели строки ДО того, как эти изменения были применены. Когда Oracle изменяет данные, он всегда сохраняет копию измененных данных до внесения каких-либо изменений. Таким образом, он сохраняет копию старого значения столбца для обновления столбца, сохраняет всю строку для удаления и ничего не сохраняет для оператора вставки. Эти старые копии хранятся в специальном месте, называемом табличным пространством UNDO. Пользователи могут получить доступ к этой специальной области базы данных с помощью ретроспективного запроса. Вы можете просмотреть более старые версии данных, используя предложение VERSIONS в операторе SELECT. Пример: SELECT employee\_id,first\_name ||' '|| last\_name AS "NAME", versions\_operation AS "OPERATION", versions\_starttime AS "START\_DATE", versions\_endtime AS "END\_DATE", salary FROM employees VERSIONS BETWEEN SCN MINVALUE AND MAXVALUE WHERE employee\_id = 1;

**Constraints**

**14-1 Intro to Constraints; NOT NULL and UNIQUE Constraints (Введение в ограничения; Ограничения NOT NULL и UNIQUE)**

Все определения ограничений хранятся в словаре данных. Ограничения предотвращают удаление таблицы, если есть зависимости от других таблиц. Ограничения применяют правила к данным всякий раз, когда строка вставляется, обновляется или удаляется из таблицы. Ограничения важны, и их имена также важны.

Используют оператор CREATE TABLE, чтобы установить ограничения для каждого столбца в таблице. В операторе CREATE TABLE есть два разных места, где вы можете указать детали ограничения:

− На уровне столбца рядом с именем и типом данных

− На уровне таблицы после того, как будут перечислены все имена столбцов.

Пример: CREATE TABLE clients (client\_number NUMBER(4), first\_name VARCHAR2(14), last\_name VARCHAR2(13)); Уровень столбца просто относится к области в операторе CREATE TABLE, где определены столбцы. Уровень таблицы относится к последней строке оператора под списком имен отдельных столбцов.

Ограничение уровня столбца ссылается на один столбец. Чтобы установить ограничение на уровне столбца, это ограничение должно быть определено в операторе CREATE TABLE как часть определения столбца. Пример: CREATE TABLE clients (client\_number NUMBER(4) CONSTRAINT clients\_client\_num\_pk PRIMARY KEY, first\_name VARCHAR2(14), last\_name VARCHAR2(13)); Имя ограничения client\_client\_num\_pk. Он обеспечивает соблюдение бизнес-правила, согласно которому client\_number является первичным ключом таблицы клиентов.

Каждое ограничение в базе данных имеет имя. Когда ограничение создано, ему можно дать имя. Соглашение об именовании может быть комбинацией сокращенного имени таблицы и сокращенного имени столбца, за которым следует сокращение ограничения: имя таблицы\_имя-столбца\_тип-ограничения. Если в определении CREATE TABLE используется зарезервированное слово CONSTRAINT, должны дать ограничению имя. Имена ограничений ограничены 30 символами.

Лучше всего присваивать имена ограничениям самостоятельно, потому что имена, сгенерированные системой, нелегко распознать. Пример: CREATE TABLE clients (client\_number NUMBER(4), last\_name VARCHAR2(13), email VARCHAR2(80)); Согласно нашему соглашению об именах:

− Ограничение первичного ключа для client\_number будет называться client\_client\_number\_pk.

− Ненулевое ограничение для last\_name будет называться client\_last\_name\_nn.

− Уникальное ограничение на адрес электронной почты будет называться client\_email\_uk.

CREATE TABLE clients (client\_number NUMBER(4) CONSTRAINT clients\_cient\_num\_pk PRIMARY KEY, last\_name VARCHAR2(13) CONSTRAINT clients\_last\_name\_nn NOT NULL, email VARCHAR2(80) CONSTRAINT clients\_emil\_uk UNIQUE);

В этом примере показано как ограничение с именем пользователя, так и ограничение с именем системы: CREATE TABLE clients (client\_number NUMBER(4) CONSTRAINT clients\_client\_num\_pk PRIMARY KEY, last\_name VARCHAR2(13) NOT NULL, email VARCHAR2(80));

Ограничения на уровне таблицы перечислены отдельно от определений столбцов в операторе CREATE TABLE. Определения ограничений на уровне таблицы перечислены после определения всех столбцов таблицы. В показанном примере ограничение уникальности указано последним в операторе CREATE TABLE: CREATE TABLE clients ( client\_number NUMBER(6) NOT NULL, first\_name VARCHAR2(20), last\_name VARCHAR2(20), phone VARCHAR2(20), email VARCHAR2(10) NOT NULL, CONSTRAINT clients\_phone\_email\_uk UNIQUE (email,phone));

Ограничения, относящиеся к более чем одному столбцу (составной ключ), должны быть определены на уровне таблицы. Ограничение NOT NULL можно указать только на уровне столбца, а не на уровне таблицы. Ограничения UNIQUE, PRIMARY KEY, FOREIGN KEY и CHECK могут быть определены либо на уровне столбца, либо на уровне таблицы. Если слово CONSTRAINT используется в операторе CREATE TABLE, вы должны дать ограничению имя.

Пример: CREATE TABLE clients( client\_number NUMBER(6), first\_name VARCHAR2(20), last\_name VARCHAR2(20), phone VARCHAR2(20) CONSTRAINT phone\_email\_uk UNIQUE(email,phone), email VARCHAR2(10) CONSTRAINT NOT NULL, CONSTRAINT emailclients\_email NOT NULL, CONSTRAINT clients\_client\_num\_pk PRIMARY KEY (client\_number));

В базе данных Oracle существует пять типов ограничений. Каждый тип применяет различные правила. Типы:

− NOT NULL ограничения

− UNIQUE ограничения

− ограничения PRIMARY KEY

− ограничения FOREIGN KEY

− CHECK ограничения

Столбец, определенный с ограничением NOT NULL, требует, чтобы для каждой строки, введенной в таблицу, существовало значение для этого столбца. При определении столбцов NOT NULL обычно используется suffix\_nn в имени ограничения.

Ограничение UNIQUE требует, чтобы каждое значение в столбце или наборе столбцов (составной ключ) было уникальным; то есть никакие две строки таблицы не могут иметь повторяющихся значений. Столбец или набор столбцов, определенный как UNIQUE, называется уникальным ключом. Если комбинация двух или более столбцов должна быть уникальной для каждой записи, ограничение называется составным уникальным ключом. Заявление о том, что все комбинации адреса электронной почты и фамилии должны быть УНИКАЛЬНЫМИ, является примером составного уникального ключа. Слово «ключ» относится к столбцам, а не к именам ограничений. При определении ограничений UNIQUE принято использовать suffix\_uk в имени ограничения. Чтобы определить составной уникальный ключ, необходимо определить ограничение на уровне таблицы, а не на уровне столбца. Пример составного имени ограничения уникального ключа: CONSTRAINT clients\_phone\_email\_uk UNIQUE(email,phone)

Ограничения UNIQUE позволяют вводить пустые значения, если для столбца также не определено ограничение NOT NULL. Значение NULL в столбце (или во всех столбцах составного уникального ключа) всегда удовлетворяет ограничению UNIQUE, поскольку значения NULL не считаются равными чему-либо. Чтобы удовлетворить ограничение, определяющее составной уникальный ключ, никакие две строки в таблице не могут иметь одинаковую комбинацию значений в ключевых столбцах. Кроме того, любая строка, содержащая значения NULL во всех ключевых столбцах, автоматически удовлетворяет этому ограничению.

Когда вы добавляете ограничение NOT NULL как часть оператора создания таблицы, база данных Oracle создает в базе данных Check Constraint, чтобы обеспечить соблюдение значения в столбце NOT NULL. Это создание ограничения может быть почти невидимым для вас, когда вы создаете свою таблицу — Oracle просто делает это. В конце вашего оператора создания таблицы отображается сообщение «Таблица создана», но не предоставляется никаких подробностей о количестве или типах ограничений, которые также были созданы.

**14-2 PRIMARY KEY, FOREIGN KEY, and CHECK Constraints (PRIMARY KEY, FOREIGN KEY и ограничения CHECK)**

Ограничение PRIMARY KEY — это правило, согласно которому значения в одном столбце или комбинации столбцов должны однозначно идентифицировать каждую строку в таблице. Ни одно значение первичного ключа не может появляться более чем в одной строке таблицы. Чтобы удовлетворить ограничение PRIMARY KEY, оба следующих условия должны быть истинными:

−Ни один столбец, являющийся частью первичного ключа, не может содержать значение NULL.

− Таблица может иметь только один первичный ключ.

Ограничения PRIMARY KEY могут быть определены на уровне столбца или таблицы. Однако если создается составной PRIMARY KEY, он должен быть определен на уровне таблицы. При определении столбцов PRIMARY KEY рекомендуется использовать suffix\_pk в имени ограничения. В операторе CREATE TABLE указывается синтаксис ограничения PRIMARY KEY уровня столбца: CREATE TABLE clients (client\_number NUMBER(4) CONSTRAINT clients\_client\_num\_pk PRIMARY KEY, first\_name VARCHAR2(14), last\_name VARCHAR2(13)); Уровень столбца просто относится к области в операторе CREATE TABLE, где определены столбцы. Уровень таблицы относится к последней строке оператора под списком имен отдельных столбцов. Для создания ограничения PRIMARY KEY на уровне таблицы используется следующий синтаксис: CREATE TABLE clients (client\_number NUMBER(4), first\_name VARCHAR2(14), last\_name VARCHAR2(13), CONSTRAINT clients\_client\_num\_pk PRIMARY KEY (client\_number)); Имя столбца PRIMARY KEY следует за типом ограничения и заключено в круглые скобки. Чтобы определить составной PRIMARY KEY, вы должны определить ограничение на уровне таблицы, а не на уровне столбца. Пример составного ограничения первичного ключа: CREATE TABLE copy\_job\_history (employee\_id NUMBER(6,0), start\_date DATE, job\_id VARCHAR2(10), department\_id NUMBER(4,0), CONSTRAINT copy\_jhist\_id\_st\_date\_pk PRIMARY KEY(employee\_id, start\_date));

Ограничения FOREIGN KEY также называются ограничениями «ссылочной целостности». Ограничения внешнего ключа определяют столбец или комбинацию столбцов как внешний ключ. Внешние ключи связаны с первичным ключом (или уникальным ключом) в другой таблице, и эта ссылка является основой отношений между таблицами.

Таблица, содержащая внешний ключ, называется «дочерней» таблицей, а таблица, содержащая указанный ключ, называется «родительской» таблицей.

Чтобы удовлетворить ограничение ссылочной целостности, значение внешнего ключа должно совпадать с существующим значением в родительской таблице или иметь значение NULL. Значение первичного ключа может существовать без соответствующего значения внешнего ключа; однако внешний ключ должен иметь соответствующий первичный ключ.

Правило таково: прежде чем вы определите ограничение ссылочной целостности в дочерней таблице, ограничение UNIQUE или PRIMARY KEY в родительской таблице уже должно быть определено. Другими словами, вы должны сначала определить родительский первичный ключ, прежде чем вы сможете создать внешний ключ в дочерней таблице.

Для определения ограничения FOREIGN KEY рекомендуется использовать suffix\_fk в имени ограничения.

Синтаксис для определения ограничения FOREIGN KEY требует ссылки на таблицу и столбец в родительской таблице. Ограничение FOREIGN KEY в операторе CREATE TABLE можно определить следующим образом. Пример синтаксиса на уровне столбца: CREATE TABLE copy\_employees (employee\_id NUMBER(6,0) CONSTRAINT copy\_emp\_pk PRIMARY KEY, first\_name VARCHAR2(20), last\_name VARCHAR2(25), department\_id NUMBER(4,0) CONSTRAINT c\_emps\_dept\_id\_fk REFERENCES departments(department\_id), email VARCHAR2(25)); Пример синтаксиса на уровне таблицы: CREATE TABLE copy\_employees (employee\_id NUMBER(6,0) CONSTRAINT copy\_emp\_pk PRIMARY KEY, first\_name VARCHAR2(20), last\_name VARCHAR2(25), department\_id NUMBER(4,0), email VARCHAR2(25), CONSTRAINT c\_emps\_dept\_id\_fk FOREIGN KEY (department\_id) REFERENCES departments(department\_id));

Использование параметра ON DELETE CASCADE при определении внешнего ключа позволяет удалять зависимые строки в дочерней таблице при удалении строки в родительской таблице. Если внешний ключ не имеет параметра ON DELETE CASCADE, строки, на которые есть ссылки в родительской таблице, не могут быть удалены. Другими словами, ограничение FOREIGN KEY дочерней таблицы включает разрешение ON DELETE CASCADE, позволяющее родительской таблице удалять строки, на которые она ссылается.

Если столбец department\_id в файле employee был создан с параметром ON DELETE CASCADE, будет выполнен оператор DELETE, выданный для таблицы отделов. Если параметр ON DELETE CASCADE не был указан при создании ВНЕШНЕГО КЛЮЧА, попытка удалить из таблицы отделов отдел, имеющий записи в таблице сотрудников, не удастся. Таблица создана без ON DELETE CASCADE: CREATE TABLE copy\_employees (employee\_id NUMBER(6,0) CONSTRAINT copy\_emp\_pk PRIMARY KEY, first\_name VARCHAR2(20), last\_name VARCHAR2(25), department\_id NUMBER(4,0), email VARCHAR2(25), CONSTRAINT cdept\_dept\_id\_fk FOREIGN KEY (department\_id) REFERENCES copy\_departments(department\_id));

Таблица, созданная с помощью ON DELETE CASCADE: CREATE TABLE copy\_employees (employee\_id NUMBER(6,0) CONSTRAINT copy\_emp\_pk PRIMARY KEY, first\_name VARCHAR2(20), last\_name VARCHAR2(25), department\_id NUMBER(4,0), email VARCHAR2(25), CONSTRAINT cdept\_dept\_id\_fk FOREIGN KEY (department\_id) REFERENCES copy\_departments(department\_id) ON DELETE CASCADE);

Вместо того, чтобы удалять строки в дочерней таблице при использовании параметра ON DELETE CASCADE, дочерние строки могут быть заполнены нулевыми значениями с помощью параметра ON DELETE SET NULL: CREATE TABLE copy\_employees (employee\_id NUMBER(6,0) CONSTRAINT copy\_emp\_pk PRIMARY KEY, first\_name VARCHAR2(20), last\_name VARCHAR2(25), department\_id NUMBER(4,0), email VARCHAR2(25), CONSTRAINT cdept\_dept\_id\_fk FOREIGN KEY (department\_id) REFERENCES copy\_departments(department\_id) ON DELETE SET NULL); Это может быть полезно, когда значение родительской таблицы изменяется на новое число. Вы не хотели бы удалять строки в дочерней таблице. Когда новые номера штрих-кодов вводятся в родительскую таблицу, их можно будет вставить в дочернюю таблицу без необходимости полностью воссоздавать каждую строку дочерней таблицы.

Ограничение CHECK явно определяет условие, которое должно быть выполнено. Чтобы удовлетворить ограничение, каждая строка в таблице должна сделать условие либо истинным, либо неизвестным (из-за нулевого значения). Условие ограничения CHECK может относиться к любому столбцу в указанной таблице, но не к столбцам других таблиц. Пример: CREATE TABLE copy\_job\_history (employee\_id NUMBER(6,0), start\_date DATE, end\_date DATE, job\_id VARCHAR2(10), department\_id NUMBER(4,0), CONSTRAINT cjhist\_emp\_id\_st\_date\_pk PRIMARY KEY(employee\_id, start\_date), CONSTRAINT cjhist\_end\_ck CHECK (end\_date > start\_date));

Ограничение CHECK должно быть только в той строке, где оно определено. Ограничение CHECK нельзя использовать в запросах, ссылающихся на значения в других строках. Ограничение CHECK не может содержать вызовы функций SYSDATE, UID, USER или USERENV. Ограничение CHECK не может использовать псевдостолбцы CURRVAL, NEXTVAL, LEVEL или ROWNUM. Один столбец может иметь несколько ограничений CHECK, которые ссылаются на столбец в его определении. Количество ограничений CHECK, которые можно определить для столбца, не ограничено.

Ограничения CHECK могут быть определены на уровне столбца или таблицы. Синтаксис для определения ограничения CHECK:

− Синтаксис уровня столбца: salary NUMBER(8,2) CONSTRAINT employees\_min\_sal\_ck CHECK (salary > 0)

− Синтаксис на уровне таблицы: CONSTRAINT employees\_min\_sal\_ck CHECK (salary > 0)

**14-3 Managing Constraints (Управление ограничениями)**

Оператор ALTER TABLE используется для внесения изменений в ограничения в существующих таблицах. Эти изменения могут включать добавление или удаление ограничений, включение или отключение ограничений и добавление ограничения NOT NULL к столбцу. Рекомендации по внесению изменений в ограничения:

− Вы можете добавлять, удалять, включать или отключать ограничение, но не можете изменять его структуру.

− Вы можете добавить ограничение NOT NULL к существующему столбцу, используя предложение MODIFY оператора ALTER TABLE.

−MODIFY используется, поскольку NOT NULL является изменением на уровне столбца.

− Вы можете определить ограничение NOT NULL, только если таблица пуста или если столбец содержит значение для каждой строки.

Оператор ALTER требует:

−имя таблицы

−имя ограничения

−тип ограничения

−имя столбца, затронутого ограничением

В показанном ниже примере кода с использованием таблицы сотрудников ограничение первичного ключа можно было добавить после того, как таблица была изначально создана: ALTER TABLE employees ADD CONSTRAINT emp\_id\_pk PRIMARY KEY (employee\_id);

Чтобы добавить ограничение в существующую таблицу, используйте следующий синтаксис SQL: ALTER TABLE table\_name ADD [CONSTRAINT constraint\_name] type of constraint (column\_name);

Если ограничение является ограничением FOREIGN KEY, ключевое слово REFERENCES должно быть включено в инструкцию. Синтаксис: ALTER TABLE tablename ADD CONSTRAINT constraint\_name FOREIGN KEY(column\_name) REFERENCES tablename(column\_name);

Пример: ALTER TABLE employees ADD CONSTRAINT emp\_dept\_fk FOREIGN KEY (department\_id) REFERENCES departments (department\_id) ON DELETE CASCADE;

Если ограничение является ограничением NOT NULL, инструкция ALTER TABLE использует MODIFY вместо ADD. Ограничения NOT NULL можно добавлять, только если таблица пуста или столбец содержит значение для каждой строки. Синтаксис: ALTER TABLE table\_name MODIFY (column\_name CONSTRAINT constraint\_name NOT NULL); Пример: ALTER TABLE employees MODIFY (email CONSTRAINT emp\_email\_nn NOT NULL);

Чтобы обеспечить соблюдение правил, определенных ограничениями целостности, ограничения всегда должны быть включены. Однако в некоторых ситуациях желательно временно отключить ограничения целостности таблицы из соображений производительности, например:

−При загрузке больших объемов данных в таблицу

− При выполнении пакетных операций, которые вносят массовые изменения в таблицу.

Чтобы удалить ограничение, вам нужно знать имя ограничения. Если вы его не знаете, вы можете найти имя ограничения в USER\_CONSTRAINTS и USER\_CONS\_COLUMNS в словаре данных. Параметр CASCADE в предложении DROP приводит к тому, что любые зависимые ограничения также будут удалены. Когда вы отбрасываете ограничение целостности, это ограничение больше не применяется сервером Oracle и больше не доступно в словаре данных. Никакие строки или какие-либо данные в любой из затронутых таблиц не удаляются при удалении ограничения.

Синтаксис: ALTER TABLE table\_name DROP CONSTRAINT name [CASCADE} Пример: ALTER TABLE copy\_departments DROP CONSTRAINT c\_dept\_dept\_id\_pk CASCADE;

По умолчанию, всякий раз, когда в операторе CREATE или ALTER TABLE определено ограничение целостности, оно автоматически включается (применяется) Oracle, если оно специально не создано в отключенном состоянии с помощью предложения DISABLE. Вы можете отключить ограничение, не удаляя его и не создавая заново, с помощью параметра ALTER TABLE DISABLE. DISABLE разрешает входящие данные независимо от того, соответствуют ли они ограничениям. Эта функция позволяет добавлять данные в дочернюю таблицу без соответствующих значений в родительской таблице. DISABLE просто отключает ограничение. Можно использовать предложение DISABLE как в операторе ALTER TABLE, так и в операторе CREATE TABLE: CREATE TABLE copy\_employees ( employee\_id NUMBER(6,0) PRIMARY KEY DISABLE, ... ...); ALTER TABLE copy\_employees DISABLE CONSTRAINT c\_emp\_dept\_id\_fk; Отключение ограничения уникальности или первичного ключа удаляет уникальный индекс.

Предложение CASCADE отключает зависимые ограничения целостности. Если ограничение позже будет включено, зависимые ограничения не будут включены автоматически. Синтаксис: ALTER TABLE table\_name DISABLE CONSTRAINT constraint\_name [CASCADE]; Пример: ALTER TABLE copy\_departments DISABLE CONSTRAINT c\_dept\_dept\_id\_pk CASCADE;

Чтобы активировать отключенное в настоящее время ограничение целостности, используйте предложение ENABLE в операторе ALTER TABLE. ENABLE гарантирует, что все входящие данные соответствуют ограничениям. Синтаксис: ALTER TABLE table\_name ENABLE CONSTRAINT constraint\_name; Пример: ALTER TABLE copy\_departments ENABLE CONSTRAINT c\_dept\_dept\_id\_pk;

Если вы включите ограничение, это ограничение будет применяться ко всем данным в таблице. Все данные в таблице должны соответствовать ограничению. Если вы включаете ограничение UNIQUE KEY или PRIMARY KEY, автоматически создается индекс UNIQUE или PRIMARY KEY. Включение ограничения PRIMARY KEY, которое было отключено с помощью параметра CASCADE, не включает внешние ключи, зависящие от первичного ключа. ENABLE снова включает ограничение после того, как вы его отключили.

Каскадные ограничения ссылочной целостности позволяют определить действия, предпринимаемые сервером базы данных, когда пользователь пытается удалить или обновить ключ, на который указывают существующие внешние ключи. Предложение CASCADE CONSTRAINTS используется вместе с предложением DROP COLUMN. Он отбрасывает все ограничения ссылочной целостности, которые относятся к первичным и уникальным ключам, определенным в отброшенных столбцах. Он также отбрасывает все многостолбцовые ограничения, определенные для отброшенных столбцов.

Если оператор ALTER TABLE не включает параметр CASCADE CONSTRAINTS, любая попытка удалить ограничение первичного ключа или многостолбца завершится неудачно. Мы не можем удалить родительское значение, если дочерние значения существуют в других таблицах. Синтаксис: ALTER TABLE table\_name DROP(column name(s)) CASCADE CONSTRAINTS;

Если все столбцы, на которые ссылаются ограничения, определенные для удаленных столбцов, также удаляются, то CASCADE CONSTRAINTS не требуется. Синтаксис: ALTER TABLE tablename DROP (pk\_column\_name(s)); Однако, если на какое-либо ограничение ссылаются столбцы из других таблиц или оставшиеся столбцы в целевой таблице, вы должны указать CASCADE CONSTRAINTS, чтобы избежать ошибки.

После создания таблицы вы можете подтвердить ее существование, введя команду DESCRIBE. Единственное ограничение, которое можно проверить с помощью DESCRIBE, — это ограничение NOT NULL. Ограничение NOT NULL также появится в словаре данных как ограничение CHECK.

Чтобы просмотреть все ограничения для вашей таблицы, запросите таблицу USER\_CONSTRAINTS: SELECT constraint\_name, table\_name, constraint\_type, status FROM USER\_CONSTRAINTS WHERE table\_name ='COPY\_EMPLOYEES';

Типы ограничений, перечисленные в Словаре данных:

−P – PRIMARY KEY;

−R – REFERENCES (внешний ключ);

−C – ограничение CHECK (включая NOT NULL);

−U – UNIQUE

**Views**

**15-1 Creating Views (Создание представлений)**

Представление, как и таблица, является объектом базы данных. Однако представления не являются «настоящими» таблицами. Они являются логическим представлением существующих таблиц или другого представления. Представления не содержат собственных данных. Они функционируют как окно, через которое можно просмотреть или изменить данные из таблиц. Таблицы, на которых основано представление, называются «базовыми». Представление — это запрос, хранящийся как оператор SELECT в словаре данных. Пример: CREATE VIEW view\_employees AS SELECT employee\_id,first\_name, last\_name, email FROM employees WHERE employee\_id BETWEEN 100 and 124;

Представления ограничивают доступ к данным базовой таблицы, поскольку представление может отображать выбранные столбцы из таблицы. Представления можно использовать для уменьшения сложности выполнения запросов на основе более сложных операторов SELECT. Пользователь представления не видит ни базового кода, ни того, как его создать. Пользователь через представление взаимодействует с базой данных с помощью простых запросов. Представления можно использовать для извлечения данных из нескольких таблиц, обеспечивая независимость данных для пользователей. Пользователи могут просматривать одни и те же данные по-разному. Представления предоставляют группам пользователей доступ к данным в соответствии с их конкретными разрешениями или критериями.

Чтобы создать представление, вставьте подзапрос в инструкцию CREATE VIEW. Синтаксис: CREATE [OR REPLACE] [FORCE| NOFORCE] VIEW view [(alias [, alias]...)] AS subquery [WITH CHECK OPTION [CONSTRAINT constraint]] [WITH READ ONLY [CONSTRAINT constraint]];

OR REPLACE — повторно создает представление, если оно уже существует.

FORCE — создает представление независимо от того, существуют ли базовые таблицы.

NOFORCE — создает представление, только если базовая таблица существует (по умолчанию).

view\_name — указывает имя представления.

alias — указывает имя для каждого выражения, выбранного запросом представления.

subquery — это полный оператор SELECT. Вы можете использовать псевдонимы для столбцов в списке SELECT. Подзапрос может содержать сложный синтаксис SELECT.

WITH CHECK OPTION — указывает, что строки остаются доступными для представления после операций вставки или обновления.

CONSTRAINT — это имя, присвоенное ограничению CHECK OPTION.

WITH READ ONLY — гарантирует, что никакие операции DML не могут выполняться в этом представлении.

Пример: CREATE OR REPLACE VIEW view\_euro\_countries AS SELECT country\_id, region\_id, country\_name, capitol FROM wf\_countries WHERE location LIKE '%Europe';

Подзапрос, определяющий представление, может содержать сложный синтаксис SELECT. Из соображений производительности подзапрос, определяющий представление, не должен содержать предложение ORDER BY. Предложение ORDER BY лучше всего указывать при извлечении данных из представления. Вы можете использовать параметр OR REPLACE, чтобы изменить определение представления без необходимости его удаления или повторного предоставления объектных привилегий, ранее предоставленных ему. Псевдонимы могут использоваться для имен столбцов в подзапросе. Используются две классификации видов: простые и сложные.

|  |  |  |
| --- | --- | --- |
| **Особенность** | **Простые представления** | **Сложные представления** |
| **Количество таблиц, используемых для получения данных** | Один | Один или больше |
| **Может содержать функции** | Нет | Да |
| **Может содержать группы данных** | Нет | Да |
| **Может выполнять операции DML (INSERT, UPDATE, DELETE) через представление** | Да | Не всегда |

Пример простого: CREATE OR REPLACE VIEW view\_euro\_countries AS SELECT country\_id, country\_name, capitol FROM wf\_countries WHERE location LIKE '%Europe'; Подзапрос извлекает данные только из одной таблицы и не содержит функции объединения или каких-либо групповых функций. Поскольку это простое представление, операции INSERT, UPDATE, DELETE и MERGE, влияющие на базовую таблицу, могут выполняться через представление.

Имена столбцов в операторе SELECT могут иметь псевдонимы. Псевдонимы также могут быть перечислены после оператора CREATE VIEW и перед подзапросом SELECT. Пример: CREATE OR REPLACE VIEW view\_euro\_countries AS SELECT country\_id AS "ID", country\_name AS "Country", capitol AS "Capitol City" FROM wf\_countries WHERE location LIKE '%Europe';

Можно создать представление независимо от того, существуют ли базовые таблицы. Добавление слова FORCE к оператору CREATE VIEW создает представление. Администратору баз данных эта опция может быть полезна при разработке базы данных, особенно если вы ожидаете скорого предоставления необходимых привилегий для объекта, на который указывает ссылка. Опция FORCE создаст представление несмотря на то, что оно недействительно. Параметр NOFORCE используется по умолчанию при создании представления.

Сложные представления — это представления, которые могут содержать групповые функции и объединения. Пример: CREATE OR REPLACE VIEW view\_euro\_countries ("ID", "Country", "Capitol City", "Region") AS SELECT c.country\_id, c.country\_name, c.capitol, r.region\_name FROM wf\_countries c JOIN wf\_world\_regions r USING (region\_id) WHERE location LIKE '%Europe';

Групповые функции также могут быть добавлены к операторам сложного представления. Пример: CREATE OR REPLACE VIEW view\_high\_pop ("Region ID", "Highest population") AS SELECT region\_id, MAX(population) FROM wf\_countries GROUP BY region\_id;

Чтобы изменить существующее представление, не удаляя его и не создавая заново, используйте параметр OR REPLACE в операторе CREATE VIEW. Старый вид заменяется новой версией. Пример: CREATE OR REPLACE VIEW view\_euro\_countries AS SELECT country\_id, region\_id, country\_name, capitol FROM wf\_countries WHERE location LIKE '%Europe';

**15-2 DML Operations and Views (Операции и представления DML)**

Операции DML INSERT, UPDATE и DELETE можно выполнять с простыми представлениями. Эти операции можно использовать для изменения данных в базовых базовых таблицах. Если вы создаете представление, которое позволяет пользователям просматривать ограниченную информацию с помощью предложения WHERE, пользователи по-прежнему могут выполнять операции DML для всех столбцов представления. Пример: CREATE VIEW view\_dept50 AS SELECT department\_id, employee\_id,first\_name, last\_name, salary FROM copy\_employees WHERE department\_id = 50;

Используя представление, как указано, можно INSERT, UPDATE и DELETE информацию для всех строк в представлении, даже если это приведет к тому, что строка перестанет быть частью представления. Это может быть не то, что планировал администратор базы данных при создании представления. Для управления доступом к данным в инструкцию CREATE VIEW можно добавить две опции:

−WITH CHECK OPTION

−WITH READ ONLY

Представление определяется без WITH CHECK OPTION: CREATE VIEW view\_dept50 AS SELECT department\_id, employee\_id, first\_name, last\_name, salary FROM copy\_employees WHERE department\_id = 50;

Параметр WITH CHECK OPTION гарантирует, что операции DML, выполняемые над представлением, остаются в домене представления. Любая попытка изменить номер отдела для любой строки в представлении терпит неудачу, поскольку нарушает ограничение WITH CHECK OPTION.

Параметр WITH READ ONLY гарантирует, что в представлении не будет выполняться никаких операций DML. Любая попытка выполнить оператор INSERT, UPDATE или DELETE приведет к ошибке сервера Oracle. Пример: CREATE OR REPLACE VIEW view\_dept50 AS SELECT department\_id, employee\_id, first\_name, last\_name, salary FROM employees WHERE department\_id = 50 WITH READ ONLY;

Простые представления и сложные представления различаются по своей способности разрешать операции DML через представление. Для простых представлений операции DML могут выполняться через представление. Для сложных представлений операции DML не всегда разрешены. При выполнении операций DML над представлениями необходимо учитывать следующие три правила. Вы не можете удалить строку из базовой таблицы, если представление содержит что-либо из следующего:

−Групповые функции

− Предложение GROUP BY

− Ключевое слово DISTINCT

−Псевдостолбец ROWNUM Ключевое слово

Вы не можете изменять данные через представление, если представление содержит:

−Групповые функции

− Предложение GROUP BY

− Ключевое слово DISTINCT

− Ключевое слово псевдостолбца ROWNUM

− Столбцы, определенные выражениями

Вы не можете добавлять данные через представление, если представление:

− включает групповые функции

− включает предложение GROUP BY

− включает ключевое слово DISTINCT

− включает ключевое слово псевдостолбца ROWNUM

− включает столбцы, определенные выражениями

− не включает столбцы NOT NULL в базовые таблицы.

**15-3 Managing Views (Управление представлениями)**

Поскольку представление не содержит собственных данных, его удаление не влияет на данные в базовых таблицах. Если представление использовалось для INSERT, UPDATE, или DELETE данных в прошлом, эти изменения в базовых таблицах остаются. Удаление представления просто удаляет определение представления из базы данных. Помните, что представления хранятся как операторы SELECT в словаре данных. Только создатель или пользователи с привилегией DROP ANY VIEW могут удалить представление. Синтаксис SQL для удаления представления: DROP VIEW viewname;

Встроенные представления также называются подзапросами в предложении FROM. Вы вставляете подзапрос в предложение FROM так же, как если бы подзапрос был именем таблицы. Встроенные представления обычно используются для упрощения сложных запросов за счет удаления операций соединения и объединения нескольких запросов в один. Пример: SELECT e.last\_name, e.salary, e.department\_id, d.maxsal FROM employees e, (SELECT department\_id, max(salary) maxsal FROM employees GROUP BY department\_id) d WHERE e.department\_id = d.department\_id AND e.salary = d.maxsal; Данные, возвращаемые подзапросом, получают псевдоним (d), который затем используется в сочетании с основным запросом для возврата выбранных столбцов из обоих источников запросов.

Top-n-analysis — это операция SQL, используемая для ранжирования результатов. Использование top-n-analysis полезно, когда вы хотите получить первые 5 записей или первые n записей результирующего набора, возвращаемого запросом: SELECT ROWNUM AS "Longest employed", last\_name, hire\_date FROM employees WHERE ROWNUM <=5 ORDER BY hire\_date;

Запрос top-n-analysis использует встроенное представление (подзапрос) для возврата набора результатов. Вы можете использовать ROWNUM в запросе, чтобы присвоить номер строки результирующему набору. Затем основной запрос использует ROWNUM для упорядочения данных и возврата первых пяти: SELECT ROWNUM AS "Longest employed", last\_name, hire\_date FROM (SELECT last\_name, hire\_date FROM employees ORDER BY hire\_date) WHERE ROWNUM <=5; Предложение внешнего запроса WHERE используется для ограничения количества возвращаемых строк и должно использовать оператор < или <=.

**Sequences and Synonyms**

**16-1 Working With Sequences (Работа с последовательностями)**

Существует три объекта базы данных: TABLE, VIEW и SEQUENCE (последовательность). SEQUENCE — это совместно используемый объект, используемый для автоматической генерации уникальных номеров. Поскольку это общий объект, к нему могут получить доступ несколько пользователей. Обычно последовательности используются для создания значения первичного ключа. Первичные ключи должны быть уникальными для каждой строки. Последовательность генерируется и увеличивается (или уменьшается) внутренней процедурой Oracle. Этот объект экономит ваше время, потому что он уменьшает объем кода, который вам нужно написать. Порядковые номера хранятся и генерируются независимо от таблиц. Поэтому одну и ту же последовательность можно использовать для нескольких таблиц. Синтаксис SEQUENCE: CREATE SEQUENCE sequence [INCREMENT BY n] [START WITH n] [{MAXVALUE n | NOMAXVALUE}] [{MINVALUE n | NOMINVALUE}] [{CYCLE | NOCYCLE}] [{CACHE n | NOCACHE}];

sequence - имя генератора последовательности (объекта)

INCREMENT BY n -

указывает интервал между порядковыми номерами, где n — целое число

START WITH n - указывает первый порядковый номер, который будет сгенерирован

MAXVALUE n — указывает максимальное значение, которое может сгенерировать последовательность.

NOMAXVALUE — указывает максимальное значение 10^27 для восходящей последовательности и -1 для нисходящей последовательности (по умолчанию).

MINVALUE n - указывает минимальное значение последовательности

NOMINVALUE — указывает минимальное значение 1 для восходящей последовательности и –(10^26) для восходящей последовательности (по умолчанию).

CYCLE | NOCYCLE — указывает, продолжает ли последовательность генерировать значения после достижения максимального или минимального значения (NOCYCLE — это параметр по умолчанию).

CACHE n | NOCACHE — указывает, сколько значений сервер Oracle предварительно выделяет и хранит в памяти. (По умолчанию сервер Oracle кэширует 20 значений.) В случае сбоя системы значения теряются.

Пример: CREATE SEQUENCE runner\_id\_seq INCREMENT BY 1 START WITH 1 MAXVALUE 50000 NOCACHE NOCYCLE;

Параметр NOCACHE предотвращает кэширование значений SEQUENCE в памяти, что в случае сбоя системы предотвращает потерю предварительно выделенных и хранящихся в памяти номеров. Параметр NOCYCLE предотвращает начало нумерации с 1, если превышено значение 50 000. Не используйте параметр CYCLE, если последовательность используется для генерации значений первичного ключа, если только не существует надежного механизма, удаляющего старые строки быстрее, чем добавляются новые.

Чтобы убедиться, что последовательность была создана, запросите словарь данных USER\_OBJECTS. Чтобы просмотреть все настройки SEQUENCE, запросите словарь данных USER\_SEQUENCES. Пример: SELECT sequence\_name, min\_value, max\_value, increment\_by, last\_number FROM user\_sequences;

Если указано NOCACHE, столбец last\_number в запросе отображает следующий доступный порядковый номер. Если указан параметр CACHE, в столбце last\_number отображается следующее доступное число в последовательности, которая не была кэширована в памяти.

Псевдостолбец NEXTVAL используется для извлечения последовательных порядковых номеров из указанной последовательности. Вы должны указать NEXTVAL с именем последовательности. Когда вы ссылаетесь на sequence.NEXTVAL, генерируется новый порядковый номер, а текущий порядковый номер помещается в CURRVAL. Пример: INSERT INTO departments (department\_id, department\_name, location\_id) VALUES (departments\_seq.NEXTVAL, 'Support', 2500);

NEXTVAL необходимо использовать для создания порядкового номера в сеансе текущего пользователя, прежде чем можно будет сослаться на CURRVAL. Вы должны указать CURRVAL с именем последовательности. При ссылке на sequence.CURRVAL возвращается последнее значение, сгенерированное процессом этого пользователя: INSERT INTO employees (employee\_id, department\_id, ...) VALUES (employees\_seq.NEXTVAL, dept\_deptid\_seq.CURRVAL, ...);

После создания последовательности она генерирует порядковые номера для использования в ваших таблицах. Ссылайтесь на значения последовательности, используя псевдостолбцы NEXTVAL и CURRVAL. Вы можете использовать NEXTVAL и CURRVAL в следующих контекстах:

− Список SELECT оператора SELECT, который не является частью подзапроса.

− Список SELECT подзапроса в операторе INSERT

− Предложение VALUES оператора INSERT

− Предложение SET оператора UPDATE

Вы не можете использовать NEXTVAL и CURRVAL в следующих контекстах:

− Список SELECT представления

− Оператор SELECT с ключевым словом DISTINCT

− Оператор SELECT с предложениями GROUP BY, HAVING или ORDER BY.

− Подзапрос в операторе SELECT, DELETE или UPDATE.

− Выражение DEFAULT в операторе CREATE TABLE или ALTER TABLE.

Пример: CREATE TABLE runners (runner\_id NUMBER(6,0) CONSTRAINT runners\_id\_pk PRIMARY KEY, first\_name VARCHAR2(30), last\_name VARCHAR2(30)); Затем мы создаем последовательность, которая будет генерировать значения для столбца первичного ключа runner\_id: CREATE SEQUENCE runner\_id\_seq INCREMENT BY 1 START WITH 1 MAXVALUE 50000 NOCACHE NOCYCLE;

Использование следующего синтаксиса позволит вставлять новых участников в таблицу бегунов: INSERT INTO runners (runner\_id, first\_name, last\_name) VALUES (runner\_id\_seq.NEXTVAL, 'Joanne', 'Everely'); Идентификационный номер бегуна будет сгенерирован путем извлечения NEXTVAL из последовательности.

Кэшированные последовательности в памяти обеспечивают более быстрый доступ к значениям последовательности. Кэш заполняется при первом обращении к последовательности. Каждый запрос следующего значения последовательности извлекается из кэшированной последовательности. После использования последнего значения последовательности следующий запрос последовательности извлекает в память еще один кэш последовательностей. 20 — это количество кэшируемых порядковых номеров по умолчанию.

Хотя генераторы последовательностей выдают порядковые номера без пропусков, это действие происходит независимо от фиксации или отката базы данных. Пробелы (непоследовательные числа) могут быть созданы:

− При откате оператора, содержащего последовательность, число теряется

− Системный сбой. Если последовательность кэширует значения в памяти и происходит сбой системы, эти значения теряются.

− Одна и та же последовательность используется для нескольких таблиц. Если вы сделаете это, каждая таблица может содержать пробелы в порядковых номерах.

Если последовательность была создана с помощью NOCACHE, можно просмотреть следующее доступное значение последовательности, не увеличивая его, запросив таблицу USER\_SEQUENCES: SELECT sequence\_name, min\_value, max\_value, last\_number AS "Next number" FROM USER\_SEQUENCES WHERE sequence\_name = 'RUNNER\_ID\_SEQ';

Как и в случае с другими созданными вами объектами базы данных, SEQUENCE также можно изменить с помощью оператора ALTER SEQUENCE. Последовательность может быть изменена для увеличения MAXVALUE без изменения существующего порядка чисел. Пример: ALTER SEQUENCE runner\_id\_seq INCREMENT BY 1 MAXVALUE 999999 NOCACHE NOCYCLE;

Некоторая проверка выполняется при изменении последовательности. Например, новое MAXVALUE, которое меньше текущего порядкового номера, не может быть выполнено: ALTER SEQUENCE runner\_id\_seq INCREMENT BY 1 MAXVALUE 90 NOCACHE NOCYCLE;

При выполнении инструкции ALTER SEQUENCE применяются несколько рекомендаций.

• Они есть:

− Вы должны быть владельцем или иметь привилегию ALTER для последовательности, чтобы изменить ее.

− Оператор ALTER SEQUENCE влияет только на будущие порядковые номера.

− Параметр НАЧАТЬ С нельзя изменить с помощью ALTER SEQUENCE. Последовательность должна быть удалена и создана заново, чтобы перезапустить последовательность с другим номером.

Чтобы удалить последовательность из словаря данных, используйте оператор DROP SEQUENCE. Вы должны быть владельцем последовательности или иметь привилегии DROP ANY SEQUENCE, чтобы удалить ее. После удаления на последовательность больше нельзя ссылаться. Пример: DROP SEQUENCE runner\_id\_seq;

**16-2 Indexes and Synonyms (Индексы и синонимы)**

Индекс Oracle Server — это объект схемы, который может ускорить извлечение строк с помощью указателя. Индексы могут быть созданы явно или автоматически. Если у вас нет индекса для выбранного столбца, выполняется полное сканирование таблицы. Индекс обеспечивает прямой и быстрый доступ к строкам в таблице. Его цель — уменьшить потребность в дисковом вводе-выводе (вводе-выводе) за счет использования индексированного пути для быстрого поиска данных. Индекс используется и поддерживается сервером Oracle автоматически. После создания индекса от пользователя не требуется никаких действий. ROWID представляет собой строковое представление адреса строки с основанием 64, содержащее идентификатор блока, расположение строки в блоке и идентификатор файла базы данных. Индексы используют ROWID, потому что это самый быстрый способ доступа к любой конкретной строке. Индексы логически и физически независимы от таблицы, которую они индексируют. Это означает, что их можно создавать или удалять в любое время и они не влияют на базовые таблицы или другие индексы. Когда вы удаляете таблицу, соответствующие индексы также удаляются.

Можно создать два типа индексов:

− Уникальный индекс: сервер Oracle автоматически создает этот индекс, когда вы определяете столбец в таблице для ограничения PRIMARY KEY или UNIQUE KEY.

Имя индекса — это имя, данное ограничению. Хотя вы можете создать уникальный индекс вручную, рекомендуется создать ограничение уникальности в таблице, которое неявно создает уникальный индекс.

− Неуникальный индекс: это индекс, который пользователь может создать для ускорения доступа к строкам.

Синтаксис создания: CREATE INDEX index\_name ON table\_name( column...,column)

Чтобы создать индекс в схеме, вы должны иметь привилегию CREATE TABLE. Чтобы создать индекс в любой схеме, вам потребуется привилегия CREATE ANY INDEX или привилегия CREATE TABLE для таблицы, для которой вы создаете индекс. Нулевые значения не включаются в индекс. Пример: CREATE INDEX wf\_cont\_reg\_id\_idx ON wf\_countries(region\_id);

Индекс следует создавать только в том случае, если:

− Столбец содержит широкий диапазон значений

− Столбец содержит большое количество нулевых значений.

− Один или несколько столбцов часто используются вместе в предложении WHERE или условии соединения.

− Таблица большая, и ожидается, что большинство запросов будут извлекать менее 2-4% строк.

При принятии решения о создании индекса больше не всегда значит лучше. Каждая операция DML (INSERT, UPDATE, DELETE), выполняемая над таблицей с индексами, означает, что индексы должны быть обновлены. Чем больше индексов связано с таблицей, тем больше усилий требуется для обновления всех индексов после операции DML. Обычно не стоит создавать индекс, если:

−Стол маленький

− Столбцы не часто используются в качестве условия в запросе.

− Ожидается, что большинство запросов будут извлекать более 2-4% строк в таблице.

− Таблица часто обновляется

− На индексированные столбцы ссылаются как на часть выражения

Составной индекс (также называемый «сцепленным» индексом) — это индекс, который вы создаете для нескольких столбцов в таблице. Столбцы в составном индексе могут появляться в любом порядке и не обязательно должны быть соседними в таблице. Составные индексы могут ускорить извлечение данных для инструкций SELECT, в которых предложение WHERE ссылается на все столбцы составного индекса или на его начальную часть: CREATE INDEX emps\_name\_idx ON employees(first\_name, last\_name); Нулевые значения не включаются в составной индекс. Чтобы оптимизировать соединения, вы можете создать индекс для столбца FOREIGN KEY, который ускорит поиск для сопоставления строк со столбцом PRIMARY KEY. Оптимизатор не использует индекс, если предложение WHERE содержит выражение IS NULL. Подтвердите существование индексов из представления словаря данных USER\_INDEXES. Вы также можете проверить столбцы, участвующие в индексе, запросив представление USER\_IND\_COLUMNS. Пример: SELECT DISTINCT ic.index\_name, ic.column\_name, ic.column\_position, id.uniqueness FROM user\_indexes id, user\_ind\_columns ic WHERE id.table\_name = ic.table\_name AND ic.table\_name = 'EMPLOYEES';

Индекс на основе функций хранит проиндексированные значения и использует индекс на основе инструкции SELECT для извлечения данных. Функциональный индекс — это индекс, основанный на выражениях. Выражение индекса строится из столбцов таблицы, констант, функций SQL и пользовательских функций. Индексы на основе функций полезны, когда вы не знаете, в каком случае данные были сохранены в базе данных. Индексы на основе функций, определенные с помощью ключевых слов UPPER(имя\_столбца) или LOWER(имя\_столбца), позволяют выполнять поиск без учета регистра. Если вы не знаете, как фамилии сотрудников были введены в базу данных, вы все равно можете использовать индекс, введя заглавные буквы в операторе SELECT. Когда запрос изменяется с помощью выражения в предложении WHERE, индекс не будет использовать его, если вы не создадите индекс на основе функции для соответствия выражению. Пример: CREATE INDEX upper\_last\_name\_idx ON employees (UPPER(last\_name)); SELECT \* FROM employees WHERE UPPER(last\_name) LIKE 'KIN%';

Чтобы убедиться, что сервер Oracle использует индекс, а не выполняет полное сканирование таблицы, убедитесь, что значение функции не равно null в последующих запросах. Например, следующий оператор гарантированно использует индекс, но без предложения WHERE сервер Oracle может выполнить полное сканирование таблицы: SELECT \* FROM employees WHERE UPPER (last\_name) IS NOT NULL ORDER BY UPPER (last\_name); Сервер Oracle обрабатывает индексы со столбцами, помеченными DESC, как индексы на основе функций. Столбцы с пометкой DESC сортируются в порядке убывания.

Вы не можете изменять индексы. Чтобы изменить индекс, вы должны удалить его, а затем создать заново. Удалите определение индекса из словаря данных, выполнив оператор DROP INDEX. Чтобы удалить индекс, вы должны быть владельцем индекса или иметь привилегию DROP ANY INDEX. Если вы удаляете таблицу, индексы и ограничения автоматически удаляются, но представления и последовательности остаются. Пример: DROP INDEX upper\_last\_name\_idx;

В SQL, как и в языке, синоним — это слово или выражение, которое является принятой заменой другого слова. Синонимы используются для упрощения доступа к объектам путем создания другого имени для объекта. Синонимы могут упростить обращение к таблице, принадлежащей другому пользователю, и сократить длинные имена объектов. Создание синонима устраняет необходимость уточнения имени объекта схемой и предоставляет альтернативное имя для таблицы, представления, последовательности, процедуры или другого объекта. Этот метод может быть особенно полезен при использовании длинных имен объектов, таких как представления. Администратор базы данных может создать общедоступный синоним, доступный для всех пользователей, и может специально предоставить привилегию CREATE PUBLIC SYNONYM любому пользователю, и этот пользователь может создавать общедоступные синонимы. Синтаксис: CREATE [PUBLIC] SYNONYM synonym FOR object;

PUBLIC: создает синоним, доступный для всех пользователей

synonym: имя создаваемого синонима

object: идентифицирует объект, для которого создается синоним

Пример: CREATE SYNONYM amy\_emps FOR amy\_copy\_employees;

− Объект не может содержаться в пакете

− Частное синонимическое имя должно отличаться от всех других объектов, принадлежащих тому же пользователю.

Чтобы удалить синоним: DROP [PUBLIC] SYNONYM name\_of\_synonym Пример: DROP SYNONYM amy\_emps;

Существование синонимов можно подтвердить, запросив представление словаря данных USER\_SYNONYMS.

**Privileges and Regular Expressions**

**17-1 Controlling User Access (Управление доступом пользователей)**

В многопользовательской среде необходимо обеспечить безопасность доступа к базе данных и ее использования. С помощью безопасности базы данных Oracle Server вы можете делать следующее:

−Управление доступом к базе данных

−Дать доступ к определенным объектам в базе данных

−Подтверждение предоставленных и полученных привилегий в данных Oracle

толковый словарь

− Создание синонимов для объектов базы данных

Безопасность базы данных можно разделить на две категории:

−Системная безопасность

− Безопасность данных

Безопасность системы охватывает доступ и использование базы данных на системном уровне, например создание пользователей, имен пользователей и паролей, выделение дискового пространства пользователям и предоставление системных привилегий, которые могут выполнять пользователи, например создание таблиц, представлений и последовательностей. Существует более 100 различных системных привилегий. Безопасность данных (также известная как безопасность объектов) относится к привилегиям объектов, которые охватывают доступ и использование объектов базы данных, а также действия, которые эти пользователи могут выполнять над объектами. Эти привилегии включают возможность выполнять операторы DML.

Привилегии — это право выполнять определенные операторы SQL. Администратор базы данных — это пользователь высокого уровня, который может предоставлять пользователям доступ к базе данных и ее объектам. Пользователям требуются системные привилегии, чтобы получить доступ к базе данных. Им требуются объектные привилегии для управления содержимым объектов в базе данных. Пользователям также может быть предоставлена привилегия предоставлять дополнительные привилегии другим пользователям или ролям, которые представляют собой именованные группы связанных привилегий. Схема — это набор объектов, таких как таблицы, представления и последовательности. Схема принадлежит пользователю базы данных и имеет то же имя, что и этот пользователь. В этом курсе имя вашей схемы представляет собой комбинацию вашей страны/штата, школы, курса и учащегося.

количество.

Этот уровень безопасности охватывает доступ и использование базы данных на системном уровне. Системные привилегии, такие как возможность создавать или удалять пользователей, удалять таблицы или создавать резервные копии таблиц, обычно принадлежат только администратору баз данных.

Администратор базы данных создает пользователя, выполняя оператор CREATE USER. На данный момент у пользователя нет никаких привилегий. Затем администратор базы данных может предоставить этому пользователю необходимые привилегии. Синтаксис: CREATE USER user IDENTIFIED BY password; Пример: CREATE USER scott IDENTIFIED BY ur35scott; Используя оператор ALTER USER, пользователь может изменить свой пароль. Пример: ALTER USER scott IDENTIFIED BY imscott35;

Администратор базы данных использует оператор GRANT для предоставления пользователю системных привилегий. Системные привилегии определяют, что пользователь может делать на уровне базы данных. Как только пользователю были предоставлены привилегии, пользователь может немедленно использовать эти привилегии. Синтаксис: GRANT privilege [, privilege...] TO user [, user| role, PUBLIC...]; Пример: GRANT create session, create table, create sequence, create view TO scott;

Пользователь должен иметь привилегию CREATE SESSION и идентификатор пользователя, если он должен иметь доступ к базе данных. Вы не можете выполнить команду CREATE SESSION в Oracle Application Express; это происходит автоматически за кулисами.

Этот уровень безопасности охватывает доступ и использование объектов базы данных, а также действия, которые пользователи могут выполнять с этими объектами.

Каждый объект имеет определенный набор предоставляемых привилегий.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Привилегия объекта** | **Table** | **View** | **Sequence** | **Procedure** |
| **ALTER** | X |  | X |  |
| **DELETE** | X | X |  |  |
| **EXECUTE** |  |  |  | X |
| **INDEX** | X | X |  |  |
| **INSERT** | X | X |  |  |
| **REFERENCES** | X |  |  |  |
| **SELECT** | X | X | X |  |
| **UPDATE** | X | X |  |  |

Важно отметить следующие четыре момента, касающиеся объектных привилегий:

− К последовательности применяются только привилегии SELECT и ALTER.

− Помните, что последовательность использует ALTER для изменения параметров INCREMENT, MAXVALUE, CACHE/NOCACHE или CYCLE/NOCYCLE.

−START WITH нельзя изменить с помощью ALTER

Вы можете предоставить привилегии UPDATE, REFERENCES и INSERT для отдельных столбцов в таблице. Пример: GRANT UPDATE (salary) ON employees TO steven\_king Привилегию SELECT можно ограничить, создав представление с подмножеством столбцов и предоставив привилегию SELECT только для представления. Вы не можете предоставить SELECT для отдельных столбцов. Привилегия, предоставленная синониму, преобразуется в привилегию базовой таблицы, на которую ссылается синоним. Другими словами, синоним — это просто новое, более простое в использовании имя. Использование этого имени для предоставления привилегии аналогично предоставлению привилегии самой таблице.

Владелец таблицы может предоставить доступ всем пользователям, используя ключевое слово PUBLIC. Пример: GRANT select ON alice.departments TO PUBLIC;

Если в операторе не используется полное имя объекта, сервер Oracle неявно добавляет к имени объекта префикс с именем текущего пользователя (или схемой). Если в операторе не используется полное имя объекта, и текущий пользователь не владеет объектом с таким именем, система добавляет к имени объекта префикс PUBLIC.

Если вы попытаетесь выполнить неавторизованную операцию, такую как удаление строки из таблицы, для которой у вас нет привилегии DELETE, сервер Oracle не разрешит выполнение этой операции. Если вы получаете сообщение об ошибке сервера Oracle «таблица или представление не существует», вы сделали одно из следующих действий:

− назвал несуществующую таблицу или представление.

− Попытка выполнить операцию над таблицей или представлением, для которой у вас нет соответствующей привилегии.

Вы можете получить доступ к словарю данных, чтобы просмотреть имеющиеся у вас привилегии.

**17-2 Creating and Revoking Object Privileges (Создание и отзыв объектных привилегий)**

Роль — это именованная группа связанных привилегий, которые могут быть предоставлены пользователю. Этот метод упрощает отмену и сохранение привилегий. Пользователь может иметь доступ к нескольким ролям, и одна и та же роль может быть назначена нескольким пользователям. Роли обычно создаются для приложения базы данных. Чтобы создать и назначить роль, сначала администратор базы данных должен создать роль. Затем администратор базы данных может назначить привилегии роли, а роль — пользователям. Пример: CREATE ROLE manager; GRANT create table, create view TO manager; GRANT manager TO jennifer\_cho; Синтаксис для создания роли: CREATE ROLE role\_name; После создания роли администратор базы данных может использовать оператор GRANT, чтобы назначить роль пользователям, а также присвоить роли привилегии.

Если пользователям предоставлено несколько ролей, они получают все привилегии, связанные со всеми ролями. CREATE ROLE — это системная привилегия, которая не была предоставлена классам Академии.

Роли — это именованные группы связанных привилегий. Они могут быть предоставлены пользователям. Они упрощают процесс предоставления и отзыва привилегий. Они создаются администратором баз данных. Синтаксис для предоставления объектных привилегий: GRANT object\_priv [(column\_list)] ON object\_name TO {user|role|PUBLIC} [WITH GRANT OPTION];

Чтобы предоставить привилегии объекту, объект должен находиться в вашей собственной схеме или вам должны быть предоставлены привилегии с помощью параметра WITH GRANT OPTION. Владелец объекта может предоставить любую объектную привилегию для объекта любому другому пользователю или роли базы данных. Владелец объекта автоматически получает все объектные привилегии для этого объекта.

Привилегия, предоставленная с помощью предложения WITH GRANT OPTION, может быть передана другим пользователям и ролям получателем привилегии. Объектные привилегии, предоставленные с помощью предложения WITH GRANT OPTION, отменяются, когда отменяется привилегия лица, предоставившего право. Пример: GRANT SELECT, INSERT ON clients TO scott\_king WITH GRANT OPTION;

Владелец таблицы может предоставить доступ всем пользователям, используя ключевое слово PUBLIC. Пример: GRANT SELECT ON jason\_tsang.clients TO PUBLIC;

Если вы попытаетесь выполнить неавторизованную операцию, такую как удаление строки из таблицы, для которой у вас нет привилегии DELETE, сервер Oracle не разрешит выполнение этой операции. Если вы получаете сообщение об ошибке Oracle Server «таблица или представление не существует», вы выполнили одно из следующих действий:

−Ссылка на несуществующую таблицу или представление.

− Попытка выполнить операцию над таблицей или представлением, для которой у вас нет соответствующих привилегий.

Вы можете удалить привилегии, предоставленные другим пользователям, с помощью оператора REVOKE. Когда вы используете оператор REVOKE, указанные вами привилегии отменяются у пользователей, которых вы называете, и у любых других пользователей, которым эти привилегии были предоставлены с помощью предложения WITH GRANT OPTION. Синтаксис для отзыва объектных привилегий: REVOKE {privilege [, privilege...]|ALL} ON object FROM {user[, user...]|role|PUBLIC} [CASCADE CONSTRAINTS];

CASCADE CONSTRAINTS требуется для удаления любых ограничений ссылочной целостности, наложенных на объект с помощью привилегии REFERENCES. Пример: REVOKE SELECT, INSERT ON clients FROM scott\_king; Если пользователю предоставлена привилегия с помощью предложения WITH GRANT OPTION, этот пользователь также может предоставить привилегию с помощью предложения WITH GRANT OPTION. Это означает, что возможна длинная цепочка получателей грантов, но циклические гранты не допускаются. Если владелец отменяет привилегию у пользователя, который предоставил привилегии другим пользователям, инструкция отзыва распространяется на все предоставленные привилегии.

Как упоминалось ранее в этом уроке, вы можете создать синоним, чтобы исключить необходимость уточнения имени объекта схемой и предоставить вам альтернативное имя для таблицы, представления, последовательности, процедуры или другого объекта. Синонимы могут быть как частными (по умолчанию), так и общедоступными. Общедоступный синоним может быть создан администраторами базы данных или пользователями базы данных, которым предоставлены соответствующие права, но не все могут автоматически создавать общедоступные синонимы. Привилегия CREATE PUBLIC SYNONYM не была предоставлена студентам Академии.

Роли и привилегии различаются по ряду признаков:

− Привилегия пользователя — это право выполнять оператор SQL определенного типа или право доступа к объекту другого пользователя.

−Все привилегии определяются Oracle

− Роли, с другой стороны, создаются пользователями (обычно администраторами) и используются для группировки привилегий или других ролей.

− Они созданы, чтобы упростить управление предоставлением пользователям нескольких привилегий или ролей.

− Привилегии предоставляются вместе с базой данных, а роли назначаются администраторами базы данных или пользователями конкретной базы данных.

Ссылка на базу данных — это указатель, который определяет односторонний путь связи от одной базы данных Oracle к другой базе данных. Указатель ссылки фактически определяется как запись в таблице словаря данных. Чтобы получить доступ к ссылке, вы должны быть подключены к локальной базе данных, содержащей запись словаря данных. Соединение с базой данных является «односторонним» в том смысле, что клиент, подключенный к локальной базе данных A, может использовать ссылку, хранящуюся в базе данных A, для доступа к информации в удаленной базе данных B, но пользователи, подключенные к базе данных B, не могут использовать ту же ссылку для доступа. данные в базе данных A. CREATE DATABASE LINK — в Oracle Application Express нет постоянного подключения к базе данных, и, как следствие, эта функция недоступна. Если локальные пользователи в базе данных B хотят получить доступ к данным в базе данных A, они должны определить ссылку, которая хранится в словаре данных базы данных B. Соединение со ссылкой на базу данных предоставляет локальным пользователям доступ к данным в удаленной базе данных. Чтобы это соединение произошло, каждая база данных в распределенной системе должна иметь уникальное глобальное имя базы данных. Глобальное имя базы данных однозначно идентифицирует сервер базы данных в распределенной системе. Большое преимущество ссылок на базы данных состоит в том, что они позволяют пользователям получать доступ к объектам других пользователей в удаленной базе данных, так что они ограничены набором привилегий владельца объекта. Другими словами, локальный пользователь может получить доступ к удаленной базе данных, не будучи пользователем удаленной базы данных. Как правило, администратор базы данных отвечает за создание связи с базой данных. Представление словаря USER\_DB\_LINKS содержит информацию о ссылках, к которым у пользователя есть доступ. После создания связи с базой данных вы можете писать операторы SQL для данных на удаленном сайте. Если задан синоним, вы можете писать операторы SQL, используя этот синоним. Пример: CREATE PUBLIC SYNONYM HQ\_EMP FOR emp@HQ.ACME.COM;

**17-3 Regular Expressions (Обычные выражения)**

Использование регулярных выражений основано на использовании метасимволов:

− Метасимволы — это специальные символы, имеющие особое значение, такие как подстановочный знак, повторяющийся символ, несовпадающий символ или диапазон символов.

− Вы можете использовать несколько предопределенных символов метасимволов в сопоставлении с образцом.

− На следующих слайдах перечислены некоторые метасимволы и даны краткие пояснения по каждому из них.

|  |  |
| --- | --- |
| **Символ** | **Описание** |
| **. (точка)** | Соответствует любому символу из поддерживаемого набора символов, кроме NULL. |
| **?** | Соответствует нулю или одному вхождению. |
| **\*** | Соответствует нулю или более вхождений. |
| **+** | Соответствует одному или нескольким вхождениям. |
| **()** | Выражение группировки, рассматриваемое как одно подвыражение. |
| **\** | Escape-символ. |
| **|** | Оператор альтернативы для указания альтернативных совпадений. |
| **^/$** | Соответствует началу строки/концу строки. |
| **[]** | Выражение скобки для списка соответствия, соответствующее любому из выражений, представленных в списке. |

Простое регулярное выражение очень похоже на поиск по подстановочным знакам.

Давайте рассмотрим пример: воспользуемся оператором точки, чтобы найти букву «а», за которой следует любой символ, за которым следует буква «с». В качестве регулярного выражения это будет выглядеть так: 'a.c'. То же выражение, что и стандартный поиск по подстановочным знакам SQL, будет выглядеть так: WHERE column LIKE 'a\_c'.

Oracle предоставляет набор функций SQL, которые можно использовать для поиска и обработки строк с помощью регулярных выражений. Вы можете использовать эти функции для любого типа данных, который содержит символьные данные, такие как CHAR, CLOB и VARCHAR2. Регулярное выражение должно быть заключено в одинарные кавычки.

|  |  |
| --- | --- |
| **Имя** | **Описание** |
| **REGEXP\_LIKE** | Аналогичен оператору LIKE, но выполняет сопоставление с регулярным выражением вместо простого сопоставления с образцом. |
| **REGEXP\_REPLACE** | Ищет шаблон регулярного выражения и заменяет его строкой замены. |
| **REGEXP\_INSTR** | Ищет заданную строку для шаблона регулярного выражения и возвращает позицию, в которой найдено совпадение. |
| **REGEXP\_SUBSTR** | Ищет шаблон регулярного выражения в заданной строке и возвращает совпавшую подстроку. |
| **REGEXP\_COUNT** | Возвращает количество раз, когда шаблон встречается в строке. |

Пример: SELECT first\_name, last\_name FROM employees WHERE REGEXP\_LIKE(first\_name, '^Ste(v|ph)en$');

Функция регулярного выражения REPLACE заменит один строковый шаблон другим. В примере выполняется поиск буквы «H», за которой следует любая гласная, и замена их двумя символами «\*»: SELECT last\_name, REGEXP\_REPLACE(last\_name, '^H(a|e|i|o|u)', '\*\*') AS "Name changed" FROM employees;

Функция COUNT регулярного выражения возвращает количество раз, когда шаблон встречается в строке. В этом примере выполняется поиск подвыражения "ab": SELECT country\_name, REGEXP\_COUNT(country\_name, '(ab)') AS "Count of 'ab'" FROM wf\_countries WHERE REGEXP\_COUNT(country\_name, '(ab)')>0;

Регулярные выражения также можно использовать как часть кода приложения, чтобы гарантировать, что в базе данных хранятся только допустимые данные. Можно включить вызов функции регулярного выражения, например, в ограничение CHECK. Пример: ALTER TABLE employees ADD CONSTRAINT email\_addr\_chk CHECK(REGEXP\_LIKE(email,'@')); Это гарантирует, что все адреса электронной почты будут содержать знак «@».

Используя регулярные выражения, вы можете более тщательно проверять формат адресов электронной почты, чтобы убедиться, что они действительны. Действительный адрес электронной почты будет иметь один или несколько символов, затем @, за которым следует один или несколько символов, а затем . (точка), за которым следует один или несколько символов. Пример: CREATE TABLE my\_contacts (first\_name VARCHAR2(15), last\_name VARCHAR2(15), email VARCHAR2(30) CHECK(REGEXP\_LIKE(email, '.+@.+\..+'))); Определения синтаксиса:

−.+ означает один или несколько символов

−@ символ @

−\. а . (точка) (здесь обратная косая черта — escape-символ).