SQL

Книга

Автор: Олег Иванов Copyright © 2017

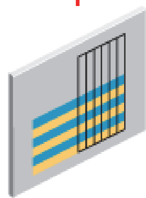
Автор: Олег Иванов Copyright © 2017
Не для коммерческого использования

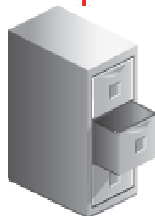
1 Реляционные базы данных

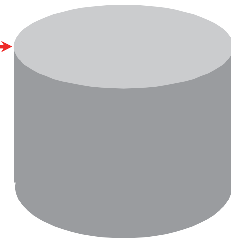
Хранение данных

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	
20	Marketing	201	
50	Shipping	124	
60	IT	103	
80	Sales	149	
90	Executive	100	
110	Accounting	205	
190	Contracting		[illegible]

GRA	LOWEST_SAL	HIGHEST_SAL
A	1000	2999
B	3000	5999
C	6000	9999
D	10000	14999
E	15000	24999
F	25000	40000







1 - 2 SQL

Хранение информации

Каждая организация работает с данными. Например, организации требуется хранить данные о клиентах, заказах, товарах, наличии товаров на складе.

Организации стараются хранить данные в электронном, а не бумажном виде. Это позволяет быстро обрабатывать данные.

Система управления базами данных (*СУБД*), *database management system* (*DBMS*) – это программа, с помощью которой организация может хранить, выбирать, модифицировать, поддерживать данные. Существуют различные модели хранения и обработки данных. Эти модели могут реализовывать БД конкретных производителей. Наиболее развита модель реляционных БД. Другие модели, имеющие научное обоснование (иерархическая и сетевая) не получили распространение по причине непрактичности.

Большую часть данных организации хранят в реляционных базах данных. Другие типы баз данных используются для хранения специфических данных. Например, системы Big Data используются для накопления неструктурированных данных для последующего анализа. Выбор типа базы данных для хранения определяется возможностями и простотой использования в программах, которые выпускаются производителями программного обеспечения. Например, при больших объемах неструктурированных данных удобнее с практической точки зрения использовать системы Big Data (Hadoop) или NoSQL (MongoDB). Анализировать данные удобнее в реляционной модели или OLAP. Поэтому, реляционные БД часто имеют интерфейсы связи с нереляционными хранилищами. Начальная обработка данных выполняется внешней системой хранения и передается в реляционную БД. Реляционные БД развиваются и получают функционал, не связанный с реляционной моделью. Например, в БД postgres есть возможность хранить и обрабатывать по логике NoSQL данные в форматах XML, JSON и др. Нереляционные технологии не стоит переоценивать (Informix и Illustra; Disapora и MongoDB).

Концепция реляционной базы данных

- Доктор Е.Ф.Кодд предложил реляционную модель для систем баз данных в 1970 г.
- Это основа системы управления реляционными базами данных (RDBMS).
- Реляционные БД:
  - совокупность двумерных таблиц (столбцы и строки)
  - доступ к данным командами языка SQL
  - непротиворечивость данных
  - транзакции

Реляционная модель

Принципы реляционной модели хранения данных были впервые сформулированы британским ученым Тедом Коддом в июне 1970 г. Наиболее распространенными в то время были иерархические и сетевые модели данных. Реляционная модель по сравнению с ними была более простой в реализации и получила программную поддержку. В 1993г. Кодд предложил модель архитектуру обработки данных OLAP, основанную на многомерных таблицах. Новый класс БД не получил распространения и функционал OLAP стал одним из расширений реляционных БД (ROLAP).

Реляционные БД стали популярными благодаря широким возможностям по обработке данных, высокой скорости работы и программной поддержке.

Язык SQL появился позже и стал фактическим стандартом для реляционных БД.

Основные особенности реализации реляционной модели в RDBMS

- Данные хранятся в виде определенных типов (строки, числа, даты), которые покрывают большую часть потребностей организаций
- Данные хранятся в виде двумерных таблиц, состоящих из столбцов (column) и строк (row). Синоним таблицы: **отношение (relation)** и **сущность (entity)**. Синоним столбца: **атрибут (attribute)**. Синоним строки: **запись (record)** и **кортеж (tuple)**. Значение в столбце строки – **поле (field)**.
- Между столбцами таблиц можно определить связи, а также другие ограничения целостности (database constrains), которые обеспечивают качество данных и их соответствие бизнес-правилам.
- Имеется набор стандартных операторов и функций для обработки данных.
- Хранящиеся данные выдаются в виде двумерных таблиц с помощью простого командного языка SQL (Structured Query Language).

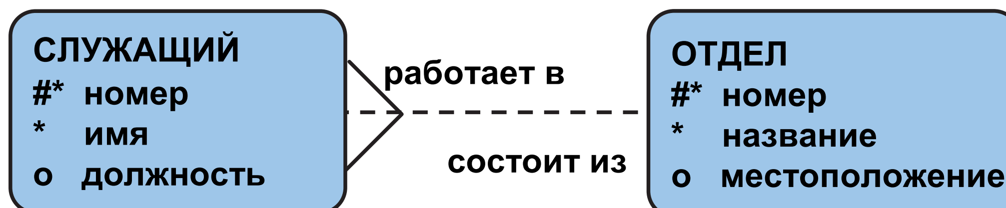
Таблицы (отношения, сущности)

- состоят из строк (кортежей, записей) и столбцов (атрибутов)
- Число и порядок столбцов в таблице фиксированы, а каждый столбец имеет имя
- Каждый столбец имеет тип данных

Таблицы
Таблица в реляционной базе данных похожа на таблицу на бумаге: она так же состоит из строк и столбцов. Число и порядок столбцов фиксированы, а каждый столбец имеет имя. Число строк переменно - оно отражает текущее количество находящихся в ней данных. SQL не даёт никаких гарантий относительно порядка строк таблицы. При чтении таблицы строки выводятся в произвольном порядке, если только явно не требуется сортировка.
В таблице можно хранить полностью идентичные строки. Однако, такое дублирование нежелательно и приводит к нежелательным эффектам. Поэтому стараются назначить столбец или несколько столбцов, значения в которых уникально идентифицируют строку.
Каждому столбцу сопоставлен тип данных. Тип данных ограничивает набор допустимых значений, которые можно присвоить столбцу, и определяет смысловое значение данных для вычислений. Например, в столбец числового типа нельзя записать обычные текстовые строки, но зато его данные можно использовать в математических вычислениях. И наоборот, если столбец имеет тип текстовой строки, для него допустимы практически любые данные, но он непригоден для математических действий (хотя другие операции, например конкатенация строк, возможны). Наиболее часто в реляционных БД используют следующие типы данных: численные, текстовые строки, дата и время.

Модель “сущность-связь“ (ER-модель)

- Создание ER-модели по документам или словесным описаниям
- ER-диаграмма - графическое представление ER-модели:



- Сценарий:
  - “. . . Сотрудник может работать только в одном отделе . . .”
  - “. . . Некоторые отделы могут не иметь сотрудников . . .”

Модель “сущность-связь“ (Entity-Relationship)

Диаграммы часто используются при описании бизнес-данных и процессов. В объектных языках программирования часто используют диаграммы классов. При создании таких диаграмм могут использоваться стандарты, например, UML (Unified Modeling Language). Бизнес-процессы могут описываться функциональными диаграммами по стандарту IDEF.

Сущность это таблица или представление (view – объект реляционной базы данных), в которой, обычно, хранятся данные одного бизнес-объекта. Например, сотрудники, отделы, заказы. Многие сущности логически связаны друг с другом. Например, сотрудники работают в отделах. ER-модель удобна для понимания того, какие данные хранятся в реляционной базе данных.

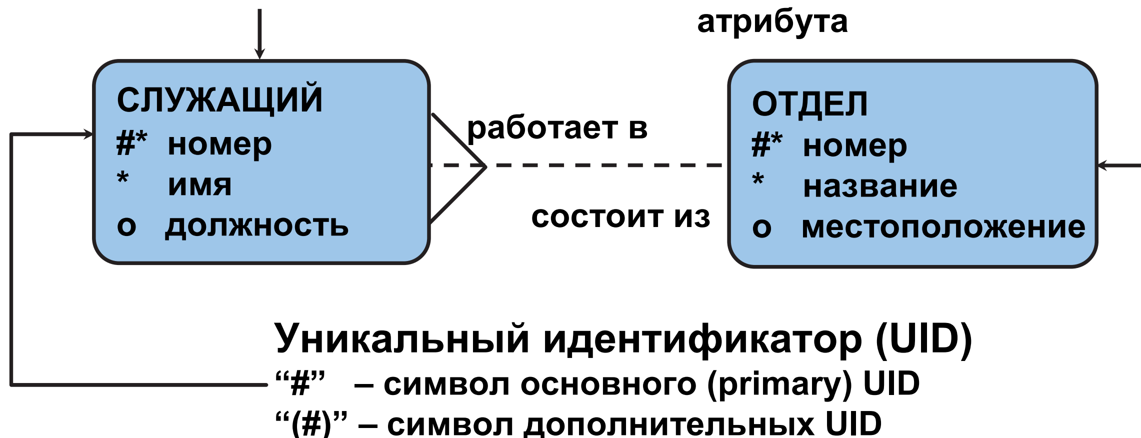
Основные компоненты

- ***Сущность* (*Entity*)** – бизнес-объект, о котором нужно хранить данные. Например, отделы, служащие и заказы. Однако, таблицы могут использоваться для хранения служебных данных. Например, связей многое ко многому, когда сотрудник может работать в нескольких отделах.
- ***Атрибут* (*Attribute*)** – столбец. Например, атрибутами сущности СЛУЖАЩИЙ могут быть номер служащего, имя, должность, дата найма, номер отдела
- ***Связь* (Relationship)** – связь между сущностями, характеризуемая *опциональностью* и *мощностью* (кардинальностью, cardinality). Например, связь между служащими и отделами. Мощность связи – сколько строк одной таблицы связано со строками другой. Например связь “один к одному”: в одном отделе один сотрудник. Опциональность - это когда сотруднику может быть назначен или не назначен отдел.

Обозначения в ER-диаграммах

Сущность
- Уникальное имя в единственном числе
- Заглавные буквы
- Скругленная рамка
- Синонимы в скобках

Атрибут
- Имя в единственном числе
- Строчные буквы
- “*” – символ обязательного атрибута
- “o” – символ необязательного атрибута



Значки в ER-диаграммах

Сущности обозначаются на диаграмме так:
- уникальное имя в единственном числе заглавными буквами;
- четырехугольник с закругленными углами любого размера;
- необязательные имена-синонимы заглавными буквами в скобках: ( ).

Атрибуты:
- уникальное имя строчными буквами;
- обязательные атрибуты (значения которых *должны* быть известны) помечаются символом “*”;
- необязательные атрибуты (значения которых *могут* быть известны) помечаются буквой “o”.

Связи

Символ	Описание
Пунктирная линия	Необязательный элемент (“может быть”)
Сплошная линия	Обязательный элемент (“должен быть”)
“Воронья лапа”	Элемент, обозначающий мощность связи “один или более”.
Одна линия	Элемент, обозначающий мощность связи “один или только один”.

Связи между таблицами

- В БД реализованы ограничениями целостности
- Для идентификации строк используется *первичный ключ* (PK) или уникальный ключ (UK)
- Внешние ключи (FK) связывают данные

Таблица EMPLOYEES

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	DEPARTMENT_ID
174	Ellen	Abel	80
142	Curtis	Davies	50
102	Lex	De Haan	90
104	Bruce	Ernst	60
202	Pat	Fay	20
206	William	Gietz	110

...

Таблица DEPARTMENTS

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
50	Shipping	124	1500
60	IT	103	1400
80	Sales	149	2500
90	Executive	100	1700
110	Accounting	205	1700
190	Contracting		1700

Главный ключ   Внешний ключ   Главный ключ

1 - 7   SQL

Связи между таблицами

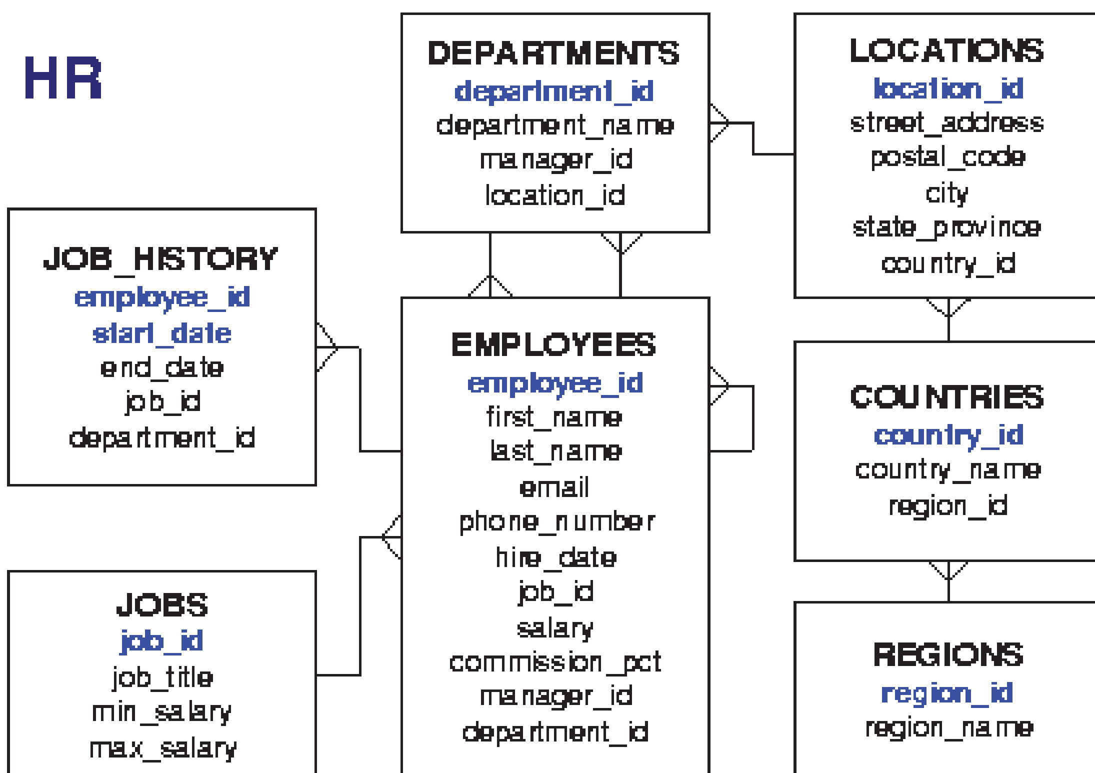
Каждая таблица содержит данные, обычно, описывающие одну бизнес-сущность. Например, таблица EMPLOYEES содержит информацию о служащих. В верхней строке каждой таблицы на слайде указаны названия столбцов, а под ними перечислены конкретные данные.

Для получения данных может понадобиться соединение двух или более таблиц. Например, требуется узнать, где находится отдел, в котором работает служащий. В этом случае нужна информация из таблицы EMPLOYEES, содержащей данные о служащих, и таблицы DEPARTAMENTS, содержащей данные об отделах.

Реляционные базы данных могут хранить таблицы, в которых могут присутствовать полностью идентичные строки, однако, это редко используется и может создать проблемы при поддержке таких таблиц. Поэтому, обычно, в каждой таблице один столбец или набор столбцов обозначается как первичный ключ. Это означает, что в таблицу нельзя вставить строки, где значения в таких столбцах уже существуют в другой сроке. Такие столбцы уникально идентифицируют сроки таблицы. Например, это может быть номер сотрудника (EMPLOYEE_ID) или номер отдела (DEPARTMENT_ID). Первичный ключ в одной таблице только один. Можно иметь в таблице и другие столбцы или наборы столбцов, уникально идентифицирующие строки (или хранящие неопределённые значения NULL), такие ключи называются уникальными (unique key - UK).

RDBMS позволяет связать данные одной таблицы с данными другой с помощью внешних ключей. *Внешний ключ* (*foreign key*) – это столбец или несколько столбцов, ссылающихся на *первичный* (*primary key*) или уникальный (*unique key*) в той же или в другой таблице.

Ограничения целостности



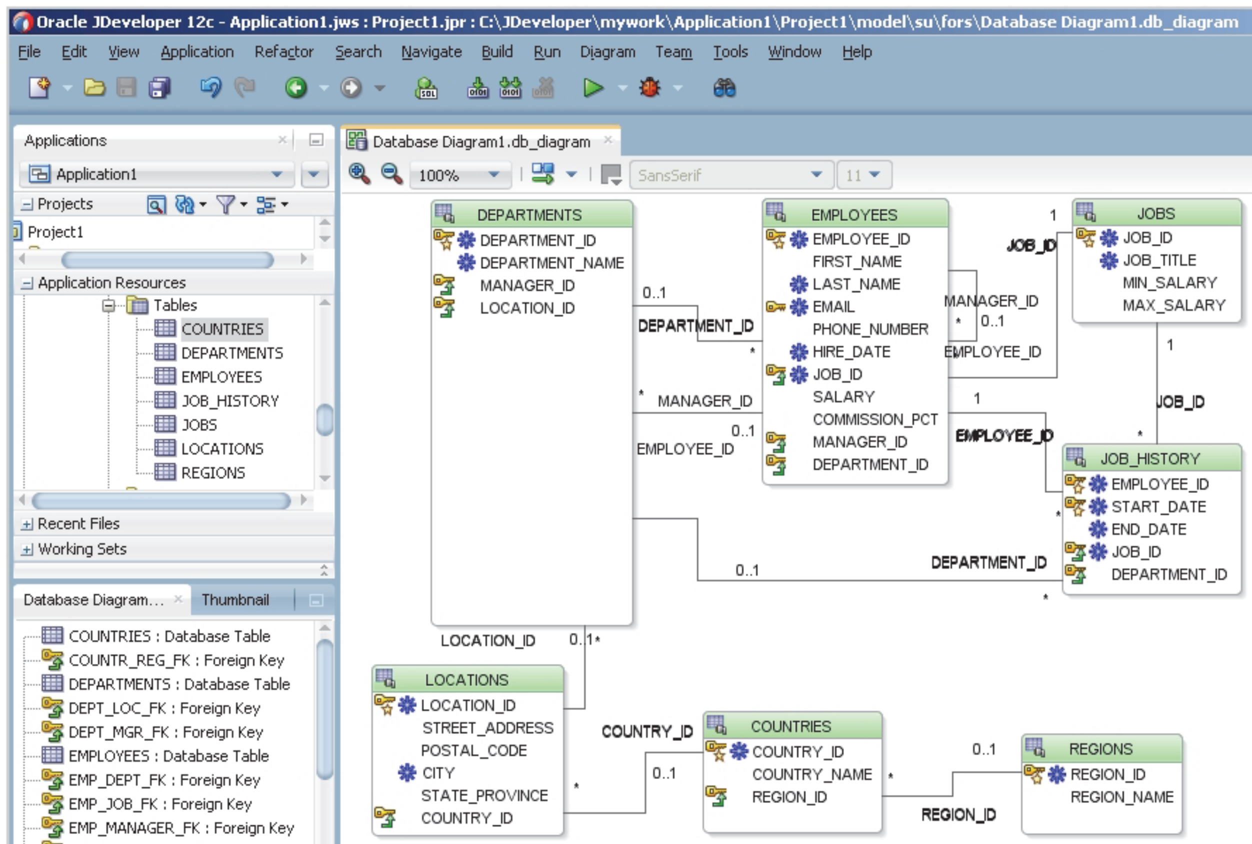
Ограничения целостности

Ограничения целостности поддерживаются БД и она не позволяет менять данные в строках таблиц так, чтобы ограничения целостности были бы нарушены. Поэтому все приложения, имеющие доступ к таблице, должны соблюдать правила, определенные в ограничениях. Реляционные базы данных позволяют менять определения ограничений целостности, добавлять или удалять их даже если в таблицах уже имеются данные.

Реляционные базы данных различных производителей могут реализовывать разные ограничения целостности. Обычно, поддерживаются следующие виды ограничений:

- `NOT NULL` ; запрещение неопределенных (пустых) значений в столбце таблицы.
- `UNIQUE` ; запрещение повторяющихся значений в столбце или наборе столбцов.
- `PRIMARY KEY`; запрещение повторяющихся и неопределенных значений в столбце или наборе столбцов.
- `FOREIGN KEY` ; каждое значение в столбце или наборе столбцов должно соответствовать значению ключа `UNIQUE` или `PRIMARY KEY` в главной таблице. Ограничение `FOREIGN KEY` задает также действия, которые выполняет Oracle в подчиненной таблице при удалении данных ключа главной таблицы.
- `CHECK` ; запрет значений, не удовлетворяющих логическому выражению.

ER-диаграмма



SQL

Взаимосвязи между таблицами

Средства разработки, например, Oracle JDeveloper, могут создавать диаграммы различных объектов, в том числе таблиц и представлений реляционных баз данных. Диаграммы могут использовать различные нотации ER, UML, диаграммы java-классов. Диаграммы могут настраиваться для более удобного отображения объектов и связей меду ними. Могут использоваться иконки, подписи к элементам диаграммы. Эти визуальные элементы являются расширениями к общим стандартам, которые использует диаграмма.

Например, в диаграмме на слайде можно убрать или добавить названия столбцов на линиях, обозначающих связи между таблицам, для того, чтобы видеть по какому столбцу идёт связь. Также на этой диаграмме можно добавить названия типов данных столбцов, ограничений целостности.

Связи между таблицами используют ключи. Правила для первичных (PK), уникальных (UK) и внешних (FK) ключей

- Значения первичного ключа не могут повторяться.
- Внешние ключи основаны на данных и являются логическими указателями, не влияющими на физическое хранение данных.
- Значение внешнего ключа должно соответствовать существующему значению первичного ключа, уникального ключа или быть неопределенным (NULL).
- Внешний ключ может ссылаться только на первичный или уникальный ключ.

Типы данных столбцов

Тип данных	Описание
VARCHAR(*size*)	**Символьные данные переменной длины**
CHAR(*size*)	**Символьные данные постоянной длины**
NUMERIC(*p,s*)	**Числовые данные переменной длины**
DATE	**Дата**
TIMESTAMP	**Дата и время с долями секунды**
LONG/text LONG RAW/bytea	**Символьные и бинарные данные большой длины**

Типы данных столбцов

Столбцы в таблицах реляционных БД имеют типы. Есть простые скалярные типы для хранения строк, чисел (целых, десятичных, с мантиссой или без), временных данных (даты, время, временные интервалы с учетом временных зон или без). БД разных производителей могут иметь типы данных разной размерности и использовать разные названия для обозначения типов данных. Например, **БД Oracle** поддерживает VARCHAR длиной до 4000 или 32767 байт и DATE хранит время с точностью до секунды, а в **БД postgres** VARCHAR до 10485760 байт, а DATE не хранит время.

Сложные типы данных

- **Примеры**

Тип данных	Описание
XML	хранение иерархических структур простых типов
JSON	простая альтернатива XML
геометрические	point line lseg box path polygon circle
LOB,CLOB	неструктурированный с произвольным доступом
boolean	TRUE, FALSE, NULL
служебные	UUID, OID, pg_lsn, ROWID

Сложные типы данных

Компании могут использовать и сложные данные. Например, текстовые и бинарные данные большого размера: фотографии товаров, медиафайлы, документы в формате PDF, RTF. Во многих БД есть типы данных для хранения больших объектов (large objects - LOB) с произвольным доступом. Работа с такими типами имеет особенности. Например, используется поточный ввод-вывод, указатель на место в LOB для произвольного доступа к части данных. Также такие столбцы имеют ограничения по использованию. Например, не могут являться частью ключа или обычных индексов. **БД postgres** не поддерживает LOB, так как функционал существующих типов достаточен.

Поддержка сложных типов включает в себя возможность их определения:

1) из каких частей состоят типы, какие части можно сопоставить простым типам

2) набор функций и операторов для обработки сложных типов. Например, тип может хранить текст, состоящий из слов. Нужно выбрать строки, в которых встречаются слова и словоформы (игнрировать суффиксы слов). Тип может хранить сетевые адреса IPv4, нужны функции, определяющие относится два IPv4 адреса к одной подсети. Тип может хранить пространственные координаты, нужно определить расстояние между двумя точками.

3) поддержку быстрой обработки данных. Например, для поиска словоформ в текстах нужны индексы, которые будут хранить словарь слов и индексировать тексты.

Выбор типа данных

- Компактность физического хранения
- Скорость обработки
- Простота использования
- Поддержка функциями и операторами
- Возможность использования с ограничениями целостности
- Поддержка индексами
- Приведение к другим типам
- Переносимость
- Возможность расширения размерности

1 - 12 SQL

Выбор типа данных

При планировании использования типов данных нужно принимать во внимание с какими типами работает бизнес, который обслуживает БД, а не с какими типами работают внутренности программ. Можно придерживаться принципа проектирования KISS (keep it short and simple) и отдавать предпочтение простым типам данных или тем, с которыми проще работать и БД их способна эффективно обрабатывать. По этой причине создавать собственные объектные типы, не создавая поддержки (функций, операторов, индексов) не стоит.

Однако, те типы данных, которые поставляются разработчиком БД или её расширениями, могут успешно использоваться. При принятии решения об использовании нужно обращать внимание на эффективность (скорость) обработки данных и простоту интерфейсов работы. Например, хранение и выдачу бинарных данных большого размера и произвольный доступ к ним более эффективно могут обеспечить файловые системы, в том числе, распределённые. В БД могут существовать типы, имеющие программную поддержку с низкой производительностью, например, вложенные таблицы в БД Oracle (вместо них проще использовать обычные таблицы).

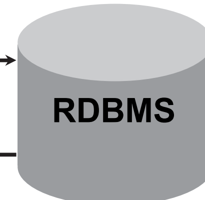
Типы данных, имеющие структуру: геоинфрмационные, XML, JSON могут успешно обрабатываться БД. Для работы с такими типами, часто, используются функции, специфические для конкретного производителя. На больших объемах, более эффективно хранить такие данные на большом количестве хостов, которые могут выполнять несложную предварительную обработку данных и затем через внешние интерфейсы передавать реляционным БД. Такая модель используется в Apache Hadoop HDFS.

Работа с БД с помощью SQL

Ввод команды SQL

```
SELECT department_name
FROM   departments;
```

Команда передается базе данных



DEPARTMENT_NAME
Administration
Marketing
Shipping
IT
Sales
Executive
Accounting
Contracting

Работа с БД с помощью SQL

Доступ к данным в реляционной базе данных осуществляется командами языка *SQL* (*structured query language*, *язык структурированных запросов*) , соответствующего стандарту по операциям с реляционными базами данных. С помощью команд SQL можно выбирать данные, модифицировать данные, менять структуры, в которых хранятся данные.
Реляционные базы данных могут реализовывать стандарты языка SQL: ANSI, ISO - SQL89, SQL-92, SQL:1999 , SQL:2003, SQL:2008, SQL:2011.
Последняя версия стандарта SQL:2016 (ISO/IEC 9075:2016). Каждая следующая версия заменяет предыдущую. SQL-92 определяет три уровня функциональной совместимости: начальный, промежуточный и полный. Большинство СУБД заявляют о совместимости со стандартом SQL только на начальном уровне, так как полный набор возможностей на промежуточном и полном уровнях либо слишком велик, либо конфликтует с ранее реализованным в них поведением.
Начиная с SQL:1999, вместо трёх уровней определено множество отдельных функциональных возможностей. Большое его подмножество представляет "Основную" функциональность, которую должны обеспечивать все совместимые с SQL реализации. Поддержка остальных возможностей не является обязательной.
Начиная с версии SQL:2003, также разделяется на несколько частей. Каждая такая часть имеет короткое имя и номер. ISO/IEC 9075-2 Основа (SQL/Foundation), ISO/IEC 9075-11 Схемы информации и определений (SQL/Schemata), ISO/IEC 9075-1 Структура (SQL/Framework), ISO/IEC 9075-13 Программы и типы, использующие язык Java (SQL/JRT).
Многие производители БД обеспечить совместимость с последней официальной версией стандарта, оставаясь при этом в рамках традиционной. БД Oracle и БД postgres реализуют большую часть требуемой стандартами функционала, хотя иногда с немного другими функциями или синтаксисом, а также добавляют к стандартам свои расширения.

Язык структурированных запросов (SQL)

- Стандартный язык взаимодействия с реляционной БД, позволяющий получать и изменять информацию базы данных
- Позволяет выполнять:
  - запрос данных
  - вставка, изменение и удаление строк таблицы
  - создание, замещение, изменение и удаление объектов
  - контроль доступа к базе данных и ее объектам
  - обеспечение непротиворечивости и целостности базы данных

1 - 14 SQL

Язык структурированных запросов

Язык структурированных запросов (Structured Query Language – SQL) - формальный непроцедурный язык программирования, применяемый для создания, модификации и управления данными в реляционных базах данных.

Включает набор команд, с помощью которых все программы и пользователи осуществляют доступ к информации базы данных.

Реляционные базы данных принимают только команды на языке SQL и дополнительные команды, специфичные для конкретной базы данных.

Дополнительные команды, обычно, используются для управления самой базой данных и её внутренними структурами хранения.

Был создан в 70х годах и назывался SEQUEL, позже переименован в SQL. В 1986г. был принят первый стандарт языка ANSI SQL-86.

Команды SQL

Команды SQL

Команды	Описание
SELECT WITH	Выборка данных из БД
INSERT UPDATE DELETE MERGE	Ввод новых строк, изменение существующих и удаление ненужных строк из таблиц базы данных. В совокупности называются *Языком манипулирования данными* (DML).
CREATE ALTER DROP RENAME TRUNCATE COMMENT	Эти команды создают, изменяют и удаляют структуры данных. В совокупности называются *Языком определения данных* (DDL).
GRANT REVOKE	Предоставляет или изымает права доступа как к базе данных, так и к структурам в ней. В совокупности называются *Языком управления данными* (DCL).
COMMIT ROLLBACK SAVEPOINT	Управляют изменениями, производимыми с помощью команд DML. Изменения можно группировать в логические транзакции.

Доступ к БД

Для обращения к БД могут использоваться:

- **Приложения на языке Java (JDeveloper) – интерфейс JDBC**
- **Исполняемые приложения (.exe) – динамически линкуемые библиотеки поставляемые производителем БД**

Пример исполняемого приложения:

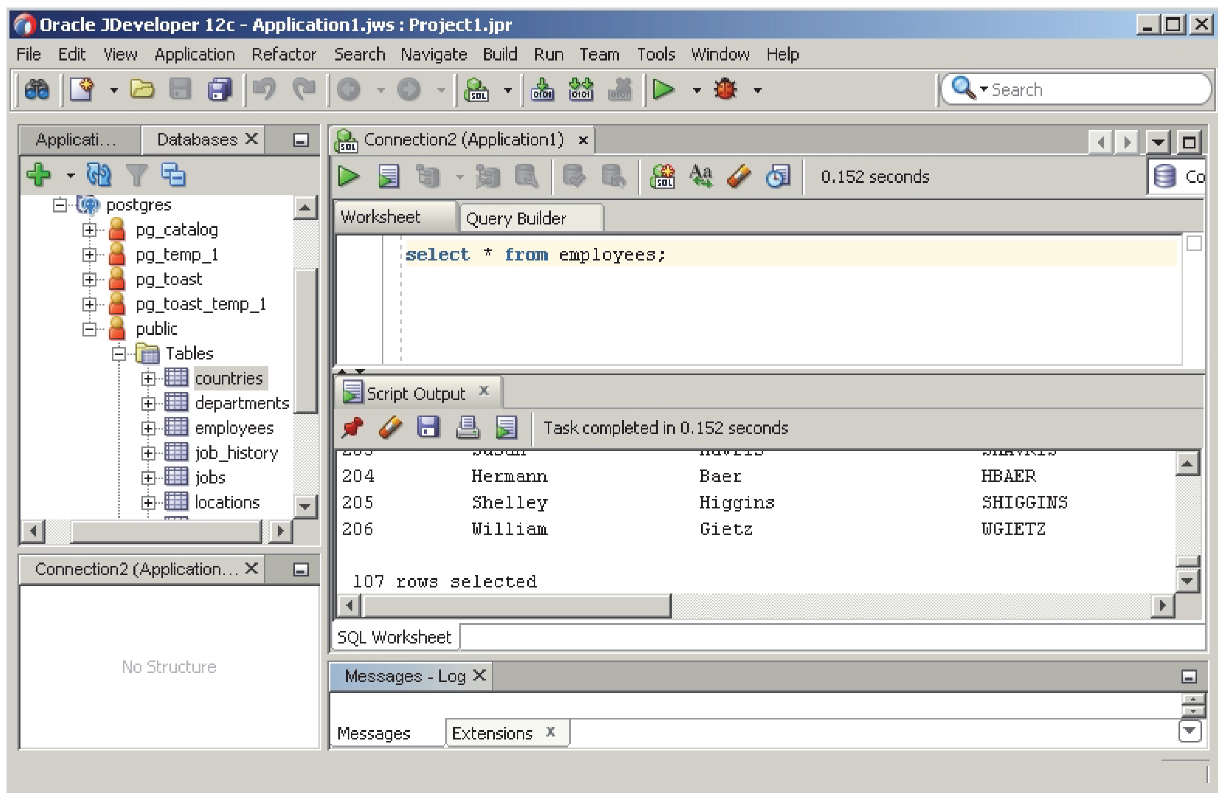
- **Утилиты командной строки (Oracle – sqlplus или sqlcl, postgres - psql)**

Использование SQL

Многие приложения и операционные системы имеют текстовые консоли управления. Для БД Oracle это утилита sqlcl и sqlplus. Для БД postgres это psql. Эти утилиты используются в интерактивном и пакетном режиме.
В интерактивном режиме в утилите вводится команда SQL или команды управления, специфичные для БД конкретного производителя. В пакетном режиме команды берутся из текстового файла, который часто называют По умолчанию расширение файла sql. Например, “create.sql”. Пакетные файлы используются администраторами БД для выполнения часто повторяющихся задач и разработчиками. Например, разработчики могут использовать скрипты для создания: наборов объектов в БД при инсталляции их приложения; изменения объектов в БД при выпуске обновлений приложения.

К БД можно подсоединяться по сети с использованием сетевых протоколов. На транспортном уровне большинство БД используют протокол TCP/IP. БД представляет собой программу, запускаемую на узле сервера. Программа использует процесс или набор процессов в операционной системе. Один или несколько процессов прослушивают порт или порты TCP, на которые приходят запросы на соединения от клиентских программ. Также процессы БД часто поддерживают соединения от утилит, запускаемых на том же узле, что и процессы БД без использования сетевых протоколов, с использованием протоколов межпроцессного взаимодействия. Такие соединения называют локальными. При использовании локальных соединений передаются быстрее, можно запускать процессы БД, использовать автоматическую аутентификацию без паролей.

Oracle JDeveloper



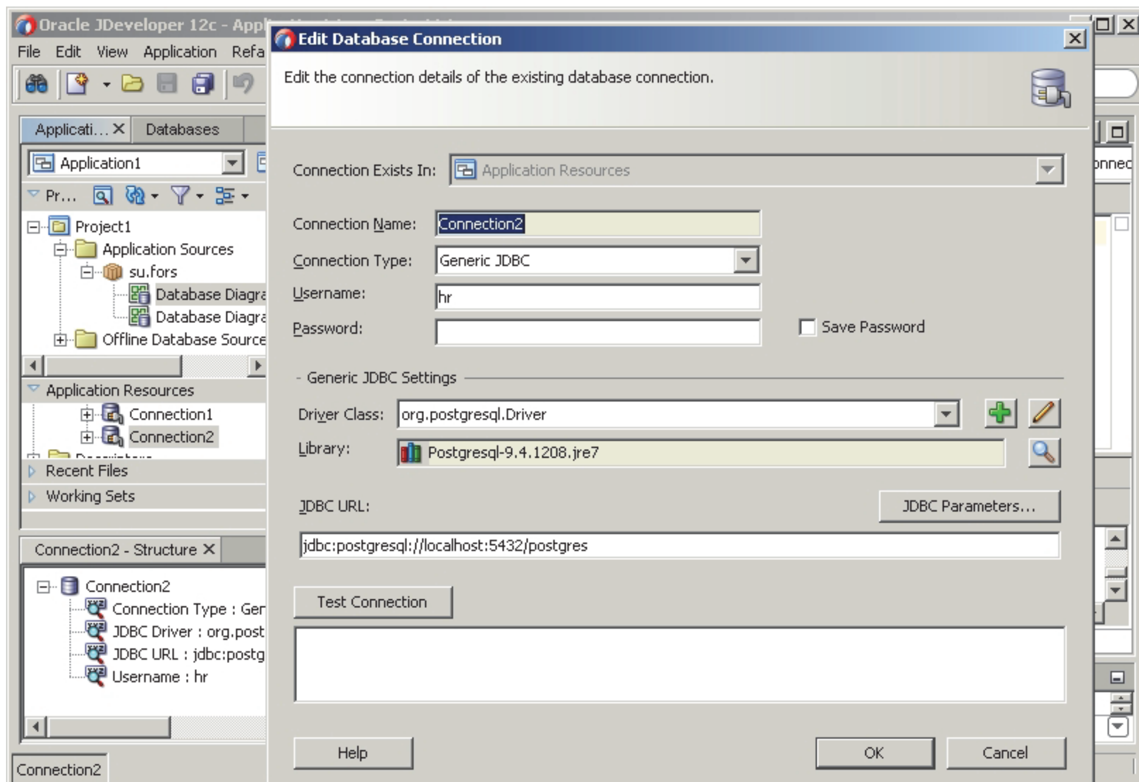
1 - 17 SQL

JDeveloper

Oracle JDeveloper – приложение для визуальной разработки приложений на языке java и смежных технологий. Java-приложения могут использовать БД и в JDeveloper есть инструменты работы с любыми реляционными БД.

Для работы с БД Oracle есть приложение Oracle SQLDeveloper. Функционал JDeveloper и SQLDeveloper схож, но имеются и различия. Например, SQLDeveloper включает функционал для администрирования БД Oracle, а JDeveloper функционал для архитектора и дизайнера приложений.

Навигатор соединений JDeveloper



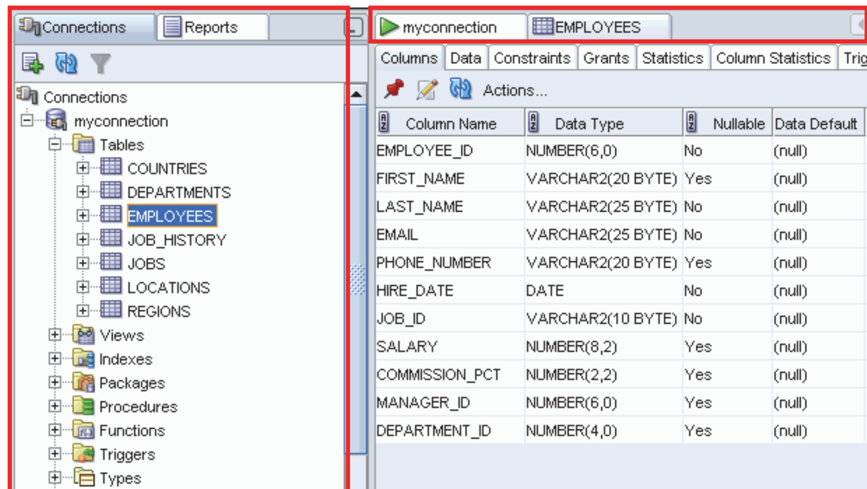
Навигатор соединений

Используя Oracle JDeveloper, можно хранить детали соединения с базой данных. Соединение можно использовать в разработке java-приложений или для работы с БД.

Для создания соединения нужно указать тип БД, имя и пароль для доступа к БД, путь к jar-файлу драйвера (драйвера к некоторым БД поставляются с JDeveloper), класс, инициализирующий соединение из драйвера, JDBC URL в формате драйвера.

Просмотр объектов базы данных

- **Создайте объект подключения к базе данных**
- **Используйте навигатор подключений в следующих целях:**
  - **обзор различных объектов в схеме базы данных;**
  - **просмотр кратких сведений об определениях объектов**



Просмотр объектов базы данных

После создания подключения к базе данных можно выполнять просмотр объектов: таблицы, представления, индексы, пакеты, процедуры, триггеры, типы и др.

Расположенная слева область используется для навигации при поиске и выборе объектов, а в правой области отображаются сведения о выбранных объектах. Можно видеть определение объектов, разбитое по вкладкам с информацией, извлеченной из словаря данных. Например, при выборе таблицы в навигаторе сведения о столбцах, ограничениях, правах, статистике, триггерах и т. д. отображаются на удобной для чтения странице с вкладками.

Чтобы просмотреть определение таблицы EMPLOYEES, как показано на слайде, выполните следующие шаги:

1. Разверните узел подключений в навигаторе подключений.
2. Разверните таблицы.
3. Щелкните EMPLOYEES. По умолчанию будет выбрана вкладка столбцов. На ней показано описание столбцов таблицы. На вкладке данных можно просмотреть данные таблицы, а также ввести новые строки, обновить данные и зафиксировать эти изменения в базе данных.

Использование sqlcl и sqlplus

sqlplus и sqlcl:

- **утилиты командной строки**
- **используются интерактивно или в режиме пакетной обработки**

```
D:\sqldeveloper\sqlcl\bin>sql.bat hr/hr

SQLcl: Release 4.2.0 Production on Tue Feb 28 15:40:29 2017
Connected to:
Oracle Database 12c Enterprise Edition Release 12.1.0.2.0 - 64bit
Production With the Partitioning, OLAP, Advanced Analytics and
Real Application Testing options

SQL> select last_name from employees fetch first 1 row only;

LAST_NAME
-------------------------
King

SQL>
```

1 - 20 SQL

Использование sqlcl и sqlplus

Интерфейсы командной строки sqlplus и sqlcl можно использовать для выполнения команд SQL в целях целях:

ввод, изменение, запуск, хранение, извлечение и сохранение SQL-команд;
форматирование, вычисление, хранение и печать результатов выполнения запроса;
отображение определений столбцов для любой таблицы;
отправка сообщений конечным пользователям и получение ответов;
администрирование базы данных.

Чтобы запустить sqlplus:

Откройте командный интерфейс - терминал для Linux или cmd для Windows.

В командной строке введите команду sqlplus в следующем формате:

```
sqlplus hr/hr@host:1521/orcl
```

Вызов скрипта из sqlcl и sqlplus

script.sql

```
SELECT * from regions where region_id=1;
QUIT
```

Результат

```
sql hr/hr@//localhost:1521/orcl @script.sql
SQLcl: Release 4.2.0 ..
Connected to: Oracle Database 12c..
 REGION_ID REGION_NAME
---------- -------------------------
         1 Europe
Disconnected from Oracle Database 12c..
```


Вызов скрипта из sqlcl и sqlplus

Существующий файл SQL-сценария можно открыть из sqlplus. Это можно сделать в командной строке, предварительно вызвав sqlcl, как показано на слайде. Также это можно сделать из сеанса sqlcl с помощью оператора @. Ниже приведен пример запуска сценария из установленного сеанса sqlcl:

SQL> @script.sql

Примечания

1) Последней командой скрипта должна быть команда sqlcl QUIT.

2) Если скрипт запускается из другого скрипта, то для того, чтобы он запускался из директории родительского скрипта используется двойной символ @@

3) Вместо @ можно использовать команду "start script.sql", но так как в стандарте языка SQL есть команда start transaction, то для исключения неоднозначности использовать команду sqlcl start нежелательно.

4) Расширением по умолчанию для файлов сценариев является .sql Сценарии с таким расширением можно выполнять, не указывая расширение файла, как в следующем примере:

SQL> @script

Выборка данных командой SELECT

Базовая команда SELECT

```
SELECT что
FROM   откуда;
```

SELECT указывает, что нужно выбрать: через запятую "," перечисляются названия столбцов, выражения, литералы, имена функций и т.п.

FROM указывает откуда выбрать: через запятую таблицы, представления и т.п. данные возвращаются в табличном виде: набор строк и столбцов выборки

2 - 2 SQL

Базовая команда SELECT

В БД postgres конструкция FROM опциональна, в **БД Oracle** обязательна. Такой SELECT используется для получения результатов функций или литералов.

Пример:

```
SELECT 'literal' alias1;
alias1
-------
literal
```

В БД Oracle для этих целей используется таблица DUAL:

```
SELECT 'literal' alias1 from dual;
```

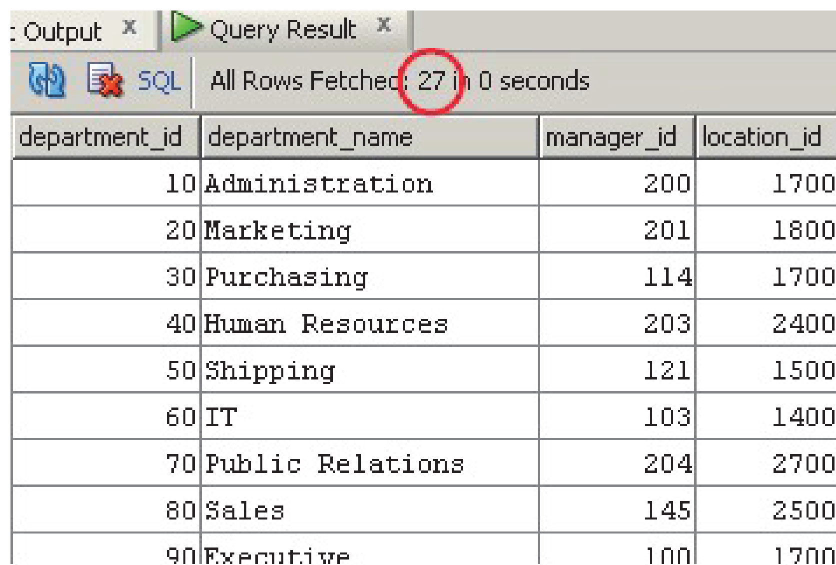
Таблица DUAL

Эта таблица содержит одну стрку.

```
SQL> desc dual;
Name  Null? Type
----- ----- -----------
DUMMY       VARCHAR2(1)
```

Выборка всех столбцов

```
SELECT *
FROM   departments;
```



department_id	department_name	manager_id	location_id
10	Administration	200	1700
20	Marketing	201	1800
30	Purchasing	114	1700
40	Human Resources	203	2400
50	Shipping	121	1500
60	IT	103	1400
70	Public Relations	204	2700
80	Sales	145	2500
90	Executive	100	1700

Выборка всех столбцов и строк

Для вывода значений *всех* столбцов таблицы достаточно поставить звездочку * после ключевого слова SELECT. В примере на слайде таблица отделов содержит четыре столбца: DEPARTMENT_ID, DEPARTMENT_NAME, MANAGER_ID и LOCATION_ID. Количество строк – 27, по одной на каждый отдел.

Также можно перечислить названия всех столбцов. Пример:

```
SELECT  department_id, department_name, manager_id, location_id
FROM    departments;
```

Нужно помнить, что таблицы могут меняться. Например, можно добавлять или удалять столбцы в таблице после её создания, при обновлении приложения. Чтобы приложение не зависело от этого, можно использовать перечисление названий столбцов вместо звездочки.

Выборка не всех столбцов

```
SELECT department_id, location_id
FROM   departments;
```

DEPARTMENT_ID	LOCATION_ID
10	1700
20	1800
50	1500
60	1400
80	2500
90	1700
110	1700
190	1700

Выборка части столбцов

С помощью команды SELECT можно вывести конкретные столбцы таблицы, указав их имена через запятые.

Нужные столбцы указываются в той последовательности, в которой они должны выдаваться.

В postgres можно комбинировать * с выражениями, но в этом обычно нет необходимости:

```
SELECT *, 'literal' alias1 FROM departments
department_id department_name  manager_id  location_id alias1
------------- ---------------- ----------- ----------- -------
40            Human Resources  203         2400        literal
50            Shipping         121         1500        literal
...
```

Выборка части строк

- Стандартное выражение SQL:2011
- Вывести с 4 по 5 строки из выборки

```
SELECT department_id, location_id
FROM   departments
OFFSET 3 ROWS FETCH FIRST 2 ROWS ONLY;
```

2 - 5 SQL

Выборка части строк

LIMIT используется для ограничения количества выбираемых строк. OFFSET опционален и указывает количество пропускаемых строк.

При использовании OFFSET с ORDER BY БД должна выполнять выборку полностью, сортировать и только потом пропускать часть строк и выдавать нужные.

Если использовать OFFSET с повторными командами SELECT, то если повторные SELECT не выдают данные на момент первого SELECT (что бывает на уровне изоляции транзакций по умолчанию), то в таблицу могут быть вставлены или удалены строки между моментами начала выполнения первого и последующего SELECT и последующий SELECT вернёт строки со сдвигом.

Написание команд SQL

- Команды SQL не различают регистры символов
- В sqlcl и psql команда SQL должна заканчиваться точкой с запятой (;) без ее указания даже при нажатии клавиши ENTER продолжается набор команды

2 - 6 SQL

Написание команд SQL

- Команды SQL по умолчанию не различают регистры символов
- Команды SQL могут занимать одну или несколько строк.
- Ключевые слова нельзя сокращать.
- Для упрощения чтения и редактирования предложения обычно пишутся на отдельных строках.
- Для упрощения чтения команды можно использовать табуляцию и отступы.
- Ключевые слова для удобства чтения можно вводить заглавными буквами, а все остальные (например, имена таблиц и столбцов) – строчными.
- Выдают данные и принимают в виде текста, поэтому для передачи значений для нетекстовых типов данных (даты и время) нужно задавать формат преобразования (в команде, на уровне сессии, по молчанию для клиентской программы)

В клиентских приложениях (программы на языках java и др.):

- Структура команд SQL такая же
- “**;**” в конце команды не требуется
- для передачи значений в команды используются символы “**?**” или “**:имя**”. Например SELECT * FROM employees WHERE employee_id = ? AND job_id = ?
- Для передачи значений для нетекстовых типов данных (даты и время) задание их формата не требуется. Данные передаются из и в объект приложения.

Арифметические операторы

Создаются из численных (NUMERIC) типов данных с помощью арифметических операторов

Оператор	Описание
+	Сложение
-	Вычитание
*	Умножение
/	Деление

Арифметические операторы

Арифметическое выражение может включать имена столбцов, числовые константы и арифметические операторы.

Арифметические операторы

На слайде указаны арифметические операторы SQL. Все они могут использоваться в любых предложениях команды SQL, за исключением предложения FROM.

С типами данных DATE и TIMESTAMP можно использовать только операторы сложения и вычитания, с типами временных интервалов (INTERVAL) - все.

Оператор сложения не работает со строками, в отличие от языка java, где он означает конкатенацию.

Арифметические операции

```
SELECT last_name, salary, salary + 300
FROM   employees;
```

LAST_NAME	SALARY	SALARY+300
King	24000	24300
Kochhar	17000	17300
De Haan	17000	17300
Hunold	9000	9300
Ernst	6000	6300

...

```
SELECT last_name, salary, 12*(salary+100)
FROM   employees;
```

LAST_NAME	SALARY	12*(SALARY+100)
King	24000	289200
Kochhar	17000	205200
De Haan	17000	205200

...

Использование арифметических операторов

В примере на слайде оператор сложения используется для прибавления 300 долларов к окладу всех служащих; новый оклад выводится в столбце SALARY+300.

Столбец SALARY+300 не является новым столбцом таблицы EMPLOYEES, а используется только для вывода результатов вычислений. По умолчанию имя этого столбца соответствует произведенным вычислениям – в данном случае "salary+300".

Приоритеты операторов

Если арифметическое выражение содержит более одного оператора, сначала выполняются умножение и деление. Если операторы в выражении имеют одинаковый приоритет, они выполняются слева направо.

Для изменения порядка выполнения оператора можно использовать круглые скобки

Неопределенное значение (NULL)

- Неопределенное значение (NULL) – это значение, которое недоступно, не присвоено, неизвестно или неприменимо.
- Это не ноль и не пробел.

```
SELECT last_name, job_id, salary, commission_pct
FROM   employees;
```

LAST_NAME	JOB_ID	SALARY	COMMISSION_PCT
King	AD_PRES	24000	
Kochhar	AD_VP	17000	
...			
Zlotkey	SA_MAN	10500	.2
Abel	SA_REP	11000	.3
Taylor	SA_REP	8600	.2
...			

2 - 9 SQL

Неопределенные значения

Особенностью языка SQL является использование неопределённых значений NULL. Если в строке отсутствует значение в поле (пересечение строки и столбца на слайде), считается, что столбец содержит *неопределенное* значение (NULL).

Неопределенное значение – это значение, которое недоступно, не присвоено, неизвестно или неприменимо. Это не ноль и не пробел. Ноль – это число, а пробел – символ.

Это понятие было введено для экономии места при физическом хранении данных и является отступлением от реляционной модели. Наличие NULL создаёт сложности. Всегда нужно проверять возможно ли в столбце присутствие неопределённого значения, знать правила выполнения функций и операций (например, сравнения), если аргумент функции или операнд может быть NULL.

Более того, есть существенные различия в работе с NULL в разных БД. В **БД Oracle** конкатенация строки и NULL даст строку, а в **БД postgres** NULL. В **БД Oracle** пустые значения не индексируются, **в БД postgres** индексируются.

Чтобы избежать сложностей, при моделировании таблиц стараются определять, что в столбце не могут храниться пустые значения – определяют ограничение целостности NOT NULL и для удобства устанавливают значение по умолчанию для такого столбца, например, ноль для численного типа.

Неопределенные значения допускаются в столбцах с данными любого типа за исключением случаев, когда столбец был создан с ограничением NOT NULL или PRIMARY KEY.

Неопределенное значение и ограничение UNIQUE

Ограничение целостности PRIMARY KEY не допускакт NULL, в том числе в случае в каждом из столбцов, если оно создано по нескольким столбцам.

Ограничение целостности UNIQUE допускает NULL в каждом и во всех столбцах.

Различия в работе с NULL в ограничениях UNIQUE по <u>более чем одному</u> столбцу:

БД Oracle

```
create table t (x varchar2(5), y varchar2(5));
CREATE UNIQUE INDEX t_uk ON t(x,y);
ALTER TABLE t ADD CONSTRAINT t_uk UNIQUE (x,y) USING INDEX
t_uk;
insert into t values (NULL, NULL);
insert into t values (NULL, NULL);
insert into t values (NULL, 'A');
insert into t values (NULL, 'A');
ORA-00001: unique constraint (HR.T_UK) violated
insert into t values ('A', NULL);
insert into t values ('A', NULL);
ORA-00001: unique constraint (HR.T_UK) violated
select count(*) from t;
COUNT(*)
----------
         4
```

Можно создать частичный уникальный индекс, основанный на функции, который проиндексирует только строки, содержащие NULL. Например,

```
create unique index t_f on t(coalesce(y, '')) where y
is null;
```

допустит только одну строку со значением NULL в столбце y.

Неопределенные значения в арифметических выражениях

Результат вычисления выражения, содержащего неопределенное значение, также будет неопределенным.

```
SELECT last_name, 12*salary*commission_pct
FROM   employees;
```

LAST_NAME	12*SALARY*COMMISSION_PCT
King	
Kochhar	

…

Неопределенные значения в арифметических выражениях

Если какой-либо столбец в арифметическом выражении содержит неопределенное значение `NULL`, результат вычисления также будет `NULL`. Например, попытка деления на ноль заканчивается ошибкой. Но если попытаться разделить число на неопределенное значение, результатом будет неопределенное значение.

В примере на слайде служащий `KING` не является продавцом и не получает комиссионных. Т.к. столбец `COMMISSION_PCT` в арифметическом выражении содержит неопределенное значение, результатом также будет неопределенное значение.

Определение псевдонимов столбцов

Псевдонимы столбцов:

- **Переименовывают заголовки столбцов**
- **Удобны при вычислениях**
- **Следуют сразу за именем столбца; ключевое слово AS между именем столбца и псевдонимом необязательно**
- **Заключаются в двойные кавычки, если содержат пробелы, специальные символы или необходимо различать регистры символов**

2 - 12 SQL

Псевдонимы столбцов

Обычно при выводе результатов запросов использует в качестве заголовка выходного столбца имя выбранного столбца таблицы. Такой заголовок не всегда отражает содержимое столбца и может быть непонятным. Изменить заголовок можно с помощью *псевдонима* столбца (*алиаса*).

Заголовки столбца при работе с БД из программ роли не играют. Однако, псевдонимы столбцов могут использоваться в той же команде SQL для ссылки на этот столбец.

Псевдоним указывается после имени столбца в списке `SELECT` через пробел. По умолчанию псевдонимы выводятся заглавными буквами. Псевдоним, содержащий ключевые слова, пробелы или специальные символы (например, # или $) или различающий регистры символов, заключается в двойные кавычки ("").

Использование псевдонимов столбцов

- Не могут использоваться в WHERE
- AS опционален

```
SELECT last_name AS name, commission_pct comm
FROM   employees;
```

- Используются не только в заголовках при выводе, но и для ссылок в конструкциях SELECT
- Используйте " " чтобы избежать ошибок если алиас окажется зарезервированным словом

```
SELECT last_name "With" , salary*12 "OUT"
FROM   employees;
```

- Наборы зарезервированных слов в разных БД различны.
- При обновлении стандартов могут появляться новые зарезервированные слова

Использование псевдонимов столбцов

В первом примере показан вывод имен и комиссионных всех служащих. Обратите внимание на использование необязательного ключевого слова AS перед псевдонимом столбца. Результат запроса будет один и тот же независимо от того, используется ключевое слово AS или нет.

Второй пример демонстрирует вывод имени и годовой заработной платы всех служащих. Т.к. заголовок With зарезервированное слово, оно не может использоваться в качестве псевдонима без кавычек. OUT - зарезервированное слово в БД postgres.

Наборы зарезервированных слов в разных БД различны. Более того, при обновлении стандартов могут появляться новые зарезервированные слова (With) и старый код перестанет работать.

Оператор конкатенации || и CONCAT

- Соединяет столбцы или символьные строки с другими столбцами.
- Изображается двумя вертикальными линиями (||).
- Создает столбец с результатом, представляющим символьное выражение

```
SELECT   last_name||job_id AS "Employees"
FROM     employees;
```


Оператор конкатенации || и CONCAT

С помощью оператора конкатенации (||) можно соединять одни столбцы с другими столбцами, арифметическими выражениями или константами для создания символьных выражений. Столбцы, указанные с обеих сторон оператора, соединяются и образуют единый выходной столбец.

В примере соединяются столбцы `LAST_NAME` и `JOB_ID` ; они получают псевдоним `Employees`. Следует отметить, что соединенные столбцы (должность и имя служащего) образуют единый выходной столбец.

Неопределенные значения в операторе конкатенации

В БД Oracle результатом соединения неопределенного значения и символьной строки будет символьная строка. Например операция `LAST_NAME || NULL` дает в результате значение `LAST_NAME`.

Вместо || можно использовать функцию CONCAT(a,b). Если один из аргументов не NULL и не пустая строка, то остальные NULL заменяются пустой строкой (игнорируются). В **БД Oracle** эта функция принимает только два аргумента. В БД Postgres один и более аргументов, что удобно.

Символьные литералы

- **Литерал – это символ, число или дата, включенные в `SELECT` список.**
- **Даты и символьные литералы должны быть заключены в апострофы.**
- **Каждая символьная строка выводится один раз для каждой возвращаемой строки таблицы.**

Символьные литералы

Литерал (*literal*) – это любой символ, число или дата, включенные в `SELECT` список и не являющиеся именем или псевдонимом столбца. Литерал выводится для каждой возвращаемой строки. Литеральные строки в виде свободного текста можно включать в результаты запросов; они обрабатываются так же, как столбец из списка `SELECT`.

Символьные литералы и литералы с датами *должны* быть заключены в одиночные апострофы (‘ ‘); цифровые литералы в апострофы не заключаются.

Использование символьных литералов

```
SELECT last_name ||' is a '||job_id
       AS "Employee Details"
FROM   employees;
```

Employee Details
King is a AD_PRES
Kochhar is a AD_VP
De Haan is a AD_VP
Hunold is a IT_PROG
Ernst is a IT_PROG
Lorentz is a IT_PROG
Mourgos is a ST_MAN
Rajs is a ST_CLERK

...

20 rows selected.

Использование символьных литералов

На слайде показан вывод фамилий и должностей всех служащих. Столбец называется Employee Details. Обратите внимание на пробелы, заключенные в апострофы, в команде SELECT. Эти пробелы облегчают чтение выходных данных.

Апострофы в литералах

- для использования апострофа его нужно задублировать

```
SELECT department_name ||
       ', it''s assigned Manager Id:'
|| manager_id AS "Department and Manager" FROM departments;
```

- Задание собственного ограничителя в БД Oracle

```
SELECT department_name ||
       q'[, it's assigned Manager Id: ]'
|| manager_id AS "Department and Manager" FROM departments;
```

Department and Manager
Administration, it's assigned manager ID: 200
Marketing, it's assigned manager ID: 201
Shipping, it's assigned manager ID: 124

Апострофы в литералах

В случае, когда литерал сам содержит знак апострофа (одиночную кавычку), **нужно просто задублировать знак апострофа**.

Также в **БД Oracle** можно использовать оператор, задающий кавычки (**q**) и указать собственный ограничитель. Этот оператор не стандартный и использовать его не следует.

В качестве ограничителя можно выбрать обычно используемые ограничители (однобайтовые или многобайтовые), а также следующие пары символов: [ ], { }, ( ) или < >.

Дублирование строк

По умолчанию выдаются все строки, включая дубликаты:

```
SELECT department_id
FROM   employees;
```

DEPARTMENT_ID
90
90
90

...

Дублирующиеся строки исключены:

```
SELECT DISTINCT department_id
FROM   employees;
```

DEPARTMENT_ID
10
20
50

...

Дублирование строк

В первом примере на слайде показан вывод всех номеров отделов из таблицы EMPLOYEES. Обратите внимание на то, что номера отделов повторяются.

Для устранения дубликатов в результатах запроса необходимо задать ключевое слово DISTINCT в предложении SELECT **сразу после** ключевого слова SELECT. Во втором примере на слайде таблица EMPLOYEES содержит 107 строк, но номеров отделов в таблице всего 12.

После DISTINCT можно указать несколько столбцов. DISTINCT влияет на все выбранные столбцы, и результат запроса содержит неповторяющиеся комбинации их значений.

```
SELECT   DISTINCT department_id, job_id
FROM     employees;
```

Также можно использовать:

```
SELECT   DISTINCT *
FROM     employees;
```

Вывод структуры таблицы

В sqlplus, sqlcl структуру можно посмотреть командой `DESCRIBE`

```
DESC employees
```

Name	Null?	Type
EMPLOYEE_ID	NOT NULL	NUMBER(6)
FIRST_NAME		VARCHAR2(20)
LAST_NAME	NOT NULL	VARCHAR2(25)
EMAIL	NOT NULL	VARCHAR2(25)
PHONE_NUMBER		VARCHAR2(20)
HIRE_DATE	NOT NULL	DATE
JOB_ID	NOT NULL	VARCHAR2(10)
SALARY		NUMBER(8,2)
COMMISSION_PCT		NUMBER(2,2)
MANAGER_ID		NUMBER(6)
DEPARTMENT_ID		NUMBER(4)

2 - 19 SQL

Вывод структуры таблицы

В примере на слайде выводится информация о структуре таблицы EMPLOYEES.

Результат :

Null?	показывает, должен ли столбец содержать данные; NOT NULL означает, что наличие данных в столбце обязательно;
Type	показывает тип данных для столбца.

Вывод содержимого таблицы в файл

- **Команда**

```
SELECT
 'INSERT INTO departments VALUES
  (' || department_id || ', ''' || department_name ||
   ''', ''' || location_id || ''');'
  AS "Insert Statements Script"
FROM   departments;
```

- **Результат**

```
Insert Statements Script
----------------------------------
INSERT INTO departments VALUES
  (10, 'Administration', '1700');

INSERT INTO departments VALUES
  (20, 'Marketing', '1800');
```


Вывод содержимого таблицы в файл

Иногда полезно иметь файл с командами INSERT INTO VALUES для ввода значений столбцов существующей таблицы.

Чтобы вывести апостроф внутри строки, необходимо подставить перед ним в качестве префикса еще один апостроф. Например, на слайде **''');'** окружающие апострофы заданы для выделения символьного литерала. Второй апостроф выступает в роли префикса для третьего апострофа. В результате выводится один апостроф, за которым следует скобка и точка с запятой **');**

Обратите внимание, что при продолжении литерала на следующей строке в результат добавится символ перехода на новую строку.

БД Oracle

Чтобы убрать в sqlcl заголовки столбцов и установить форматирование нужно использовать команды этой утилиты:

SET HEADING OFF ECHO OFF FEEDBACK OFF PAGESIZE 0 LINESIZE 32767

Для создания файла с результатом использовать команду SPOOL 'd:/script.sql';

Для прекращения вывода в файл SPOOL OFF;

В JDeveloper есть визард, создающий скрипт создания объектов БД Oracle. Для других БД только с командами Insert.

Практическое задание 2

- Выборка данных из разных таблиц
- Описание структуры таблиц
- Выполнение арифметических вычислений и задание имен столбцов

Практическое задание 2

3

Создание и изменение структуры таблиц

Рассматриваемые вопросы

- Объекты базы данных
- Просмотр структуры таблицы
- Типы данных, которые могут использоваться в определениях столбцов
- Создание простой таблицы
- Изменение определений таблиц
- Создание ограничений вместе с таблицей
- Описание использования объектов

Цели урока

В этом уроке вы изучите основы создания, изменения и удаления простых таблиц. Будут представлены типы данных, команды создающие и меняющие объекты.

Кроме того, на этом занятии рассматриваются ограничения, а также сообщения об ошибках, которые выдаются в случае исключительных ситуаций, которые возникают, когда в ходе выполнения команд изменения данных нарушаются ограничения.

Стандартные типы данных

Наиболее распространённые скалярные типы данных:

CHAR(n)	строка фиксированной длины n, дополненная пробелами
VARCHAR(n)	строка переменной длины
NUMERIC(p,s)	**p –** общее количество значимых цифр в числе **; s -** количество цифр справа от десятичной точки
TIMESTAMP(p)	Время и дата, p – количество разрядов долей секунд
DATE	дата

Стандартные типы данных

В БД разных производителей используются разные типы данных. Максимальная размерность и количество используемых байт для хранения значений также могут отличаться. Эти “родные” типы данных могут соответствовать или обладать характеристиками типов данных стандартов SQL.

БД Oracle

Тип данных VARCHAR2(n) – синоним VARCHAR(n), CHARACTER VARYING(n), CHAR VARYING(n). n – максимальной количество байт или символов, которое может быть вставлено. Для БД Oracle это 4000 (начиная с 12 версии может быть 32767 байт). Синонимы используются для того, чтобы команды создания или изменения таблиц соответствовали названиям типов данных стандартов SQL.

CHAR(n) – текстовое поле фиксированной длинны, где n от 1 до 2000 байт.

NUMBER(p,s) имеет синонимы NUMERIC (p,s), FLOAT(p), DECIMAL(n), INTEGER и др.

При использовании NUMBER нужно указывать требуемую точность: NUMBER(p) или NUMBER(p,s). Хотя числа в нем физически занимают столько места сколько нужно для их хранения, он принимает огромные числа, что соответствует другому классу типов: в **БД postgres** - BIGINT, в java BigInteger, что обычно излишне.

Пример вычисления места, занимаемого полями различных типов в **БД Oracle**:

```
CREATE TABLE test (col1 NUMBER(5,2), col2 FLOAT(5), col3 INTEGER);
INSERT INTO test VALUES (1.23, 1.23, 1.23);
select dump(col1), dump(col2), dump(col3) from test;
Typ=2 Len=3: 193,2,24  Typ=2 Len=3: 193,2,21 Typ=2 Len=2: 193,2
```

DATE – тип данных SQL есть во всех БД. Он появился задолго до TIMESTAMP и широко используется. Хранит дату с точностью до дня, но в БД Oracle он хранит еще и время с точность до секунды. В БД Oracle он занимает столько же места, сколько TIMESTAMP(0) – 7 байт.

Также в различных БД поддерживаются и другие типы данных для хранения:

бинарных данных

Текстовых и бинарных данных большой длинны – работа с ними идёт потоками

пространственных

xml

json

При их использовании нужно обращать внимание на то, указан ли тип в стандартах SQL, соответствие типам java, физически занимаемый размер, набор встроенных функций, работающих типом.

CHAR или VARCHAR

- Различные типы данных
- Поведение VARCHAR предсказуемо и одинаково в БД разных производителей
- CHAR не стоит использовать
- CHAR всегда занимает максимальное место

```
CREATE TABLE emp (empc CHAR(6), empv VARCHAR(6));
insert into emp values('aaa', 'aaa');
select * from emp where empc=empv;
В БД Oracle:
no rows selected
В БД Postgres:
empc   empv
------ ------
aaa    aaa
```


CHAR или VARCHAR

При выборе типа столбца лучше использовать VARCHAR, если нет специфических требований. Обычно, CHAR не имеет премуществ по сравнению с VARCHAR, только недостатки.

Значения типа char физически дополняются пробелами до максимума и хранятся, а затем отображаются с пробелами. В БД postgres при сравнении двух значений типа character **дополняющие пробелы считаются незначащими и игнорируются**, в БД Oracle не игнорируются. В БД postgres вставка пустой строки ‘’ в CHAR создаёт строку из пробелов, в БД Oracle вставляет NULL. Для VARCHAR в обоих типах БД вставляется NULL, что более предсказуемо.

```
CREATE TABLE emp (empc CHAR(6),  empv VARCHAR(6));

insert into emp values('aaa', 'aaa');

select * from emp where empc=empv;
```

В БД Oracle:

```
no rows selected
```

В **БД Oracle** при приведении значения character к другому символьному типу дополняющие пробелы не отбрасываются.

Работа с датами в БД Oracle

Тип данных DATE

- **занимает 7 байт в таблице и хранит дату и время с точностью до секунды**
- **По умолчанию дата выдается в формате DD-MON-RR (число- месяц-год).**

Тип данных TIMESTAMP(доли секунды)

- **Занимает 7 или 11 байт**

Тип данных TIMESTAMP(доли секунды) WITH TIMEZONE

- **Занимает 13 байт**

```
SELECT last_name, hire_date
FROM   employees
WHERE  hire_date < '01-FEB-88';
```

LAST_NAME	HIRE_DATE
King	17-JUN-87
Whalen	17-SEP-87

3 - 6 SQL

Работа с датами в БД Oracle

Для типа данных DATE выходной и входной формат DD-MON-RR (число-месяц-год) по умолчанию, диапазон от 1 января 4712 до н.э. до 31 декабря 9999 н.э. Формат даты по умолчанию устанавливается на уровне сессии командой

```
alter session set NLS_DATE_FORMAT='YYYY-MM-DD';
```

В таблице DATE занимает 7 байт, в памяти выравнивается до 8 байт.

Типы данных TIMESTAMP без долей секунды занимает 7 байт, с долями секунды 11 байт, с временной зоной 13 байт.

```
CREATE TABLE t1(c1 TIMESTAMP(0), c2 DATE);
INSERT INTO t1 VALUES (SYSTIMESTAMP, CURRENT_DATE);
SELECT dump(c1),dump(c1)  FROM t1;
DUMP(C1)
-----------------------------------
Typ=180 Len=7: 120,117,3,1,15,15,19
Typ=180 Len=7: 120,117,4,21,14,60,37
```

Соответствие типов данных БД Oracle и PostgreSQL

Тип данных Oracle	Тип данных PostgreSQL
VARCHAR, VARCHAR2, NVARCHAR, NVARCHAR2	VARCHAR или TEXT
CHAR, NCHAR	CHAR
CLOB, LONG	VARCHAR или TEXT
NUMBER	BIGINT, INT, SMALLINT, REAL, DOUBLE PRECISION, NUMERIC
BINARY_INTEGER, BINARY_FLOAT	INTEGER, FLOAT
BLOB, RAW, LONG RAW	BYTEA
DATE	DATE или TIMESTAMP
ROWID	CTID, OID

3 - 7 SQL

Соответствие типов данных БД Oracle и PostgreSQL

ROWID не стоит изначально использовать. Может применяться в унаследованных приложениях БД Oracle. Используется как служебный идентификатор строки, присутствующий в большинстве типов таблиц БД Oracle и указывающий на физическое расположение строки в файлах базы данных. В тех таблицах, где в нарушение реляционной теории, отсутствует первичный ключ приложения могли сохранять идентификатор строки в столбцах таблицы и ссылаться на них. Однако это приводит к проблемам: физический адрес строки, а значит их ROWID могут меняться при выполнении операций поддержки структуры таблиц.

NCHAR, NVARCHAR2 почти не используются. Они позволяют хранить и использовать данные в кодировке Unicode в БД Oracle, если основная кодировка не Unicode.

DATE в БД Oracle хранит время с точностью до секунд

Работа с датами

Для вывода текущей даты и времени используются функции LOCALTIMESTAMP, CURRENT_TIMESTAMP, CURRENT_DATE

- **2017-03-22 14:19:46.686088**
- **2017-03-22 14:19:46.686088**
- **2017-03-22**

- **22-MAR-17 02.21.17.403000000 PM**
- **22-MAR-17 02.21.00.382000000 PM ASIA/RIYADH**
- **22-MAR-17**

Работа с датами

LOCALTIMESTAMP возвращает тип данных TIMESTAMP во временной зоне клиента. TIMESTAMP содержит доли секунд (по умолчанию 6 разрядов долей секунды). Остальные функции возвращают TIMESTAMP с временной зоной.

В приложениях датавременные типы передаются в объект приложения и формат данных роли не играет.

При работе в sqcl psql эти среды выдают и принимают данные в виде текста. Для преобразования датавременных типов данных используется формат. Формат устанавливается на уровне сессии или по умолчанию. В разных БД используются разные форматы по умолчанию, но можно параметрами сессии установить нужный формат.

БД Oracle для DATE использует по умолчанию формат DD-MON-RR.

Объекты базы данных

Объект	Описание
Таблица	Основная единица хранения; состоит из строк и столбцов
Представление	Логически представляет подмножества данных из одной или нескольких таблиц
Последовательность	Генератор числовых значений
Индекс	Увеличивает производительность некоторых запросов
Функция	Используется для вычислений в командах SQL

3 - 9 SQL

Объекты базы данных

БД может содержать разные структуры данных.

- **Таблица**: хранит данные.
- **Представление**: подмножество данных из одной или нескольких таблиц.
- **Последовательность**: генерирует числовые значения.
- **Индекс**: улучшает производительность некоторых запросов.
- **Функция**: Используется для вычислений в командах SQL.

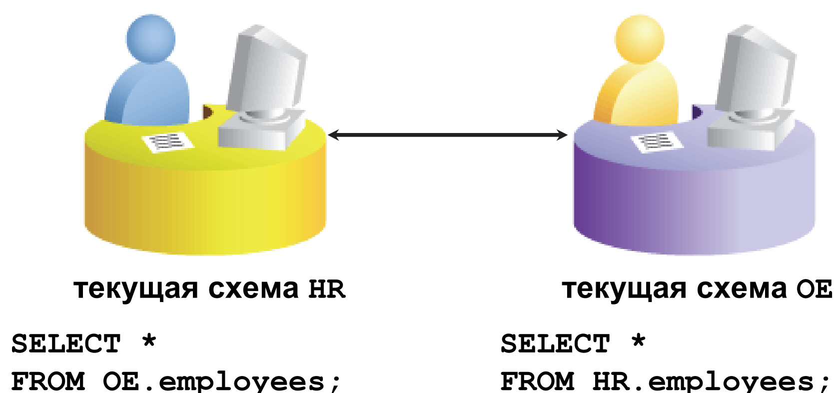
Структура таблиц

- Создавать таблицы можно в любое время, даже когда пользователи используют базу данных.
- Указывать размер таблицы необязательно. В конечном итоге он определяется количеством пространства, выделенного базе данных в целом. Важно, однако, оценить, сколько памяти под таблицу понадобится со временем.
- Структуру таблицы можно изменять и после заполнения её данными.

Примечание: в БД имеются и другие объекты. Например, внешние и временные таблицы, материализованные представления.

Обращения к объектам других схем

- Таблицы в разных схемах могут иметь одинаковые имена
- Для доступа к объектам конкретных схем используется префикс `схема.объект`



текущая схема HR

```
SELECT *
FROM OE.employees;
```

текущая схема OE

```
SELECT *
FROM HR.employees;
```

3 - 10 SQL

Схема (schema) – это пространство имён в БД для объектов. В разных схемах могут находиться объекты с одинаковыми именами.
Схема по умолчанию (текущая схема) - та, в которой осуществляется поиск и создание объектов без указания имени схемы перед именем объекта.

В БД Oracle имя текущей схемы выдает функция

```
sys_context( 'userenv', 'current_schema' )
```

Для смены схемы в сессии используется команда

```
ALTER SESSION SET CURRENT_SCHEMA=newschema;
```

Объекты ищутся только в одной (текущей) схеме. Для доступа к объектам без указания префикса можно создать объект SYNONYM, он нестандартный и его не стоит использовать. Лучше использовать объект VIEW, его использование как интерфейса доступа к данным стандартно и позволяет обновлять приложения без прерывания обслуживания (edition based redefinition).
Перенести объекты в другую схему сложно, так как привилегии и пользователи тесно связаны со схемой.

При написании кода SQL в приложениях лучше не использовать префиксы, так как поддержка схем относится к администрированию и их имена в разных базах данных могут различаться, что скажется на переносимости кода.
Приложения в многозвенной архитектуре (веб-приложения) обычно создают пул сессий с БД из под одного идентификатора безопасности (архитектура one big application user) и путь поиска един для всех сессий. Поэтому проблем с поиском объектов, обычно, нет.

Создание таблиц

- **Создание таблицы**

```
CREATE TABLE dept (
         deptno NUMERIC(4),
         dname       VARCHAR(12),
         loc         VARCHAR(15),
         create_date DATE DEFAULT CURRENT_DATE);
```

- **Детали таблицы в БД Oracle**

```
desc dept
 Name                       Null?    Type
 ----------------------- ----- -----------------
 DEPTNO                          NUMBER(4)
 DNAME                           VARCHAR2(12)
 LOC                             VARCHAR2(15)
 CREATE_DATE                     DATE
```


Создание таблиц

Сложные схемы разрабатывают визуально, например в JDeveloper, после этого среда разработки генерирует скрипт с набором команд, создающих таблицы и другие объекты.

Для тестирования или при повседневной работе с БД администраторы и разработчики могут использовать команды создания и изменения объектов и выполнять их в sqlcl , psql или в средах разработки JDeveloper.

В примере на слайде создается таблица отделов DEPT с четырьмя столбцами: DEPTNO, DNAME, LOC и CREATE_DATE. Для столбца CREATE_DATE задано значение по умолчанию. Если в команде INSERT значение для этого столбца не будет явно указано, тогда автоматически вставится текущая дата.

Датавременные типы столбцов

```
CREATE TABLE tab (tz TIMESTAMP WITH TIME ZONE,
ts TIMESTAMP, iym INTERVAL YEAR TO MONTH);

INSERT INTO tab VALUES
(CURRENT_TIMESTAMP, LOCALTIMESTAMP,INTERVAL '25' MONTH);
INSERT INTO tab VALUES
(LOCALTIMESTAMP, CURRENT_TIMESTAMP, '2-1');

SELECT tz+iym, iym/2, CURRENT_TIMESTAMP-ts from tab;
```


Датавременные типы столбцов

Результат выборки:

БД Oracle операторы возвращают значение в интервале аргументов

20-MAY-19 09.15.45.193000000 PM ASIA/RIYADH	+01-00	+02-01	+00 00:00:05.234000
20-MAY-19 09.15.45.193000000 PM ASIA/RIYADH	+01-00	+02-01	+00 00:00:05.234000

По умолчанию TIMESTAMP создаётся как TIMESTAMP(**6**).

Изменение таблиц

- **Удаление и добавление столбца**

```
ALTER TABLE dept DROP COLUMN dname;
ALTER TABLE dept ADD dname VARCHAR(12);
```

- **Изменение столбца в oracle**

```
ALTER TABLE dept MODIFY dname VARCHAR(15);
```

- **Изменение столбца в postgres**

```
ALTER TABLE dept ALTER dname TYPE VARCHAR(15);
```

- **Переименование столбца**

```
ALTER TABLE dept RENAME COLUMN dname TO ename;
```

- **Переименование таблицы**

```
ALTER TABLE dept RENAME TO depta;
```


Изменение таблиц

После создание объекта БД их можно менять. Свойства, которые можно менять и синтаксис команд может быть различным в разных БД.

Изменения на таблицах, где много строк могут занимать время. На это время таблицы могут по умолчанию блокироваться. В БД существуют команды и опции БД, которые позволяют уменьшить время блокирования при изменении объектов и опций их хранения. Такие опции специфичны для БД конкретного производителя и изучаются администраторами БД.

Опция DEFAULT

- Задает значение по умолчанию, если при вставке данных значение не указано явно.

```
... hire_date DATE DEFAULT CURRENT_DATE, ...

... hire_date TIMESTAMP(0) DEFAULT CURRENT_TIMESTAMP, ...
```

- В качестве значения допускается литерал, выражение или функция SQL.
- Не может использоваться имя другого столбца или псевдостолбец.
- Тип данных, используемый по умолчанию, должен совпадать с типом данных столбца
- Если значение по умолчанию не объявлено явно, им считается значение NULL

3 - 14 SQL

Опция DEFAULT

Если значение по умолчанию не объявлено явно, им считается значение NULL. Опция DEFAULT позволяет задать значение столбца по умолчанию. Это предотвращает запись неопределенного значения в столбец, если строка вставляется без явного указания значения. Значением, которое должно использоваться по умолчанию, может быть литерал, выражение или такая функция SQL, как CURRENT_DATE и USER, но не имя другого столбца или псевдостолбца. Выражение значения по умолчанию должно соответствовать типу данных столбца .

```
CREATE TABLE dept (
    deptno NUMERIC(4) DEFAULT 0,
    dname  VARCHAR(12) DEFAULT USER,
    create_date DATE DEFAULT CURRENT_DATE);
insert into dept (deptno) values (1);
select * from dept;
deptno dname        create_date
------ ------------ -------------------------
1      hr           2017-03-17
```

Размер строки и порядок столбцов

Первыми в таблицы должны идти столбцы:

- **К которым часто обращается приложение**
- **Ключевые столбцы**
- **Столбцы небольшой длины**

Важно оценить средний размер строки в таблице

- **Строки хранятся в блоках**
- **Желательно, чтобы размер большинства строк таблицы был меньше размера блока БД**
- **Если строка не помещается в блок, ее часть переносится в другой блок и скорость работы с этой частью ниже**

Размер строки и порядок столбцов

При создании таблицы перечисляются её столбцы. Порядок указания столбцов важен. Наиболее быстрый доступ к первым столбцам.

Также на эффективность выборки влияет средний размер строки. БД используют размер блока, который задаётся в момент создания БД. Размер блока по умолчанию 8Кб. Максимальный размер блока зависит от производителя БД и, обычно, равен 32Кб. Операционные системы работают (считывают за один вызов программы к функции операционной системы) с дисками и страницами памяти определённого размера, например, 4Кб и размер блока, который использует БД кратен этому размеру.

При планировании количества и максимальной длины столбцов в таблице, стоит учитывать, что строка будет находиться в блоке определённого размера или делиться на несколько частей. Первые столбцы попадают в первый блок и доступ к нему наиболее быстрый.

Можно вместо одной таблицы с сотней столбцов создать несколько и вынести редко используемые столбцы в отдельные таблицы, связав эти таблицы с основной ограничениями целостности. Это часто позволяет уменьшить объемы хранения. Например, в таблице EMPLOYEES вместо названия отдела длиной до 22 символов используется номер отдела, длиной 4 знака и создана таблица DEPARTMENTS, в которую вынесено название отдела. И хотя в выборках, часто требуется название отдела, благодаря меньшему размеру таблицы EMPLOYEES выборки с соединением могут быть более эффективны.

Примечание

Если планируется добавлять столбцы к таблице после её создания и наполнения данными нужно знать, возможно ли добавление нового столбца в нужном месте или столбец будет добавлен последним.

Большие таблицы

На больших объемах принимают во внимание:

- **Набор опций БД, ускоряющих обработку данных**
- **Тип таблицы для их хранения**
- **Параметры таблицы**
- **Типы данных**

3 - 16 SQL

Большие таблицы

Если создать таблицу и наполнить ее данными, то при обращении к ним запросы часто выбирают не все строки, а только часть. Для того, чтобы получить даже часть строк, БД требуется просмотреть все строки и выбрать нужные. Если данных немного, то это выполняется быстро: блоки таблиц, в которых находятся данные кэшируются в памяти сервера и строки считываются быстро. Если данных много и даже если блоки таблиц находятся в памяти, то процесс выборки данных из служебного формата блоков может потребовать значительных ресурсов, так как нужно прочесть все блоки таблицы.

Если БД не сможет обеспечить выборку данных, требуемую приложению, за приемлемое время, то возможность использования БД будет под вопросом. Такая проблема возникает на огромных объемах.

Если планируется хранить и обрабатывать огромные объемы, то оценивают опции, которые есть в БД конкретного производителя. Например, разнесение хранения и обработки данных на несколько узлов: наличие кластеризации, поддержку шардинга, параллельного выполнения команд SQL.

На более низком уровне выбирают типы таблиц, в которых планируется хранить данные т возможность распределить строки в структуре таблицы для быстрого выполнения команд SQL: обычные таблицы, секционированные, кластеризованные, со структурой индекса, хэш-таблицы. Набор опций также различен у БД разных производителей.

Базы данных могут иметь технологии, позволяющие ускорить этот процесс. В БД Oracle есть технология In-memory column store, которая позволяет развертывать данные из блоков таблиц в память сервера в виде, удобном для их сканирования. При использовании этой опции и достаточном объеме памяти на хосте сервера полное сканирование таблиц достаточно быстро. В остальных случаях, более оптимально избежать полного сканирования, что можно достигнуть с помощью индексов. Индексы позволяют быстро получить указатель на строки таблицы в базе данных, в которых находятся нужные данные.

Дальше выбирают типы данных столбцов, возможность разбиения сложных типов на более простые и размещение данных в нескольких таблицах, параметры хранения для таблиц.

Индексы

Индекс:

- **Ускоряет выборки строк**
- **Необходим ограничениям целостности первичный и уникальный ключ**
- **Наиболее распространённый тип B-дерево**
- **Обновляется автоматически при внесении изменений в таблицу**
- **Может удаляться, создаваться, перестраиваться**

3 - 17 SQL

Индексы

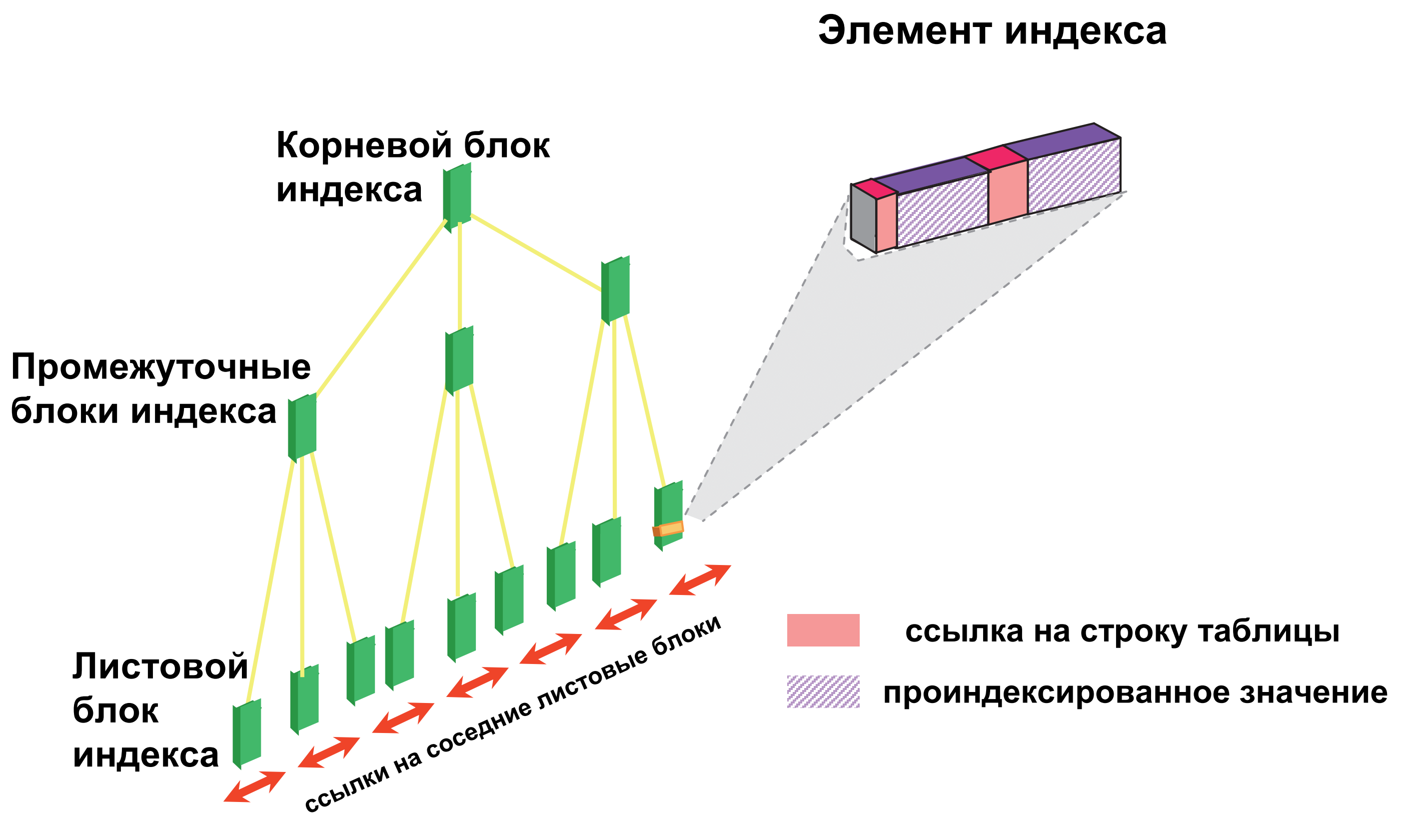
Индекс – это объект, который может ускорить выборку строк за счет использования указателя на строки таблицы. Индекс обеспечивает быстрый доступ к строкам таблицы. Его основное назначение - ускорение доступа к данным.

Если индекс по нужному столбцу отсутствует, поиск осуществляется по всей таблице (full scan). Индекс даёт преимущество только на большом количестве строк, маленькие таблицы кэшируются в памяти сервера БД и full scan в этом случае эффективнее. Более того, даже на огромных объемах, при исполнении редко выполняющихся аналитических запросах с использованием распараллеливания full scan может быть эффективнее использования индексов.

Второе назначение индекса - физическая поддержка ограничений целостности. Когда для таблицы определяется ограничение уникальности или первичный ключ, обязательно использование индекса.

Когда индекс создан, никакие дополнительные действия не требуются: БД сама будет обновлять его при изменении данных в таблице и сама будет использовать его в запросах, где, по её мнению, это будет эффективнее, чем сканирование всей таблицы. При внесении изменений в таблицу автоматически обновляются затрагиваемые индексы. С этим связаны неизбежные накладные расходы при изменении индексов. Таким образом, индексы, которые используются в запросах редко или вообще никогда, должны быть удалены.

Если в таблице часто вставляются и удаляются строки, изменяются значения в проиндексированных столбцах, то физическая структура индекса разрастается и становится менее эффективной. Такие индексы периодически перестраивают. Если есть возможность в архитектуре приложения избегать частого внесения изменений, то лучше это сделать. Например, использовать временные таблицы.

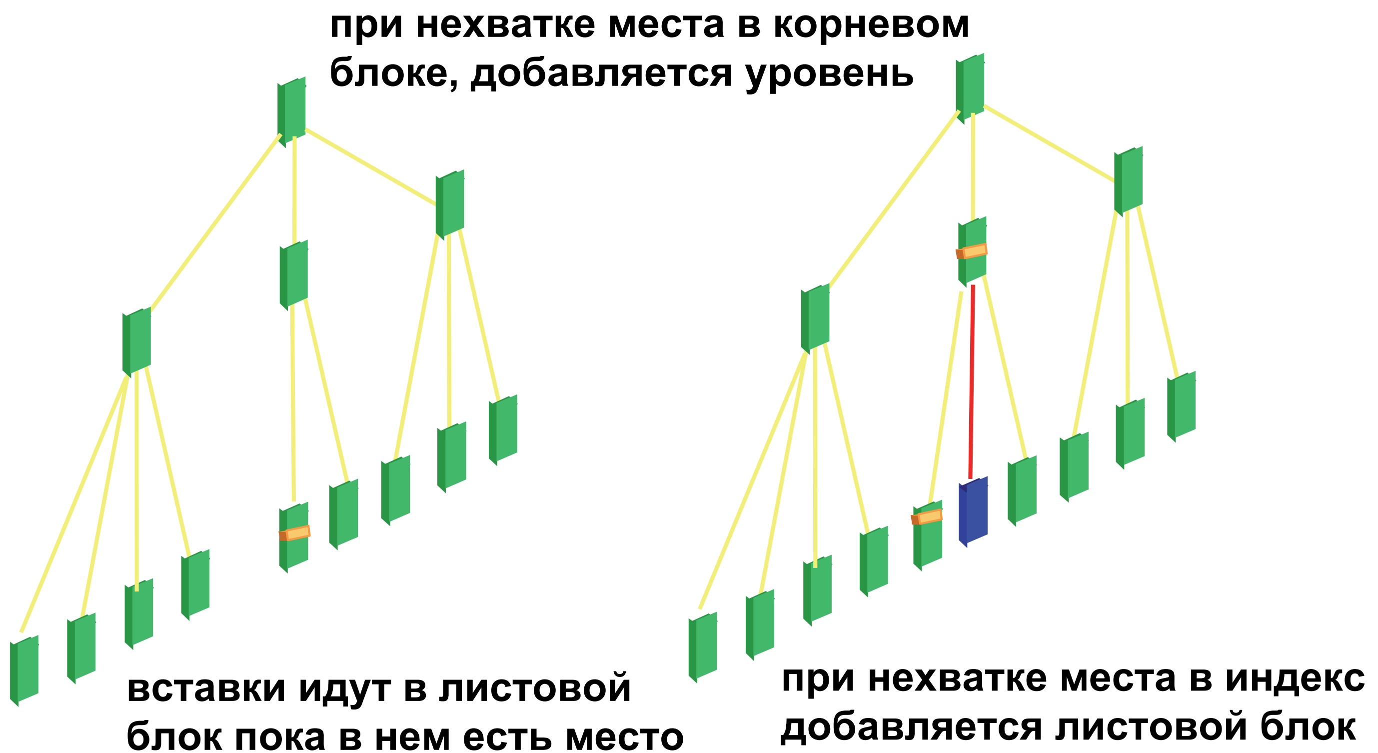


Индекс типа B-дерева

Корень индекса - один блок, который содержит элементы индекса, указывающие на следующий уровень индекса. На следующем уровне находятся промежуточные блоки индекса, в свою очередь указывающие на блоки следующего уровня индекса. Число желтых линий для каждого блока может быть различным. На последнем уровне находятся листовые блоки, содержащие элементы индекса, которые указывают на строки таблицы. Листовые блоки связаны с соседними листовыми для облегчения сканирования индекса как в возрастающем, так и в убывающем порядке значений ("ключ" индекса), которые проиндексированы. Сбалансированное дерево (**Balanced tree** или **B-tree**) означает, что путь от вершины до листового блока одинаков. В листовых блоках хранятся проиндексированное значение и ссылки на строки.

Значением может быть значение в столбце; конкатенация значений нескольких столбцов - если индекс создан на несколько столбцов ("составной индекс"), значение функции, которая принимает в качестве аргументов значение столбца или столбцов (индекс основанный на функции).

Как растёт индекс



Как растёт индекс

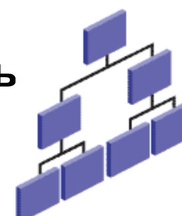
При вставках строк в таблицу обновляются все индексы, которые созданы на таблицу и должны принять новые значения. Чем больше индексов, тем более медленно работает вставка. Именно поэтому не стоит создавать индексов, которые не будут использоваться при выполнении команд.
При вставке в таблицу строки, в листовой блок индекса вставляется пара: значение и указатель на вставленную строку в таблице. Так как индекс хранит значения упорядоченно, то выбирается определённый листовой блок. Если вставить то же самое значение в таблицу ещё раз, то новая пара вставится в тот же листовой блок (или соседние, если в листовых блоках одинаковые значения).
Только если в листовом блоке нет места для вставки, в структуру индекса добавляется ещё один листовой блок, значения перераспределяются между новым (изначально пустым) листовым блоком и тем блоком, в который должна была быть вставлена новая пара. Ссылка на новый листовой блок вставляется в блок вышестоящего уровня и в нём обновляются граничные значения двух листовых блоков, затронутых вставкой. Можно назвать такой процесс "делением" листового блока.
Если в блоке вышестоящего уровня не окажется места для вставки ссылки на новый листовой блок, то он "делится". И так до блока верхнего уровня. Если в блоке верхнего уровня не хватит места, то в индекс добавляется новый уровень.
Обратите внимание, если в таблицу вставляются примерно одинаковые значения, то вставки постоянно идут в один и тот же листовой блок индекса и далее вверх. Постоянно "делится" одна ветвь индекса. Из-за этого растёт число уровней. Если пересоздать индекс, то граничные значения перераспределятся и число уровней может стать меньше.
Удаление строки из таблицы помечает запись в индексе как удалённую, но это не может сразу использоваться.
Обновления строк таблиц эквивалентно удалению в одном листовом блоке и вставке в другой листовой блок, если новое значение не попадает в тот же листовой блок.

Характеристики индекса типа B-дерево

- используются в операторах = > <
- в выражениях LIKE IN BETWEEN EXISTS
- соединениях и других конструкциях SQL
- используются ограничениями целостности
- занимают относительно много места, при внесении изменений в таблицу увеличиваются

Могут быть:

- основанными на функции - индексируют результат функции
- частичными (partial) - не индексируют часть значений или строк таблицы



Характеристики индекса типа B-дерево

Индексы типа B-дерево используются для поиска по равенству; диапазонам; сортировки; ускорения префиксных LIKE-предикатов (вида LIKE 'abc%'); соединений; операций except/intersect; в групповых и оконных функциях.

Не используются в LIKE '%abc' и в выражениях, нечувствительных к регистру (ILIKE ~*).

В БД Oracle в индексе типа B-дерева

- ключевые значения повторяются, если существует несколько строк, имеющих одинаковое ключевое значение; Индекс может быть создан с опцией COMPRESS, которая уменьшает место, занимаемое повторяющимися значениями
- не существует индексных элементов, соответствующих строке, в которой все ключевые столбцы неопределенные (содержат значение NULL). Причина этого: предположение о том, что если данные не инициализированы (пусты), то их, вероятно, много и индексировать эти значения неэффективно. При планировании архитектуры имеет смыл учитывать эту особенность. Например, допускать пустые значения в столбце, если число строк с пустыми значениями значительно (например, больше 70% строк) и в запросах, где не нужны пустые значения указывать условие AND столбец IS NOT NULL.
- Удаление строки из таблицы помечает в листовом блоке индекса ссылку на строку, как удаленную, это место не может использоваться до тех пор, пока все записи в листовом блоке не будут помечены как удаленные или до перестройки индекса (или выполнения над индексом операции COALESCE). Операция обновления поля таблицы помечает в одном листовом блоке индекса ссылку как удаленную и вставляет ссылку в другой листовой блок (если только всё не происходит в одном листовом блоке).

Индексы, основанные на функции или выражении

- Для поиска без учета регистра

```
SELECT * FROM emp WHERE LOWER(ename) = 'smith';
```

- Можно создать индекс по функции

```
CREATE INDEX e_idx ON emp (LOWER(ename));
```

- Для поиска по выражению

```
SELECT * FROM emp
WHERE (first_name || ' ' || last_name) = 'John Smith';
```

- Можно создать индекс по выражению

```
CREATE INDEX e_idx
 ON emp ((first_name || ' ' || last_name));
```


Индексы, основанные на функции или выражении

Для индексирования текстовых столбцов с нечувствительностью, обычно, используют индексы, основанные на функции UPPER() или LOWER() и указывают эту функцию в запросах.

Вместо функции можно использовать выражение.

Используемое при создании индекса выражение или функция должна быть точно такой же, как в командах. БД не может сопоставить схожие функции или выражения.

Индексы типа B-дерево и сортировки

- в листовых блоках данные упорядочены
- листовые блоки связаны
- могут использоваться в сортировках ORDER BY
- поддерживают численные, текстовые, датавременные типы данных
- порядок текстовых данных зависит от collation и регистра
- с символьными типами данных можно использовать индекс основанный на функции, задающей порядок (для регистронезависимой сортировки UPPER, LOWER)
- с группировками GROUP BY
- с окнами PARTITION BY .. OVER

Индексы типа B-дерево и сортировки

Помимо простого поиска строк для выдачи в результате запроса, индексы также могут применяться для сортировки строк в определённом порядке. Это позволяет учесть предложение ORDER BY в запросе, не выполняя сортировку дополнительно.

Для запроса, требующего сканирования большой части таблицы, явная сортировка скорее всего будет быстрее, чем применение индекса, так как при последовательном чтении она потребует меньше операций ввода/вывода. Важный особый случай представляет ORDER BY в сочетании с LIMIT *n*: при явной сортировке системе потребуется обработать все данные, чтобы выбрать первые *n* строк, но при наличии индекса, соответствующего столбцам в ORDER BY, первые *n* строк можно получить сразу, не просматривая остальные вовсе.

По умолчанию элементы B-дерева хранятся в порядке возрастания, при этом значения NULL идут в конце (**в БД Oracle** индексы не хранят NULL). Это означает, что при прямом сканировании индекса по столбцу x порядок оказывается соответствующим указанию ORDER BY x (или точнее, ORDER BY x ASC NULLS LAST). Индекс также может сканироваться в обратную сторону, и тогда порядок соответствует указанию ORDER BY x DESC (или точнее, ORDER BY x DESC NULLS FIRST, так как для ORDER BY DESC подразумевается NULLS FIRST).

Индексы с нестандартными правилами сортировки (ORDER BY DESC) весьма специфичны, но иногда они могут кардинально ускорить определённые запросы. Стоит ли вводить такие индексы, зависит от того, как часто выполняются запросы с необычным порядком сортировки.

Если не используются дисковые массивы, а используются отдельные диски, то лучше, чтобы блоки индекса и таблицы лежали на разных дисках. Для этого можно использовать табличные пространства (объекты администрирования БД).

Создание индекса типа B-дерево

- Создание индекса по одному или нескольким столбцам

```
CREATE INDEX idx ON emp(last_name, first_name)
 TABLESPACE indx_tbs;
CREATE UNIQUE INDEX idxa ON emp(empno);
```

- Создание индекса по функции или выражению

```
CREATE INDEX idxb ON
 emp ((first_name || ' ' || last_name));
```

3 - 23 SQL

Создание индекса типа B-дерево

Индекс по одному или нескольким столбцам создается с помощью команды CREATE INDEX. Порядок столбцов важен. Первый столбец по порядку называется ведущим. Составные индексы следует использовать обдуманно. В большинстве случаев индекс по одному столбцу будет работать достаточно хорошо и сэкономит время и место. Индексы по более чём трём столбцам вряд ли будут полезными.

Индекс по двум столбцам и индекс по конкатенации этих столбцов различны - во втором случае БД считает его индексом, основанным на функции и использует только с выражениями, содержащими конкатенацию.

По умолчанию команда CREATE INDEX создаёт индексы типа B-дерево, эффективные для большинства операторов и типов данных.

B-деревья могут работать в условиях на равенство и в проверках диапазонов с данными. При индексировании текстовых данных нужно принимать во внимание тип сортировки (collation).

В БД Oracle индексы не индексируют строки с пустыми значениями во всех столбцах составного индекса и если в условии выборки не задано AND IS NOT NULL или на столбце отсутствует ограничение целостности NOT NULL, то индекс не используется.

PCTFREE

- процент резервирования места в блоках для последующих изменений
- указывается для индексов и таблиц
- минимально можно резервировать 0%
- по умолчанию для индексов резервируется 10%
- может меняться после создания индекса или таблицы

БД Oracle:

```
CREATE INDEX имя ON emp (id) PCTFREE 10;
```


FILLFACTOR/PCTFREE - процент резервирования места в блоках для последующих изменений. Можно указывать это значение при создании индекса. Его можно поменять после создания индекса, но значение не подействует, если не перестроить индекс. По умолчанию резервируется 10% в **БД Oracle**. Резервирование выполняется в момент создания индекса при наличии данных. После создания процент резервирования роли не играет -индекс обновляется до полного заполнения блоков.
Рекомендуется резервировать мало места для индексов по столбцам, значения которых увеличиваются монотонно, как, например, номера счетов, генерируемые системой. В таких случаях новые элементы индекса всегда добавляются к существующим, и нет необходимости вставлять новый элемент между двумя существующими.
В тех случаях, когда значение индексированного столбца вставляемой строки может принимать любое значение, т.е. новое значение попадает в текущий диапазон значений, необходимо резервировать больше места.

Примечание

FILLFACTOR/PCTFREE также применяется для таблиц. Он задаёт процент резервирования свободного пространства в блоках таблицы под UPDATE строк в блоке, которые приводят к увеличению размера строки. Размер строки может увеличиться, так как большинство типов данных переменной длины. В БД Oracle для таблиц по умолчанию резервируется 10%, в БД postgres 0% (место не резервируется).

Типы данных столбцов и индексы

Индекс:

- **Указывает на физический адрес строки в таблице (блок), к котором находятся данные проиндексированного столбца (столбцов)**
- **Есть разные типы индексов, использующих разные алгоритмы**
- **Поддерживают не все типы данных**
- **Поддерживают не все операторы и функции**
- **Возможность создания или эффективность зависит от максимальной длины столбца**

3 - 25 SQL

Типы данных столбцов и индексы

Индексы могут использовать разные алгоритмы индексирования и поддерживать разные типы данных. При задании типа данных столбцов нужно учитывать:

1) возможность создания индексов различных типов для типа данных столбца

2) выбрать оптимальный максимальный размер для типа столбца (*n* в VARCHAR(*n*)). Например, индекс может не создаться, если максимальная длина столбца превышает какое-то пороговое значение или быть неоптимальным.

3) операторы и функции, которые будут использоваться при работе со столбцом и поддержку этих операций и функций индексами. Например, индекс типа хэш не может использоваться с операторами < >, только =.

4) распределение значений в столбце может повлиять на выбор типа индекса, его параметры и эффективность его использования.

5) как будут меняться данные в столбце - вставляться монотонной последовательностью (1,2,3,4...), часто удаляться и т.п. Это может повлиять на быстроту обновления индексов разных типов, их размер и эффективность.

6) количество строк и размер таблицы

Исходя из этих критериев выбирается оптимальный тип индекса, выбираются типы данных столбцов и, возможно, оптимизируется логика работы приложения.

Размер индексируемых столбцов

- **Указание большого размера столбцов**

```
CREATE TABLE emp
(last_name VARCHAR(4000), first_name VARCHAR(4000));
CREATE INDEX emp_idx ON emp(last_name,first_name);
```

- **в БД Oracle приведёт к невозможности создания индекса**

```
SQL Error: ORA-01450: maximum key length (6398) exceeded
```


Размер индексируемых столбцов

Рассмотрим пример. В таблице нужно хранить фамилию и имя сотрудника. При создании таблицы можно указать разумное ограничение на длину хранимых значений, а можно максимально возможное - 4000. Во втором случае это может привести к невозможности создания индекса. Нельзя создать индекс, если длина индексируемых значений может превысить примерно 70% размера блока табличного пространства, в котором создаётся индекс. Также нельзя создать индекс на столбец типа LONG. Более того, столбец типа LONG может быть только один в таблице и не может использоваться в ограничениях целостности.

В БД poostgres таких ограничений нет и можно проиндексировать поля любой длинны, в том числе типа TEXT, аналогичного типу LONG.

Алгоритмы индексирования

- **B-tree**
- **Bitmap**
- **Hash**

БД Oracle

- **IOT - index organized table**
- **доменный**

Алгоритмы индексирования

BITMAP - битовые карты (0 и 1) для каждого возможного значения столбца, где каждому биту соответствует строка с индексируемым значением, а его значение равное 1 означает, что запись, соответствующая позиции бита содержит индексируемое значение для данного столбца или свойства. Сильные стороны: компактность представления (занимает мало места); быстрое чтение и поиск по предикатам AND/OR. **В БД Oracle** это структура постоянного хранения Чтобы объединить несколько индексов, база сканирует каждый необходимый индекс и готовит битовую карту в памяти с расположением строк таблицы. Битовые карты затем обрабатываются AND/OR операцией по мере требования запроса и после этого выбираются столбцы с данными.

HASH - основан на автоматически создаваемой хеш-функции, которая отображает более мощное множество в менее мощное. Может использоваться только с оператором равенство. Для операций > < и диапазонов не применим. в БД Oracle создается как тип хранения таблицы. Редко используется.

В БД есть и другие типы индексов. Например, для индексирования текстовых документов в БД Oracle используются доменные индексы

Перестройка индексов

Индекс был повреждён

Индекс стал «раздутым», то есть в нём оказалось много пустых или почти пустых блоков

Параметр хранения индекса (фактор заполнения) был изменён

БД Oracle:

```
ALTER INDEX имя REBUILD [ONLINE];
```


Перестройка индексов

Перестройки индекса характеризуются следующим образом:

- Новый индекс строится на основе существующего индекса, если он не поврежден и сортировки не требуются.
- Старый индекс удаляется после построения нового. Во время перестройки требуется дополнительное пространство для размещения как старого, так и нового индекса.
- Структура нового индекса, обычно, более эффективна и компактна.
- Во время построения нового индекса запросы могут продолжать использовать таблицу.
- Блокирует перестраиваемые индексы, что блокирует изменение полей таблицы, к которым относится индекс, а также вставки и удаления строк.

Перестройка индекса без блокировки DML операций на таблице

В БД Oracle используется опция `ONLINE`.

Удаление индексов

- Удаляйте индекс перед массовыми загрузками, потом создавайте заново
- Удаляйте редко используемые индексы и создавайте их по мере необходимости

```
DROP INDEX deptartments_idx;
```

3 - 29 SQL

Удаление индексов

Удаление индексов может потребоваться в следующих ситуациях:

- Индекс больше не нужен в приложении.
- Индекс можно удалить перед выполнением загрузок больших массивов данных. Удаление индекса перед выполнением загрузок больших массивов данных и последующее его повторное создание:
  - улучшает производительность загрузки;
  - дает возможность более эффективного использования индексного пространства.
- Индексы, используемые периодически, не нуждаются в излишнем сопровождении, особенно если они основаны на редко используемых таблицах. Например, только конце года или квартала генерируются произвольные запросы для построения отчетов.
- Индекс поврежден.

Индексы, используемые ограничениями, не могут быть удалены. Поэтому соответствующие ограничения должны быть сначала удалены.

Ограничения целостности

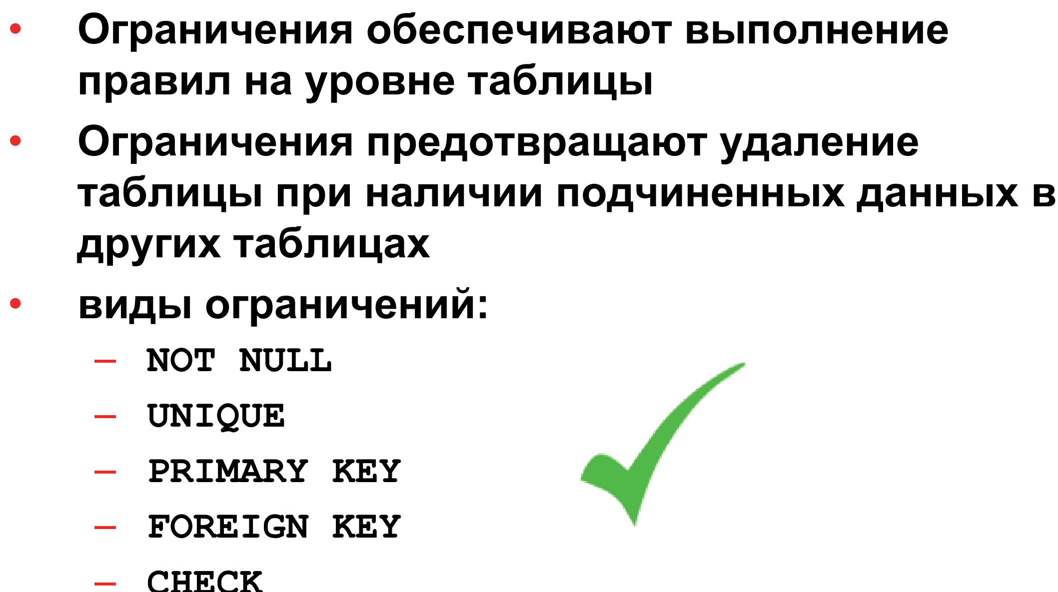
- **Устанавливаются на уровне столбца или таблицы**
- **Создаются:**
  - **при создании таблицы**
  - **после создания таблицы**
  - **могут удаляться, заменяться**
- **Могут проверять данные немедленно или при фиксации транзакции**

3 - 30 SQL

Ограничения целостности

Типы данных сами по себе ограничивают множество данных, которые можно сохранить в таблице. Однако для многих приложений такие ограничения слишком грубые. Например, столбец, содержащий цену продукта, должен, вероятно, принимать только положительные значения. Но такого стандартного типа данных нет. Возможно, вы также захотите ограничить данные столбца по отношению к другим столбцам или строкам. Например, в таблице с информацией о товаре должна быть только одна строка с определённым кодом товара.

Для решения подобных задач SQL позволяет вам определять ограничения для столбцов и таблиц. Ограничения дают вам возможность управлять данными в таблицах так, как вы захотите. Если пользователь попытается сохранить в столбце значение, нарушающее ограничения, возникнет ошибка. Ограничения будут действовать, даже если это значение по умолчанию.



Ограничения целостности

БД использует ограничения целостности (constraints), чтобы предотвратить попадание неправильных данных в таблицы и задать связи между таблицами.

Ограничения используются в следующих целях:

- Контроль за соблюдением правил на уровне таблицы при каждой вставке, обновлении или удалении данных. Операция может быть успешной только в случае выполнения заданного условия.
- Запрет удаления таблицы при наличии подчиненных данных в других таблицах.

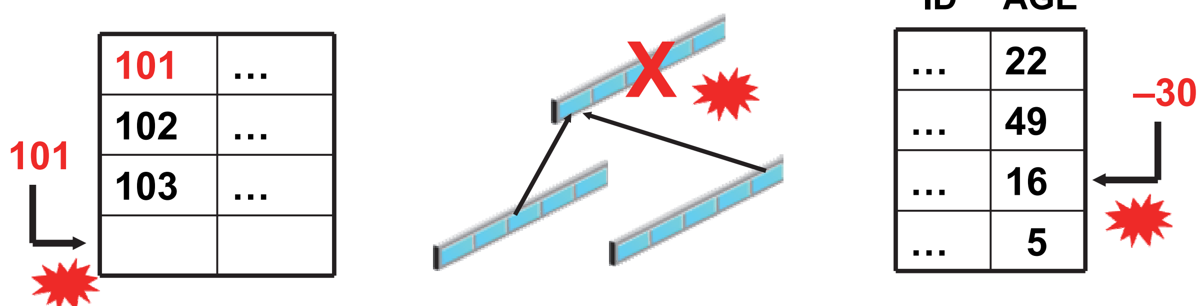
Виды ограничений

Ограничение	Описание
NOT NULL	Означает, что столбец не может содержать неопределенное значение.
UNIQUE	Обозначает столбец или группу столбцов, которые не должны повторяться в строках таблицы.
PRIMARY KEY	Однозначно идентифицирует каждую строку таблицы.
FOREIGN KEY	Создает и контролирует связь по внешнему ключу между столбцом и столбцом таблицы, на которую делается ссылка.
CHECK	Задает бизнес-правило (условие), которое должно выполняться.

Нарушение ограничений целостности

Примеры нарушения ограничений целостности:

- **Вставка дублирующего значения главного ключа**
- **Удаление родительской строки при наличии дочерней строки, связанной ограничением ссылочной целостности**
- **Изменение значения столбца, приводящее к выходу за допустимые границы, заданные ограничением `CHECK`**



Нарушение ограничений целостности

Ограничение нарушается, когда выполняется команда DML, не удовлетворяющая правилам этого ограничения. Существует множество видов нарушения ограничений целостности. Ниже приведены некоторые примеры.

- **Уникальность.** Ограничение срабатывает при попытке внесения дублирующего значения в столбец, который является главным ключом. Ограничение также срабатывает, если для значений столбца создан уникальный индекс.
- **Ссылочная целостность.** Ограничения такого типа нарушаются, когда для каждой дочерней строки отсутствует соответствующая родительская строка.
- **Ограничение `Check`** срабатывает при попытке сохранения в столбце значения, не удовлетворяющего правилам, определенным для этого столбца. Например, для столбца `AGE` (возраст) может быть задано ограничения, в соответствие с которым значения должны быть только положительными числами.

Ограничение NOT NULL

Предотвращает появление неопределенных значений в столбце

EMPLOYEE_ID	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	DEPARTMENT_ID
100	King	SKING	515.123.4567	17-JUN-87	AD_PRES	24000	90
101	Kochhar	NKOCHHAR	515.123.4568	21-SEP-89	AD_VP	17000	90
102	De Haan	LDEHAAN	515.123.4569	13-JAN-93	AD_VP	17000	90
103	Hunold	AHUNOLD	590.423.4567	03-JAN-90	IT_PROG	9000	60
104	Ernst	BERNST	590.423.4568	21-MAY-91	IT_PROG	6000	60
178	Grant	KGRANT	011.44.1644.429263	24-MAY-99	SA_REP	7000	
200	Whalen	JWHALEN	515.123.4444	17-SEP-87	AD_ASST	4400	10

...

20 rows selected.

Ограничение NOT NULL (ни одна строка не может содержать неопределенное значение в этом столбце)

Ограничение NOT NULL

Отсутствие ограничения NOT NULL (любая строка может содержать неопределенное значение в этом столбце)

3 - 33 SQL

Ограничение NOT NULL

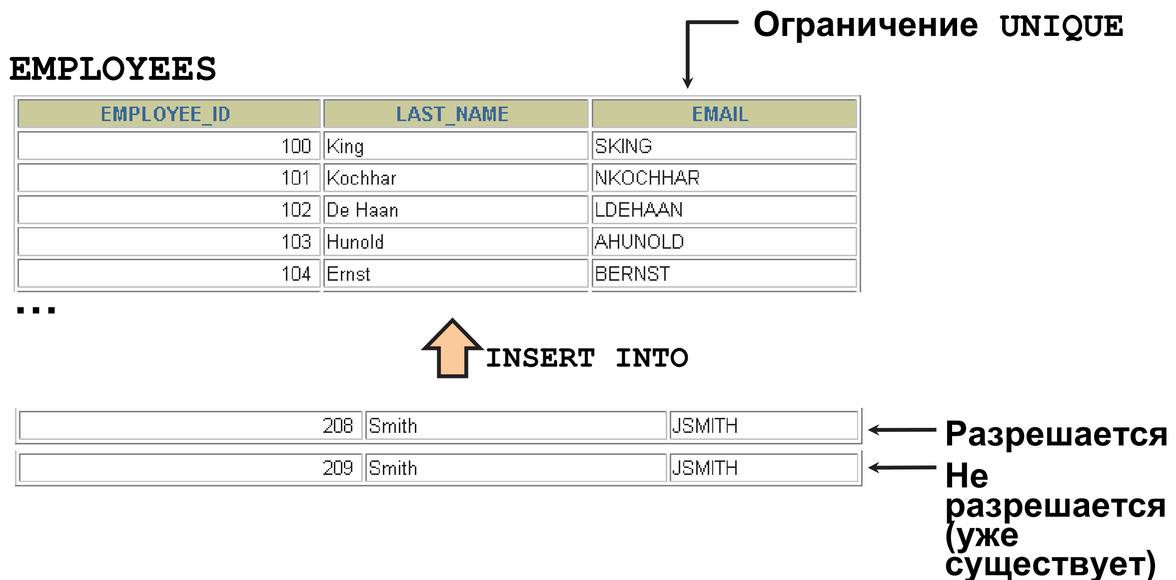
Ограничение NOT NULL означает, что столбец не может содержать неопределенное значение. Столбцы, не имеющие ограничения NOT NULL , могут содержать неопределенные значения по умолчанию.

Ограничение NOT NULL должно определяться на уровне столбца.

После создания таблицы и наполнения ее данными можно удалять и добавлять ограничение.

В БД Oracle:

```
ALTER TABLE dept MODIFY dname NOT NULL;
ALTER TABLE dept MODIFY dname NULL;
```



Ограничение UNIQUE

Ограничение UNIQUE означает, что каждое значение в столбце или наборе столбцов должно быть уникально, т.е. никакие две строки таблицы не должны содержать одинаковые значения в этом столбце или наборе столбцов. Столбец (или набор столбцов), указанный в определении ограничения UNIQUE, называется уникальным ключом (unique key). Если уникальный ключ состоит из нескольких столбцов, его называют составным уникальным ключом.

Ограничение UNIQUE допускает ввод неопределенных значений в столбцы уникального ключа при условии, что для этих же самых столбцов не задано ограничение NOT NULL. Фактически, в отсутствии ограничения NOT NULL неопределенные значения в столбцах допускаются в любом количестве строк., т.к. считается, что неопределенное значение не равно ничему. Неопределенное значение в столбце (или всех столбцах составного уникального ключа) всегда удовлетворяет требованию ограничения UNIQUE.

Примечание

Если в составном уникальном ключе имеются столбцы, где допускаются неопределенные значения, то определенные значения в остальных столбцах не должны повторяться. Это связано с механизмом поиска для ограничений UNIQUE.

Ограничение UNIQUE

```
CREATE TABLE emp (
    employee_id      NUMERIC(6),
    email            VARCHAR(25),
    CONSTRAINT emp_uk UNIQUE(email) DEFERRABLE);
```


Ограничение UNIQUE

Ограничение UNIQUE можно задать на уровне столбца или таблицы. Составной уникальный ключ определяется на уровне таблицы.

Примечание

Поскольку уникальность требует проверки существующих данных, то чтобы избежать полного сканирования строк таблицы нужен индекс. **БД Oracle** автоматически создаёт индекс со свойством уникальности для столбца или столбцов главного ключа, если не существует индекса, в котором ведущие столбцы соответствуют столбцам ограничения целостности. Лучше создавать индексы до добавления и включения ограничений целостности, так как можно детально задать их характеристики.

Ограничение UNIQUE проверяет уникальность непосредственно в момент добавления или изменения строки. Стандарт SQL говорит, что уникальность должна обеспечиваться только в конце оператора; это различие проявляется, например когда одна команда изменяет множество ключевых значений. Чтобы получить поведение, оговоренное стандартом, можно указать свойство DEFERRABLE INITIALLY IMMEDIATE. Но это может быть значительно медленнее, чем немедленная проверка ограничений. Для ограничений DEFERRABLE нужны неуникальные индексы.

Ограничение PRIMARY KEY

DEPARTMENTS

ГЛАВНЫЙ КЛЮЧ

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
50	Shipping	124	1500
60	IT	103	1400
80	Sales	149	2500

...

Не разрешается (неопределенное значение)

INSERT INTO

	Public Accounting		1400
50	Finance	124	1500

Не разрешается (50 уже существует)

Ограничение PRIMARY KEY

PRIMARY KEY задаёт главный ключ среди уникальных. Главный ключ у таблицы только один. Ограничение PRIMARY KEY – это столбец или набор столбцов, уникально идентифицирующий каждую строку таблицы. Это ограничение обеспечивает уникальность значения столбца или комбинации столбцов, а также контроль за тем, чтобы ни одна из частей составного главного ключа не содержала неопределенное значение.

Это ограничение целостности обладает всем свойствами UNIQUE, а также оно не допускает пустых значений ни в одном из столбцов, на которые оно создано.

Оно автоматически создает ограничение целостности NOT NULL на каждый из своих столбцов.

```
CREATE TABLE emp (eid NUMERIC(6), eida NUMERIC(6),
    CONSTRAINT emp_pk PRIMARY KEY (eid, eida));
insert into emp values(2, null);
ОШИБКА: нулевое значение в столбце "eida" нарушает ограничение NOT NULL

Name Null     Type
---- -------- ---------
EID  NOT NULL NUMBER(6)
EIDA NOT NULL NUMBER(6)
```

Определение ограничений

- Ограничение на уровне столбца:

```
CREATE TABLE emp (
  employee_id  NUMERIC(6)
    CONSTRAINT emp_pk PRIMARY KEY,
  first_name   VARCHAR(20) NOT NULL);
```

1

- Ограничение на уровне таблицы:

```
CREATE TABLE emp (
  employee_id  NUMERIC(6),
  first_name   VARCHAR(20),
  CONSTRAINT emp_pk
    PRIMARY KEY (EMPLOYEE_ID));
```

2

- Удаление и добавление после создания таблицы

```
ALTER TABLE emp DROP CONSTRAINT emp_pk;
ALTER TABLE emp ADD CONSTRAINT emp_pk PRIMARY KEY (EMPLOYEE_ID);
```

3 - 37 SQL

Определение ограничений

Ограничения обычно определяются во время создания таблицы. Их можно добавить после создания таблицы, а также временно выключить.

В двух примерах на слайде создается главный ключ для таблицы EMPLOYEES, содержащий столбец EMPLOYEE_ID.

1. В первом примере используется синтаксис описания ограничения на уровне столбца.
2. Во втором примере ограничение определяется на уровне таблицы.

Разница только в удобстве чтения команды, ограничение создаётся одинаковое.

Возможность добавления ограничений после создания таблицы и наполнения её данными зависит от того удовлетворяют ли существующие данные ограничению.

Ограничение FOREIGN KEY

DEPARTMENTS

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
50	Shipping	124	1500
60	IT	103	1400
80	Sales	149	2500

...

ГЛАВНЫЙ КЛЮЧ

EMPLOYEES

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
100	King	90
101	Kochhar	90
102	De Haan	90
103	Hunold	60
104	Ernst	60
107	Lorentz	60

...

ВНЕШНИЙ КЛЮЧ

INSERT INTO

200	Ford	9
201	Ford	60

Не разрешается (9 не существует)

Разрешается

Ограничение FOREIGN KEY

Ограничение FOREIGN KEY, называемое также ограничением ссылочной целостности, обозначает столбец или комбинацию столбцов как внешний ключ (foreign key) и через этот ключ устанавливает связь между данным столбцом и главным или уникальным ключом этой же самой или другой таблицы.

В примере на слайде столбец DEPARTMENT_ID определен как внешний ключ таблицы EMPLOYEES (зависимой или дочерней). Этот внешний ключ ссылается на столбец DEPARTMENT_ID в родительской (“мастер”) таблице DEPARTMENTS.

Указания

- Значение внешнего ключа должно соответствовать существующему значению первичного ключа в родительской таблице или быть неопределенным (NULL).
- Внешние ключи основаны на значениях данных и являются чисто логическими указателями, а не физическими.

Ограничение FOREIGN KEY

- Ссылается на столбец или столбцы таблицы с уникальным или первичным ключём
- Может ссылаться на столбцы своей таблицы

```
CREATE TABLE emp (
  aa NUMERIC(6),
  bb NUMERIC(6),
  cc NUMERIC(6),
  dd NUMERIC(6),
  CONSTRAINT emp_pk PRIMARY KEY (aa, bb),
  FOREIGN KEY (cc, dd) REFERENCES emp (aa, bb)
   ON DELETE CASCADE DEFERRABLE INITIALLY IMMEDIATE);
```


Ограничение FOREIGN KEY

FOREIGN KEY - Определяет на уровне таблицы столбец в подчиненной таблице, используемый в качестве внешнего ключа.

REFERENCES - Определяет родительскую таблицу и столбец в ней.

Внешний ключ должен ссылаться на столбцы, образующие первичный ключ или ограничение уникальности. Количество и типы столбцов в ограничении должны соответствовать количеству и типам целевых столбцов. Таким образом, для связанных столбцов всегда будет существовать индекс (определённый соответствующим первичным или уникальным ключом), а значит проверки соответствия связанной строки будут выполняться эффективно. Так как команды DELETE для строк главной таблицы или UPDATE для зависимых столбцов потребуют просканировать подчинённую таблицу и найти строки, ссылающиеся на старые значения, полезно будет иметь индекс и для подчинённых столбцов. Но это нужно не всегда, и создать соответствующий индекс можно по-разному, поэтому объявление внешнего ключа не создаёт автоматически индекс по связанным столбцам.

Свойства FOREIGN KEY

- **ON DELETE задаёт действие со строкой при удалении строки в родительской таблице**
- **DEFERRABLE позволяет отложить проверку ограничения до фиксации транзакции**
- **INITIALLY как по умолчанию осуществляется проверка**

3 - 40 SQL

Ключевые слова ограничения FOREIGN KEY

Внешний ключ определяется в подчиненной (дочерней) таблице, а столбец, на который ссылается ограничение, находится в родительской таблице. Может иметь свойства:

- ON DELETE CASCADE – означает, что если удаляется строка родительской таблицы, то одновременно с ней удаляются и подчиненные строки в дочерней таблице.
- ON DELETE SET NULL – при удалении строк в родительской таблице преобразует зависимое значение внешнего ключа в неопределенное.

Если опции ON DELETE CASCADE или ON DELETE SET NULL не установлены, то строка родительской таблицы не может быть удалена до тех пор, пока на нее имеется ссылка в дочерней таблице.

В БД Oracle также есть свои дополнительные свойства (RELY DISABLE NOVALIDATE), которые могут быть полезны.

DEFERRABLE ограничение может быть отложенным. Неоткладываемое ограничение будет проверяться немедленно после каждой команды. Если ограничение откладываемое, то INITIALLY DEFERRED – устанавливает, что по умолчанию проверка будет осуществляться при фиксации транзакции.

Ограничение CHECK

- Задает условие, которому должна удовлетворять каждая строка
- Можно обращаться к значениям столбцов только своей таблицы
- Не допускаются:
  - запросы, ссылающиеся на другие значения в других строках

```
..., salary  NUMERIC(2)
    CONSTRAINT emp_min
          CHECK (salary > 0),...
```


Ограничение CHECK

В ограничении CHECK задаётся выражение, возвращающее булевский результат, по которому определяется, будет ли успешна операция добавления или изменения для конкретных строк. Операция выполняется успешно, если результат выражения равен TRUE или NULL(UNKNOWN). Для задания условия можно использовать такие же конструкции, как WHERE в SELECT. Условия не могут содержать подзапросы или ссылаться на переменные, кроме как на столбцы текущей строки

В определении столбца может быть задано несколько ограничений CHECK. Количество ограничений CHECK для одного столбца не ограничено.

Ограничения CHECK можно задать как на уровне столбца, так и на уровне таблицы. Если определять на уровне столбца, CHECK не должен ссылаться на другие столбцы таблицы. CHECK, как и NOT NULL, проверяется немедленно и не может быть отложенным, в отличие от FOREIGN KEY, UNIQUE, PRIMARY KEY.

Создание таблицы с использованием подзапроса

```
CREATE TABLE  dept80
  AS
    SELECT  employee_id, last_name,
            salary*12 ANNSAL,
            hire_date
    FROM    employees
    WHERE   department_id = 80;
Table created.
```

```
DESCRIBE dept80
```

Name	Null?	Type
EMPLOYEE_ID		NUMBER(6)
LAST_NAME	NOT NULL	VARCHAR2(25)
ANNSAL		NUMBER
HIRE_DATE	NOT NULL	DATE

Создание таблицы на основе строк из другой таблицы

Определения типов данных и ограничения NOT NULL передаются новой таблице. Остальные правила целостности не передаются новой таблице.

В этом примере создается таблица DEPT80, которая содержит сведения о всех служащих отдела 80. Обратите внимание на то, что данные для этой таблицы берутся из таблицы EMPLOYEES.

Если выборка производится по выражению, то для задания имени столбца стоит указывать псевдоним. В примере на слайде для выражения SALARY*12 определяется псевдоним ANNSAL.

Удаление таблицы

- Удаляются все данные и структура таблицы
- Все индексы удаляются
- Все ограничения удаляются

```
DROP TABLE dept80;
Table dropped.
```


Удаление таблицы

Команда DROP TABLE удаляет определение таблицы. Если таблица удаляется, база данных теряет все данные, содержащиеся в таблице, и соответствующие индексы.

Если от таблицы зависят объекты, можно использовать опцию CASCADE для их удаления:

```
DROP TABLE dept80 CASCADE;
```

В БД Oracle представления и синонимы, зависящие от таблицы остаются, но имеют статус недействительных.

Временные таблицы

- Данные хранятся только до окончания транзакции или сеанса
- Данные видимы только для данного сеанса

```
CREATE GLOBAL TEMPORARY TABLE emp_temp
   (eno NUMERIC)
   ON COMMIT PRESERVE ROWS;
```


Временные таблицы

Если требуется временно сохранять данные до окончания транзакции или сессии с БД можно использовать временные таблицы. Работа с ними идёт быстрее, чем с таблицами постоянно хранящими данные и не нагружает БД.

Изначально строк в таблице нет.

В БД Oracle определение таблицы хранится постоянно и это определение могут использовать все сессии, однако каждая сессия будет видеть свои данные. Создавать таблицу при создании сессии не нужно.

`ON COMMIT PRESERVE ROWS` хранит данные до окончания сессии.

`ON COMMIT DELETE ROWS` хранит данные до окончания транзакции (опция по умолчанию **в Oracle**).

Рекомендация: используйте явно опции `ON COMMIT PRESERVE ROWS`, так как значения по умолчанию в БД различны.

Практическое задание 3

- Создание новых таблиц
- Создание новой таблицы с помощью синтаксиса `CREATE TABLE AS`
- Проверка существования таблицы
- Удаление таблиц

3 - 45 SQL

Практическое задание 3

4

Фильтрация и сортировка данных

Ограничение количества выбираемых строк

- Количество возвращаемых строк можно ограничить с помощью предложения WHERE:

```
SELECT что
FROM      откуда
[WHERE    логическое условие];
```

- WHERE следует за FROM

4 - 2 SQL

Ограничение количества выбираемых строк

Строки, выбираемые по запросу, можно ограничить с помощью *предложения* WHERE. Предложение WHERE следует сразу за предложением FROM и содержит условие, которое должно быть справедливо для выбираемых строк.

Предложение WHERE может сравнивать значения в столбцах, литералы, арифметические выражения и функции. Предложение WHERE состоит из трех элементов:

- имя столбца;
- оператор сравнения;
- имя столбца, константа или список значений.

Использование предложения WHERE

```
SELECT employee_id, last_name, job_id, department_id
FROM   employees
WHERE  department_id = 90 ;
```

EMPLOYEE_ID	LAST_NAME	JOB_ID	DEPARTMENT_ID
100	King	AD_PRES	90
101	Kochhar	AD_VP	90
102	De Haan	AD_VP	90

Использование предложения WHERE

Команда SELECT в примере возвращает табельный номер, фамилию, идентификатор должности и номер отдела каждого служащего, работающего в отделе 90.

В БД Oracle не допускается, так как в SQL командах БД Oracle не поддерживает тип BOOLEAN.

Можно использовать:

```
SELECT  department_id
FROM    employees
where 1=1;
```

Операторы сравнения

Оператор	Значение
=	Равно
>	Больше, чем
>=	Больше или равно
<	Меньше, чем
<=	Меньше или равно
<> !=	Не равно
BETWEEN ...AND...	Находится в диапазоне от одного значения до другого (включительно)
IN(set)	Совпадает с каким-либо значением списка
LIKE	Соответствует символьному шаблону
IS NULL	Является неопределенным значением

Операторы сравнения

Операторы сравнения используются в условиях, где одно выражение сравнивается с другим значением или выражением.

Пример

```
... WHERE salary >= 6000
... WHERE last_name = 'Smith'
```

Псевдонимы *не могут* использоваться в предложении WHERE .

Использование операторов сравнения

```
SELECT last_name, salary
FROM   employees
WHERE  salary <= 3000 ;
```

LAST_NAME	SALARY
Matos	2600
Vargas	2500

Использование операторов сравнения

В примере на слайде команда SELECT выбирает из таблицы EMPLOYEES фамилии и оклады всех служащих, чей оклад меньше или равен 3000.

BETWEEN..AND

Условие BETWEEN используется для вывода строк на основе диапазона значений

```
SELECT last_name, salary
FROM   employees
WHERE  salary BETWEEN 2500 AND 3500 ;
```

Нижняя граница   Верхняя граница

LAST_NAME	SALARY
Rajs	3500
Davies	3100
Matos	2600
Vargas	2500

Использование условия BETWEEN

С помощью условия BETWEEN можно проверять строки на вхождение в заданный диапазон значений. Диапазон имеет нижнюю и верхнюю границы.

Команда SELECT в примере выбирает из таблицы EMPLOYEES строки для всех служащих, чьи оклады находятся в диапазоне от 2500 до 3500.

Граничные значения, задаваемые с помощью условия BETWEEN, *входят* в диапазон (т.е. указываются *включительно*). Сначала необходимо задать нижнюю границу.

Условие BETWEEN можно также задавать для символьных значений:

```
SELECT last_name
FROM   employees
WHERE  last_name BETWEEN 'King' AND 'Smith';
```

IN

Условие принадлежности IN используется для проверки на вхождение значений в список.

```
SELECT employee_id, last_name, salary, manager_id
FROM   employees
WHERE  manager_id IN (100, 101, 201);
```

EMPLOYEE_ID	LAST_NAME	SALARY	MANAGER_ID
202	Fay	6000	201
200	Whalen	4400	101
205	Higgins	12000	101
101	Kochhar	17000	100
102	De Haan	17000	100
124	Mourgos	5800	100
149	Zlotkey	10500	100
201	Hartstein	13000	100

4 - 7 SQL

Использование условия IN

Для проверки принадлежности значений к заданному списку используется *условие IN*. Данное условие также называют условием принадлежности.

В вышеуказанном примере запрашиваются данные о всех служащих, менеджерами которых являются служащие под номерами 100, 101 или 201. По каждому такому служащему выводится номер служащего, фамилия, оклад и номер менеджера.

Условие IN может использоваться с данными любого типа. Следующий запрос возвращает строки таблицы EMPLOYEES для всех служащих, чья фамилия входит в список фамилий в предложении WHERE:

```
SELECT employee_id, manager_id, department_id
FROM   employees
WHERE  last_name IN ('Hartstein', 'Vargas');
```

Если в список входят символьные строки и даты, они должны быть заключены в апострофы (' ').

LIKE

- Оператор `LIKE` используется для поиска символьных значений по шаблону
- Условия поиска могут включать цифровые и символьные литералы:
  - `%` обозначает ноль или много символов;
  - `_` обозначает один символ.

```
SELECT   first_name
FROM     employees
WHERE    first_name LIKE 'S%' ;
```


Оператор LIKE

Иногда точное искомое значение неизвестно. Условие `LIKE` позволяет выбрать строки, соответствующие определенному символьному шаблону. Два символа могут использоваться для в создаваемых строках поиска.

Команда `SELECT` на сладе возвращает из таблицы `EMPLOYEES` все имена служащих, начинающиеся с буквы "S". Буква "S" – заглавная. Строки с именами, начинающимися со строчной буквы "s", в выходные данные не попадут. Для поиска с нечувствительностью к регистру можно использовать функцию: `UPPER(first_name) LIKE 's%';` Однако, в таком случае индексы на столбец использоваться не будут, хотя можно создать индексы на основе функции. Также можно использовать регулярные выражения.

В БД Oracle WHERE REGEXP_LIKE(first_name, 'y$').

При использовании этих функций также нужно использовать индексы основанные на функции и убедиться, что они используются.

В БД Oracle `LIKE` может использоваться с численными и временными типами данных. Например, допустима команда

```
SELECT salary FROM employees WHERE salary LIKE '_0%';
```

Однако, использовать это не стоит, так как формат дат и чисел зависят от настроек сессии и БД.

Использование LIKE

- Метасимволы можно комбинировать:

```
SELECT last_name
FROM   employees
WHERE  last_name LIKE '_o%' ;
```

LAST_NAME
Kochhar
Lorentz
Mourgos

- Для поиска % или _ можно использовать идентификатор ESCAPE .

```
SELECT employee_id, last_name, job_id
FROM employees WHERE job_id LIKE '%SA\_%' ESCAPE '\';
```

4 - 9 SQL

Использование LIKE

Символы % и _ можно комбинировать с литеральными символами. Пример на слайде показывает вывод всех фамилий служащих, вторая буква в которых – “*o*”.

Опция ESCAPE

Если поиск производится не по метасимволам, а по действительным символам “%” и “_” , необходимо использовать *опцию ESCAPE*. Эта опция задает символ, присутствие которого превращает метасимвол в обычный символ.

В примере на слайде ищется строка, содержащая SA_

В опции ESCAPE задан символ обратной косой черты (\). Т.к. в шаблоне косая черта предшествует символу подчеркивания (_), БД воспринимает символ подчеркивания не как метасимвол, а буквально.

Обычные индексы не будут использоваться, если шаблон поиска начинается с % и _ и выполнение команды может быть неэффективно.

LIKE чувствителен к регистру.

IS NULL/IS NOT NULL

С помощью оператора IS NULL производится проверка на неопределенные значения.

```
SELECT last_name, manager_id
FROM   employees
WHERE  manager_id IS NULL ;
```

LAST_NAME	MANAGER_ID
King	

IS NULL/IS NOT NULL

IS NULL проверяет на неопределенные значения. Если условие верно, это говорит о том, что значение недоступно, не присвоено, неизвестно или неприменимо. Проверка на равенство (=) в этом случае невозможна , так как неопределенное значение не может быть равно или неравно чему-либо. В вышеуказанном примере запрашиваются фамилии и менеджеры всех служащих, не имеющих менеджера.

Логические условия

Оператор	Значение
AND	**Возвращает результат ИСТИННО, если выполняются *оба* условия.**
OR	**Возвращает результат ИСТИННО, если выполняется *любое из* условий.**
NOT	**Возвращает результат ИСТИННО, если следующее условие не выполняется.**

4 - 11 SQL

Логические условия

Логическое условие формирует единый результат в зависимости от выполнения или невыполнения двух условий или инвертирует результат оценки одного условия. До сих пор в предложениях `WHERE` указывалось только одно условие. С помощью операторов `AND` и `OR` можно задать несколько условий в одном предложении `WHERE`.

Использование оператора AND

Оператор AND (“И“) требует выполнения обоих условий.

```
SELECT employee_id, last_name, job_id, salary
FROM   employees
WHERE  salary >=10000
AND    job_id LIKE '%MAN%' ;
```

EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
149	Zlotkey	SA_MAN	10500
201	Hartstein	MK_MAN	13000

4 - 12 SQL

Использование оператора AND

В этом примере должны выполняться оба условия, чтобы строка попала в результат. Следовательно, в выходные данные войдут строки о служащих, должности которых содержат строку MAN , *и* с окладом более $10000.

Любой поиск по символам различает верхний и нижний регистры. Если не указать MAN заглавными буквами, ни одна строка не будет выбрана. Символьные строки должны быть заключены в апострофы.

Таблица истинности для оператора AND

Следующая таблицы показывает результаты сочетания двух выражений с оператором AND:

AND	TRUE	FALSE	NULL
TRUE	TRUE	FALSE	NULL
FALSE	FALSE	FALSE	FALSE
NULL	NULL	FALSE	NULL

Использование оператора OR

Оператор OR ("ИЛИ") требует выполнения любого из условий.

```
SELECT employee_id, last_name, job_id, salary
FROM   employees
WHERE  salary >= 10000
OR     job_id LIKE '%MAN%' ;
```

EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
100	King	AD_PRES	24000
101	Kochhar	AD_VP	17000
102	De Haan	AD_VP	17000
124	Mourgos	ST_MAN	5800
149	Zlotkey	SA_MAN	10500
174	Abel	SA_REP	11000
201	Hartstein	MK_MAN	13000
205	Higgins	AC_MGR	12000

4 - 13 SQL

Использование оператора OR

В этом примере для выборки строки достаточно выполнения одного из условий. Следовательно, будут выбраны строки для всех служащих с окладом более $10000 *или* должностью, в наименование которой входит строка MAN .

Таблица истинности для оператора OR

Следующая таблицы показывает результаты сочетания двух выражений с оператором OR:

OR	TRUE	FALSE	NULL
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	NULL
NULL	TRUE	NULL	NULL

Использование оператора NOT

```
SELECT last_name, job_id
FROM   employees
WHERE  job_id
       NOT IN ('IT_PROG', 'ST_CLERK', 'SA_REP') ;
```

LAST_NAME	JOB_ID
King	AD_PRES
Kochhar	AD_VP
De Haan	AD_VP
Mourgos	ST_MAN
Zlotkey	SA_MAN
Whalen	AD_ASST
Hartstein	MK_MAN
Fay	MK_REP
Higgins	AC_MGR
Gietz	AC_ACCOUNT

4 - 14 SQL

Использование оператора NOT

В этом примере запрашиваются фамилии и должности всех служащих, идентификатор должности которых *не* IT_PROG, ST_CLERK или SA_REP.

Таблица истинности для оператора NOT

Следующая таблица показывает результат применения *оператора* *NOT* к условию:

NOT	TRUE	FALSE	NULL
	FALSE	TRUE	NULL

Оператор NOT может также использоваться с такими операторами SQL, как IN, BETWEEN, LIKE и NULL.

```
... WHERE  job_id     NOT  IN ('AC_ACCOUNT', 'AD_VP')
... WHERE  salary     NOT  BETWEEN  10000 AND  15000
... WHERE  last_name NOT  LIKE '%A%'
... WHERE  commission_pct  IS   NOT  NULL
```

Приоритеты операторов

```
SELECT last_name, job_id, salary
FROM   employees
WHERE  job_id = 'SA_REP'
OR     job_id = 'AD_PRES'
AND    salary > 15000;
```

1

LAST_NAME	JOB_ID	SALARY
King	AD_PRES	24000
Abel	SA_REP	11000
Taylor	SA_REP	8600
Grant	SA_REP	7000

```
SELECT last_name, job_id, salary
FROM   employees
WHERE  (job_id = 'SA_REP'
OR     job_id = 'AD_PRES')
AND    salary > 15000;
```

2

LAST_NAME	JOB_ID	SALARY
King	AD_PRES	24000

Порядок, в котором обрабатываются и вычисляются выражения, определяется приоритетами операторов. Приоритеты документированы, но могут меняться от версии к версии БД и запомнить приоритеты сложно.

Лучше задавать приоритеты с помощью круглых скобок, в которые заключаются выражения, обрабатываемые первыми.

1. Пример приоритета оператора AND

В этом примере два условия :

- Первое условие – идентификатор должности AD_PRES *и* оклад более 15000.
- Второе условие – идентификатор должности SA_REP.

SELECT читается следующим образом :

“Выбрать строку, если служащий – президент *и* зарабатывает более 15000 *или* если он – торговый представитель.”

2. Пример использования круглых скобок

В этом примере два условия:

- Первое условие – идентификатор должности AD_PRES *или* SA_REP.
- Второе условие – оклад более 15000.

Таким образом, команда SELECT читается следующим образом:

“Выбрать строку, если служащий – президент *или* торговый представитель *и* если он зарабатывает более 15000.”

ORDER BY

- **Выражение ORDER BY используется для сортировки строк. Прядок сортировки можно задать опциональным выражением**
  - **ASC: сортировка по возрастанию (используется по умолчанию)**
  - **DESC: сортировка по убыванию**

В команде SELECT выражение ORDER BY указывается последним

```
SELECT   last_name, job_id, department_id, hire_date
FROM     employees
ORDER BY hire_date ASC, last_name, job_id DESC;
```

4 - 16 SQL

Использование предложения ORDER BY

Если предложение ORDER BY не используется, порядок сортировки не определен, и при повторе запроса БД может выдать строки в другой последовательности.

ORDER BY должно быть самым последним в команде. Логически сортировка выполняется как самый последний шаг в выполнении запроса.

Для сортировки можно задать столбцы выражение, псевдонимы или номера столбцов (номер_позиции). Также можно указать сортировку по столбцам, отсутствующим в результате.

БД Oracle допускает ORDER BY только там, где это имеет смысл. Например, в конце команды или в окнах.

БД postgres допускает использование ORDER BY во вложенных SELECT, но смысла это не имеет.

Сортировка

- Сортировка в порядке убывания:

```
SELECT last_name, job_id, department_id, hire_date
FROM   employees
ORDER BY hire_date DESC;
```

1

- Сортировка по псевдониму столбца:

```
SELECT employee_id, salary*12 annsal
FROM   employees
ORDER BY annsal ;
```

2

- Сортировка по нескольким столбцам:

```
SELECT last_name, salary FROM employees
ORDER BY department_id ASC NULLS FIRST,2 DESC NULLS LAST;
```

3

Сортировка данных по умолчанию
По умолчанию данные сортируются в порядке возрастания:
Числовые значения выводятся, начиная с самых малых, например, с 1 по 999.
Даты выводятся, начиная с самых ранних, например, дата 01-JAN-92 предшествует дате 01-JAN-95.
Символьные значения выводятся в алфавитном порядке, например, от A до Z.
Неопределенные значения при сортировке по возрастанию выводятся последними, а при сортировке по убыванию - первыми, если не указать NULLS FIRST/NULLS LAST.
Сортировку можно производить и по столбцам, не указанным в предложении SELECT.

Примеры на слайде
1. Для изменения порядка сортировки на обратный укажите ключевое слово DESC после названия конкретного столбца в предложении ORDER BY. В примере на слайде первыми в списке появляются недавно принятые служащие.
ASC/DESC относится к столбцу/выражению/номеру после которого указано
2. В предложении ORDER BY можно использовать псевдоним столбца. В примере на слайде данные сортируются по годовому доходу.
3. Результаты запроса можно сортировать по нескольким столбцам. Предельным количеством столбцов для сортировки является количество столбцов в данной таблице. В предложении ORDER BY столбцы задаются через запятые. Чтобы изменить порядок сортировки столбца на убывающий, можно указать DESC после названия столбца.
Чтобы значения NULL выводились первыми, используется NULLS FIRST.

Ссылки на столбцы в ORDER BY

- **Варианты ссылок на столбцы(выражения):**

```
SELECT last_name, salary*12,
UPPER(first_name) "FN"
FROM   employees
ORDER BY job_id, 1, salary*12, "FN";
```

- **Числа в SELECT не считаются ссылками на столбцы, они считаются литералами:**

```
SELECT 1,2,3 "three"
FROM   employees;
         1          2      three
---------- ---------- ----------
         1          2          3
         1          2          3
```


Ссылки на столбцы в ORDER BY

Можно ссылаться как на столбцы и выражения результатирующей выборки, указанные после SELECT, а также на столбцы из таблиц из фразы FROM, не присутствующие в результате выборки.

Ссылаться на столбцы во фразе ORDER BY можно разными способами:

1) можно ссылаться по имени столбца. В примере на слайде job_id отсутствует в SELECT, но сортировка по нему будет производиться

2) по порядковому номеру столбца в выборке SELECT - синий цвет на слайде

3) по любому выражению. На слайде salary*12. При этом связи с тем выражением, которое указано в SELECT формально нет.

4) по псевдониму столбца в выборке SELECT.

Обратите внимание, что порядковый номер столбца можно указывать в ORDER BY, в SELECT так ссылаться на столбец нельзя, вы получите литерал.

Оператор COLLATE

- **позволяет сортировать данные по заданным лингвистическим правилам**

```
SELECT last_name FROM employees
ORDER BY last_name COLLATE generic_m_ci;
```

- **в БД postgres можно создавать правила командой `CREATE COLLATION` и использовать в операторе `COLLATE`**
- **БД Oracle, по умолчанию, выполняет сравнения по бинарному представлению символов**
- **Унаследованная функция для сортировок**

```
ORDER BY NLSSORT(last_name,'NLS_SORT=GENERIC_M_CI');
```


Оператор COLLATE
используется если сортировка по умолчанию не удовлетворяет задачам. Является стандартным оператором SQL.
Названия сортировок не стандартизованы и зависят от производителя БД или операционной системы.
БД Oracle:
По умолчанию используется сортировка типа BINARY - по бинарному представлению символов. Для кириллицы, если БД использует схем кодирования AL32UTF8, проблема с буквой Ё.
Недостаток использования других типов сортировки: БД Oracle не может использовать обычные индексы, и нужно создавать индексы, основанные на функции NLSSORT.
Аналогичное верно и для операций сравнения и поиска по диапазону WHERE last_name > 'литерал'. По умолчанию используется BINARY сравнение.
`ORDER BY 1 COLLATE generic_m_ci` (суффикс _ci - нечувствительность к регистру, суффикс _ai - нечувствительность к диакритическим значкам, _cs - нечувствительность к регистру и диакритическим значкам, название generic - сравнения текстов на разных языках по стандарту ISO 14651)
или `ORDER BY NLSSORT(n,'NLS_SORT=GENERIC_M_CI')`
Функции UPPER и LOWER не подходят для использования в сортировках без учёта регистра, так как буквы в разном регистре неразличимы после преобразования
Обычные индексы с лингвистическими сравнениями не используются, нужно создавать индексы, основанные нв функции

Соответствие сортировок и сравнений

Если базы данных используют кодировку Unicode для хранения текстовых данных:

- БД postgres использует многоязычную сортировку и сравнение текстовых строк по стандарту ISO 14651
- Буквы нижнего регистра идут до букв верхнего регистра, что не соответствует правилам русского языка
- Аналогичное поведение в БД Oracle достигается использованием
  - NLS_SORT='GENERIC_M'
  - NLS_COMP='LINGUISTIC'
  - созданием индексов, основанных на функции

4 - 20 SQL

Соответствие сортировок и сравнений

Если не обращать внимание на буквы Ё,ё то сортировки и сравнения по умолчанию **в БД Oracle** соответствуют правилам русского языка (буквы верхнего регистра идут раньше букв нижнего регистра) и используются обычные индексы.

Правила сравнения можно задать на уровне столбца:

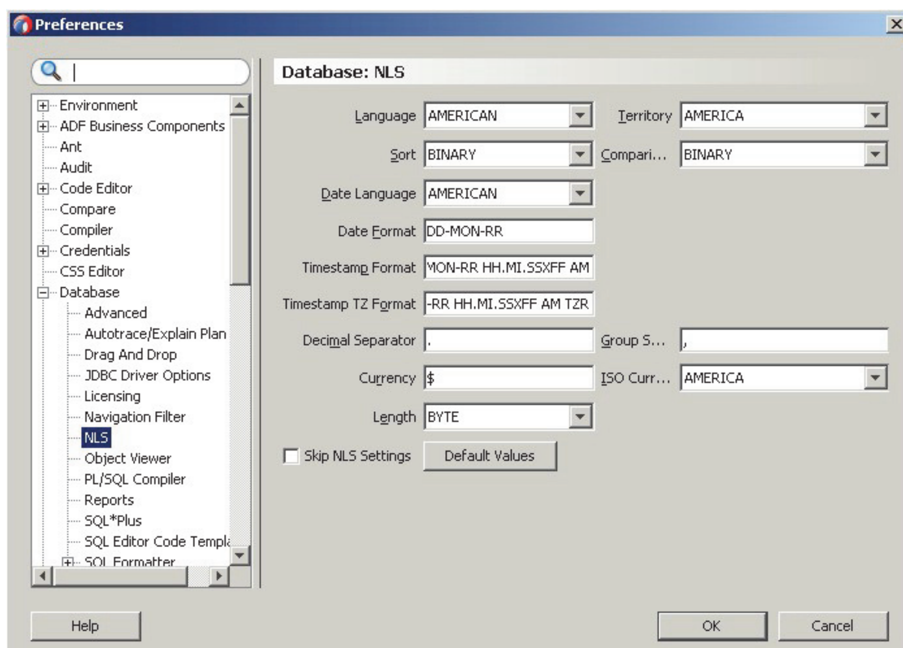
БД Oracle:

возможность появилась в версии 12.2 и если БД изменена на поддержку длинных строк (MAX_STRING_SIZE=EXTENDED), что редко используют. Пример:

```
CREATE TABLE test (n VARCHAR(2) COLLATE "GENERIC_CI");
```

Настройки сессии с БД Oracle в JDeveloper

- **JDeveloper как клиент к БД может установить свойства для сессии Tools->Preferences**



Настройки сессии

В sqlplus и sqlcl аналогичные настройки устанавливаются переменными окружения. Основная переменная окружения NLS_LANG, она переопределяет значения по умолчанию других параметров.

При установке "`NLS_LANG=RUSSIAN_RUSSIA.`" значение NLS_SORT примет значение RUSSIAN, вместо BINARY.

NLS_COMP определяет поведение операций сравнения и поиска по диапазону, принимает два значения BINARY и LINGUISTIC (или устаревшее значение ASCII). Если установлен LINGUISTIC (ASCII), то тип сравнения зависит от значения NLS_SORT.

Параметры можно поменять в сессии командой ALTER SESSION SET. Действующие в сессии параметры можно посмотреть в представлении NLS_SESSION_PARAMETERS.

В Windows для любых консольных утилит (sqlplus) могут использоваться шрифты в кодировках 866 и 1251, кодовая страница устанавливается командой chcp. Шрифты выбираются в свойствах окна и должны иметь установленную кодовую страницу.

Практическое задание 4

- Выборка данных и изменение последовательности вывода строк
- Ограничение количества возвращаемых строк с помощью предложения `WHERE`
- Сортировка строк с помощью предложения `ORDER BY`
- Использование переменных подстановки для создания более гибких команд `SELECT` языка SQL

Практическое задание 4

Упражнения этого занятия позволяют получить навыки создания отчетов и использования предложений `WHERE` и `ORDER BY`. При использовании переменных подстановки команды SQL имеют более общий вид и больше подходят для повторного использования.

5

Использование однострочных функций

Функции преобразования регистра символов

Эти функции преобразуют регистр символьных строк

Функция	Результат
`LOWER('SQL course')`	`sql course`
`UPPER('SQL course')`	`SQL COURSE`
`INITCAP('SQL course')`	`Sql Course`

5 - 2 SQL

Функции преобразования регистра символов

Однострочные функции возвращают одну строку

Три функции преобразования регистра символов: `LOWER`, `UPPER` и `INITCAP`.

- `LOWER`; преобразует строку символов верхнего регистра или обоих регистров в символы нижнего регистра.
- `UPPER`; преобразует строку символов нижнего регистра или обоих регистров в символы верхнего регистра.
- `INITCAP`; преобразует первую букву **каждого слова** в заглавную, а остальные буквы – в строчные.

Использование функций преобразования регистра

Вывод номера служащего по фамилии Higgins, его фамилии и отдела:

```
SELECT employee_id, last_name, department_id
FROM   employees
WHERE  last_name = 'higgins';
no rows selected
```

```
SELECT employee_id, last_name, department_id
FROM   employees
WHERE  LOWER(last_name) = 'higgins';
```

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
205	Higgins	110

5 - 3 SQL

Использование функций преобразования регистра

Пример показывает номер, фамилию и номер отдела служащего Higgins.

Предложение `WHERE` в первой команде SQL задает фамилию служащего – `higgins`. Т.к. все данные в таблице `EMPLOYEES` хранятся в символах соответствующего регистра, совпадения с фамилией `higgins` не будет, вследствие чего ни одна строка не будет выбрана.

Предложение `WHERE` во второй команде SQL указывает, что фамилия служащего из таблицы `EMPLOYEES` должна быть преобразована в строчные буквы и только после этого сравниваться с `higgins`. Т.к. обе фамилии теперь представлены в символах нижнего регистра, совпадение будет обнаружено и будет выбрана одна строка. Следующее предложение `WHERE` даст такой же результат:

```
...WHERE last_name = 'Higgins'
```

В выходных данных фамилия выглядит так, как хранится в базе данных. Чтобы вывести фамилию в символах верхнего регистра, можно использовать функцию `UPPER` в списке выбора команды `SELECT`.

```
SELECT employee_id, UPPER(last_name), department_id
FROM   employees
WHERE  INITCAP(last_name) = 'Higgins';
```

Функции обработки строк

Функция	Результат
CONCAT('Hello', 'World')	HelloWorld
SUBSTR('HelloWorld',1,5)	Hello
LENGTH('HelloWorld')	10
TRIM('H' FROM 'HelloWorld')	elloWorld
LPAD(last_name,10,'*')	*****King
RPAD(last_name, 10, '*')	King*****
REPLACE('JACK and JUE','J','BL')	BLACK and BLUE
REVERSE('abcd')	dcba

Функции обработки строк

CONCAT, SUBSTR, LENGTH, INSTR, LPAD, RPAD и TRIM – это функции манипулирования символами, которые обсуждаются в этом уроке.

- CONCAT: соединяет значения. Для функции CONCAT можно использовать не более двух параметров.
- SUBSTR: возвращает подстроку заданной длины.
- LENGTH: возвращает длину строки в виде числового значения.
- TRIM: удаляет из символьной строки начальные и/или конечные символы. (Если удаляемый_символ или исходная_строка являются символьными литералами, то их нужно заключить в апострофы)
- LPAD: дополняет символьное значение, выровненное справа, до заданной длины.
- RPAD: дополняет символьное значение, выровненное слева, до заданной длины.
- REVERSE: переворачивает строку
- ASCII: возвращает код ASCII первого символа строки

Пример использования функций

Oracle:

```
SELECT employee_id,
CONCAT(first_name, last_name) n, job_id, LENGTH(last_name)
FROM   employees
WHERE  SUBSTR(last_name, -1, 1) = 'g';
```

Postgres:

```
SELECT employee_id,
CONCAT(first_name, last_name) n, job_id, LENGTH(last_name)
FROM   employees
WHERE  SUBSTRING(last_name from '.$') = 'g';
```

выдаются данные по служащим, чьи фамилии заканчиваются символом “g”

5 - 5 SQL

Пример использования функций

В БД разных производителей могут быть реализованы разные наборы функций. Функции с аналогичным функционалом могут иметь разные называния.

Функции могут работать с разными типами аргументов и смысл аргументов может оличаться.

Пример:

В **БД Oracle** при задании -1 в функции SUBSTR выдается последний символ. В БД Postgres отрицательное значение не используется. Однако, есть функция SUBSTRING, которую можно использовать для достижения такого же результата. Также использование ключевых слов в БД разных производителей может отличаться.

Результат выполнения команды в psql:

```
 employee_id |       n          |  job_id   | length
-------------+-----------------+-----------+--------
         100 | StevenKing      | AD_PRES   |      4
         108 | NancyGreenberg  | FI_MGR    |      9
         122 | PayamKaufling   | ST_MAN    |      8
         137 | RenskeLadwig    | ST_CLERK  |      6
         156 | JanetteKing     | SA_REP    |      4
         188 | KellyChung      | SH_CLERK  |      5
(6 строк)
```

Примеры функций, работающих с числами

- ROUND: округляет значение до заданного количества десятичных знаков
- TRUNC: усекает значение до заданного количества десятичных знаков
- MOD: возвращает остаток от деления

Функция	Результат
ROUND(45.926, 2)	45.93
TRUNC(45.926, 2)	45.92
MOD(1600, 300)	100

Примеры функций, работающих с числами

Функции для работы с числами принимают на входе численные типы и возвращают численные типы.

Примеры часто используемых функций:

Функция	Назначение
ROUND(*аргумент*, *n*)	Округляет аргумент до *n* десятичных разрядов, а если *n* опущено, то до целого. Если *n* отрицательно, округляются разряды слева от десятичной точки.
TRUNC(*аргумент*,*n*)	Аргумент усекается до *n* десятичных разрядов, а если *n* опущено, то до целого. Если *n* отрицательно, усекаются разряды слева от десятичной точки.
MOD(*m*,*n*)	остаток от деления *m* на *n*.

Использование функции ROUND

```
SELECT ROUND(45.923,2), ROUND(45.923,0), ROUND(45.923,-1)
FROM   DUAL;
```

ROUND(45.923,2)	ROUND(45.923,0)	ROUND(45.923,-1)
45.92	46	50

DUAL – это таблица, которая обычно используется для получения результатов выполнения функций и вычислений в БД Oracle

Функция ROUND

округляет значение по математическим правилам, выражение или значение до *n* десятичных разрядов. Если второй аргумент равен нулю или отсутствует, значение округляется до нуля десятичных разрядов. Если второй аргумент равен 2, значение округляется до двух десятичных разрядов. Если второй аргумент равен -1, значение округляется слева от десятичной точки - до десятков (до целого числа с одним нулем).

Функция ROUND может использоваться и с функциями даты.

Таблица DUAL в БД Oracle

Таблица DUAL содержит один столбец DUMMY и одну строку со значением X. Таблица DUAL полезна, когда необходимо получить разовый результат, например: вывести значение константы, выражения, функции. В БД Oracle в SELECT обязательно использовать FROM и таблица DUAL используется для того, чтобы не был нарушен синтаксис. Эта таблица позволяет выполнять вычисления, для которых не нужны данные из реальных таблиц.

Использование функции TRUNC

```
SELECT TRUNC(45.923,2), TRUNC(45.923), TRUNC(45.923,-1)
FROM   DUAL;
```

TRUNC(45.923,2)	TRUNC(45.923)	TRUNC(45.923,-1)
45.92	45	40

Функция TRUNC

БД Oracle для вычислений внутренне использует тип NUMBER, который позволяет оперировать числами с высокой точностью - до 40 значащих разрядов.

```
SELECT 1/6 from dual;
```

1/6
0.16666666666666666666666666666666666666667

Чтобы отбросить ненужные разряды результата вычислений можно использовать функцию TRUNC. Функция TRUNC усекает значение, отбрасывая разряды.

Аргументы функции TRUNC аналогичны аргументам функции ROUND. Если второй аргумент равен нулю или отсутствует, значение усекается до нуля десятичных разрядов. Если второй аргумент равен 2, значение усекается до двух десятичных разрядов. Если второй аргумент равен -1, значение усекается до десятков.

Использование арифметических операторов с датами

```
SELECT last_name, ROUND((CURRENT_DATE-hire_date)/7,6)
AS WEEKS FROM employees
WHERE  department_id = 90;
```

LAST_NAME	WEEKS
King	744.245395
Kochhar	626.102538
De Haan	453.245395

5 - 9 SQL

Использование арифметических операторов с датами

Пример сверху показывает вывод фамилий и количества отработанных недель всех служащих отдела 90.

Если более ранняя дата вычитается из более поздней, результатом будет отрицательное число.

Функции SQL для работы с датами

Функция EXTRACT получает из значений даты/времени поля численное значение.

```
SELECT EXTRACT(HOUR FROM TIMESTAMP '2001-02-16
20:38:40') from dual;
20
```

Функция CAST используется для приведения типов данных

```
SELECT CAST(current_timestamp AS date) FROM DUAL;
15-MAR-17
```

Конструкция AT TIME ZONE позволяет переводить время в разные часовые пояса.

```
SELECT TIMESTAMP '2001-02-16 20:38:40' AT TIME ZONE
'MST' from dual;
16-FEB-01 10.38.40.000000000 AM MST
```

Для поиска по диапазону можно использовать выражение

```
SELECT hire_date FROM employees where hire_date
BETWEEN TIMESTAMP '2000-04-16 20:38:40' AND
current_date;
```


Функции SQL для работы с датами

Основные типы данных для хранения и манипулирования датами DATE и группа типов TIMESTAMP (с временными зонами и без). На слайде приведены стандартные функции SQL.

Временные интервалы

- Интервалы времени используются в вычислениях с датавременными типами
- YEAR, MONTH, DAY, HOUR, MINUTE, SECOND
- значение интервала заключается в апострофы
- Операторы + - * /

```
SELECT
TIMESTAMP '2017-10-19 10:23:54' - INTERVAL '5' MINUTE;
19-OCT-17 10.18.54.000000000 AM
2017-10-19 10:18:54.0
SELECT
INTERVAL '777' HOUR + INTERVAL '5.123456789' SECOND;
32 9:0:5.123457
0 years 0 mons 0 days 777 hours 0 mins 5.123457 secs
```

5 - 11 SQL

Временные интервалы

Тип данных INTERVAL дополнительно позволяет ограничить набор сохраняемых полей следующими фразами:

YEAR, MONTH, DAY, HOUR, MINUTE, SECOND, YEAR TO MONTH, DAY TO HOUR, DAY TO MINUTE, DAY TO SECOND, HOUR TO MINUTE, HOUR TO SECOND, MINUTE TO SECOND

Тип данных INTERVAL можно использовать при создании столбцов таблиц.

Дополнительные функции для работы с датами

Примеры функций в БД Oracle

Функция	Результат
MONTHS_BETWEEN	Число месяцев, разделяющих две даты в формате NUMBER (десятичное)
ADD_MONTHS	Добавление календарных месяцев к дате
NEXT_DAY	Ближайшая дата, когда наступит заданный день недели
LAST_DAY	Последняя дата текущего месяца
ROUND	Округление даты
TRUNC	Усечение даты

Дополнительные функции для работы с датами

В БД Oracle функции ROUND и TRUNC могут использоваться для числовых значений и дат. Если они используются с датами, даты округляются или усекаются в соответствие с заданной моделью формата. Следовательно, можно округлять даты до ближайшего года или месяца.

```
SELECT TRUNC(TIMESTAMP '2001-02-16 20:38:40', 'MONTH') FROM dual;
01-FEB-01
```

В БД есть функции, корректно вычисляющие временные интервалы в разных единицах (днях, годах) между двумя датами. Эта задача не сводится к арифметике, так как количество дней в месяцах и годах разное. В БД postgres используется функция AGE(), в БД Oracle MONTHS_BETWEEN и другие.

Неявное преобразование типов данных

При вычислении выражений БД Oracle может автоматически выполнить приведение типов в обе стороны:

Исходный формат	Новый формат
VARCHAR или CHAR	NUMBER
VARCHAR или CHAR	DATE

БД postgres, по умолчанию, автоматических преобразований не выполняет, нужно использовать явное преобразование или создать определение преобразования:

```
CREATE CAST (CHAR AS NUMERIC) WITH INOUT AS IMPLICIT;
```


Неявное преобразование типов данных

Обычно БД использует правила для выражения, когда необходимое преобразование не охвачено правилом преобразования типов данных для операций присваивания.

Например, в выражении `salary = '20000'` строка `'20000'` неявно преобразуется в число 20000.

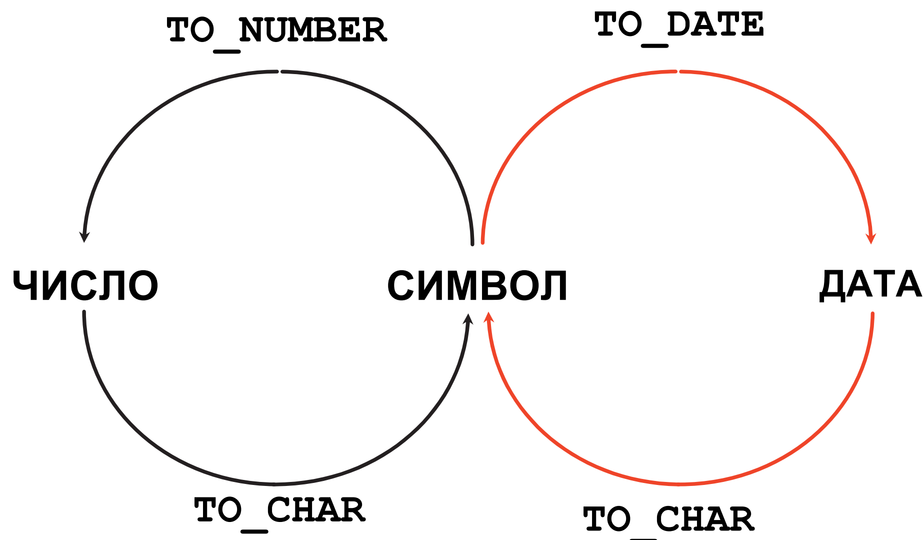
Примечание

Преобразование из `CHAR` в `NUMBER` возможно только в случае, если символьная строка действительно представляет число.

БД Oracle:

```
SELECT CAST(1 AS CHAR) + CAST(1 AS CHAR) plus FROM DUAL;
PLUS
----
   2
```

Явное преобразование типов данных



5 - 14 SQL

Явное преобразование типов данных

CAST(выражение AS тип данных) - приводит выражение к типу данных.
В **БД Oracle** для текстовых типов обязательно указывать длину, например VARCHAR**(15)**

CAST не позволяет задавать формат. Для явного преобразования значения из одного типа данных в другой используется функции:

```
TO_CHAR(аргумент,'формат'),
TO_NUMBER(аргумент,'формат',
TO_DATE(аргумент,'формат')
```

Набор допустимых метасимволов в строке 'формат' для БД разных производителей может отличаться.

Если нужно задать формат выражения, то используются функции:

```
SELECT to_char(current_timestamp, 'HH12:MI:SS');
04:28:58
SELECT to_date('05 Dec 2000', 'DD Mon YYYY');
2000-12-05
SELECT to_number('12,454.8-', '99G999D9S');
-12548
SELECT to_timestamp('05 Dec 2000', 'DD Mon YYYY');
2000-12-05 00:00:00.0
```

В формате можно указать **fm** чтобы убрать слева пробелы или нули:

```
SELECT TO_CHAR(1234.56778, 'fm99,999.00') ;
```

GREATEST/LEAST

- выбирают наибольшее или наименьшее значение из аргументов
- Могут иметь много аргументов

```
SELECT GREATEST(1,'2',3);
3
SELECT GREATEST('1','2','3','ab');
ab
```

- аргументы должны иметь совместимый тип данных

```
SELECT GREATEST('1',2,'ab');
ОШИБКА: неверное значение для целого числа: "ab"
```

5 - 15 SQL

GREATEST/LEAST

Функции GREATEST и LEAST выбирают наибольшее или наименьшее значение из списка выражений. Все эти выражения должны приводиться к общему типу данных, который станет типом результата.

Функции GREATEST и LEAST не описаны в стандарте SQL.

В БД Oracle возвращают NULL, когда любой из аргументов равен NULL:

```
select least(1,2,3, '5', null) from dual;
```

вернёт null

Использование функции COALESCE

- Возвращает первое по порядку непустое (NOT NULL) значение
- Может иметь много аргументов

```
SELECT COALESCE(NULL,NULL, 'abc','bcd');

abc
```

- аргументы должны иметь совместимый тип данных

```
SELECT COALESCE(NULL,NULL, 'abc',1);

ОШИБКА: неверное значение для целого числа: "abc"
```


Использование функции COALESCE

Все аргументы должны иметь один и тот же тип данных

```
SELECT COALESCE(NULL,NULL, CAST('abc' AS CHAR),
   CAST('abc' AS VARCHAR(10))) from DUAL;
COALESCE
----------
a
```

Примечание

В БД Oracle есть функции NVL, NVL2 с аналогичным функционалом. Эти функции нестандартны и отсутствуют в других БД. Они могут быть заменены стандартной функцией COALESCE.

Использование функции NULLIF

```
SELECT first_name, LENGTH(first_name) "expr1",
       last_name,  LENGTH(last_name)  "expr2",
       NULLIF(LENGTH(first_name), LENGTH(last_name)) result
FROM   employees;
```

FIRST_NAME	expr1	LAST_NAME	expr2	RESULT
Steven	6	King	4	6
Neena	5	Kochhar	7	5
Lex	3	De Haan	7	3
Alexander	9	Hunold	6	9
Bruce	5	Ernst	5	
Diana	5	Lorentz	7	5
Kevin	5	Mourgos	7	5
Trenna	6	Rajs	4	6
Curtis	6	Davies	6	

...

5 - 17 SQL

Использование функции NULLIF

Функция NULLIF сравнивает два аргумента. Если они равны, функция возвращает NULL, если нет – первый аргумент.

В приведенном на слайде примере сравниваются имена и фамилии служащих из таблицы EMPLOYEES . Если длины этих имен не совпадают, выдается длина имени, в противном случае – NULL.

В БД Oracle в качестве первого аргумента нельзя указать литерал NULL.

Функция NULLIF логически эквивалентна следующему выражению CASE , которое рассматривается далее.

```
CASE WHEN выражение1 = выражение2 THEN NULL ELSE выражение1 END
```

Условные выражения

- **Позволяют применять логические конструкции ЕСЛИ-ТО-ИНАЧЕ (IF-THEN-ELSE) внутри команды SQL**
- **выражение CASE**

Условные выражения CASE

Два метода используются для выполнения условной обработки: логическая конструкция IF-THEN-ELSE и выражение CASE.

CASE удовлетворяет стандарту SQL;

В БД Oracle есть унаследованное от старых версий выражение DECODE с тем же функционалом, но другим синтаксисом.

Синтаксис

```
DECODE(expr, search, result [, search, result...] [, default])
```

эквивалнтен

```
CASE WHEN expr=search THEN result [...] ELSE default END
```

Выражение CASE

Помогает создавать условные запросы, которые выполняют действия логического оператора `IF-THEN-ELSE`:

```
CASE выражение
          WHEN сравн_выражение1 THEN возвр_выражение1
         [WHEN сравн_выражение2 THEN возвр_выражение2
          WHEN сравн_выражениеn THEN возвр_выражениеn
          ELSE else_выражение]
END
```

5 - 19 SQL

Выражение CASE

Выражение CASE позволяет производить логическую обработку оператора IF-THEN-ELSE в командах SQL, не вызывая процедуры.

В простом выражении CASE БД ищет первую пару WHEN ... THEN, в которой совпадают выражение и сравн_выражение и возвращает возвр_выражение. Если нет совпадений ни в одной из сравниваемых пар WHEN ... THEN и есть предложение ELSE, возвращается else_выражение. Если такого предложения ELSE нет, возвращается неопределенное значение. Нельзя задавать литерал NULL для возвр_выражений и else_выражений.

Все выражения (выражение, сравн_выражение и возвр_выражение) должны быть одного типа. Допустимые типы: CHAR, VARCHAR2, NCHAR и NVARCHAR2.

Использование выражения CASE

Помогает создавать условные запросы, которые выполняют действия логического оператора `IF-THEN-ELSE`:

```
SELECT last_name, job_id, salary,
       CASE job_id WHEN 'IT_PROG'  THEN  1.10*salary
                   WHEN 'ST_CLERK' THEN  1.15*salary
                   WHEN 'SA_REP'   THEN  1.20*salary
       ELSE      salary END      "REVISED_SALARY"
FROM   employees;
```

LAST_NAME	JOB_ID	SALARY	REVISED_SALARY
...			
Lorentz	IT_PROG	4200	4620
Mourgos	ST_MAN	5800	5800
Rajs	ST_CLERK	3500	4025
...			
Gietz	AC_ACCOUNT	8300	8300

5 - 20 SQL

Использование выражения CASE

В приведенной команде SQL расшифровывается значение идентификатора должности `JOB_ID`. Если значение `JOB_ID` совпадает с `IT_PROG`, оклад повышается на 10%; если `JOB_ID` равно `ST_CLERK`, повышение составляет 15%; если `JOB_ID` равно `SA_REP`, оклад увеличивается на 20%. Для всех остальных должностей оклад не изменяется.

Такую же команду можно написать с помощью функции `DECODE`

Ниже приведен пример *выражения `CASE` для выполнения поиска* (*searched CASE expression*). В таком выражении производится поиск слева направо, пока не будет найдено первое верное из перечисленных условий. Если такое условие найдено, возвращается соответствующее выражение. Если нет ни одного верного условия и существует предложение `ELSE`, возвращается выражение из этого предложения; в противном случае возвращается неопределенное значение (`NULL`).

```
SELECT last_name,salary,
(CASE WHEN salary<5000 THEN 'Low'
      WHEN salary<10000 THEN 'Medium'
      WHEN salary<20000 THEN 'Good'
      ELSE 'Excellent'
END) qualified_salary
FROM employees;
```

Вложение функций

```
SELECT last_name,
  UPPER(CONCAT(SUBSTR(LAST_NAME, 1, 8), '_US'))
FROM   employees
WHERE  department_id = 60;
```

LAST_NAME	UPPER(CONCAT(SUBSTR(LAST_NAME,1,8
Hunold	HUNOLD_US
Ernst	ERNST_US
Lorentz	LORENTZ_US

Вложение функций

В примере на слайде на экран выводятся фамилии сотрудников отдела 60. Работа команды SQL включает три шага:

1. Внутренняя функция выбирает первые восемь символов фамилии.
   Результат1 = SUBSTR (LAST_NAME, 1, 8)
2. Внешняя функция объединяет результат1 с _US.
   Результат2 = CONCAT(Result1, '_US')
3. Самая последняя внешняя функция переводит результат2 в заглавные символы.

Практическое задание 5

- Составление запросов, требующих использования числовых, символьных функций и функций для работы с датами
- Использование конкатенации с функциями
- Составление запросов, нечувствительных к регистру символов

5 - 22 SQL

Практическое задание 5

Выборка данных из нескольких таблиц

6

Рассматриваемые вопросы

- Команды SELECT для выборки данных из нескольких таблиц
- Соединение таблицы с собой с помощью рефлексивного соединения
- Использование внешних соединений для просмотра данных, не удовлетворяющих обычным условиям соединения

Рассматриваемые вопросы

Урок посвящен методам выборки данных из нескольких таблиц. Для просмотра информации из нескольких таблиц используется соединение (join)

Выборка данных из нескольких таблиц

EMPLOYEES

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
100	King	90
101	Kochhar	90
...		
202	Fay	20
205	Higgins	110
206	Gietz	110

DEPARTMENTS

DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION_ID
10	Administration	1700
20	Marketing	1800
50	Shipping	1500
60	IT	1400
80	Sales	2500
90	Executive	1700
110	Accounting	1700
190	Contracting	1700

EMPLOYEE_ID	DEPARTMENT_ID	DEPARTMENT_NAME
200	10	Administration
201	20	Marketing
202	20	Marketing
...		
102	90	Executive
205	110	Accounting
206	110	Accounting

Выборка данных из нескольких таблиц

Иногда требуются данные из более, чем одной таблицы.

Например, нужно получить номер сотрудника EMPLOYEE_ID и название его отдела DEPARTMENT_NAME

- EMPLOYEE_ID существует в таблице EMPLOYEES.
- DEPARTMENT_ID существует как в таблице EMPLOYEES, так и в таблице DEPARTMENTS.
- DEPARTMENT_NAME существует в таблице DEPARTMENTS.

Для получения отчета необходимо соединить таблицы EMPLOYEES и DEPARTMENTS по столбцу DEPARTMENT_ID и произвести выборку данных из обеих таблиц.

.

NATURAL JOIN

- **Связь идёт по всем столбцам двух таблиц, имеющим одинаковые имена и типы данных**

Преимущество:

- **компактность записи**

Недостатки:

- **Вы не видите столбцы по которым идёт соединение**
- **Если столбцы с одинаковыми именами имеют разные типы данных, возвращается сообщение об ошибке**
- **Соединение может пройти не так как вы хотели**
- **При добавлении столбцов к таблицам соединения могут начать рабоать по-другому**

6 - 4 SQL

NATURAL JOIN

Возможно автоматическое соединение с использованием ключевых слов NATURAL JOIN на основе столбцов двух таблиц, которые имеют одинаковые имена и типы данных.

Соединение может быть произведено только, если столбцы имеют одинаковые имена и типы данных. Если столбцы имеют одинаковые имена, но различные типы данных, использование синтаксиса NATURAL JOIN вызывает появление сообщения об ошибке.

Выборка строк с помощью натуральных соединений

```
SELECT department_id, department_name,
       location_id, city
FROM   departments
NATURAL JOIN locations ;
```

DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION_ID	CITY
60	IT	1400	Southlake
50	Shipping	1500	South San Francisco
10	Administration	1700	Seattle
90	Executive	1700	Seattle
110	Accounting	1700	Seattle
190	Contracting	1700	Seattle
20	Marketing	1800	Toronto
80	Sales	2500	Oxford

8 rows selected.

Выборка строк с помощью натуральных соединений

В примере на слайде таблица LOCATIONS соединяется с таблицей DEPARTMENTS на основе столбца LOCATION_ID , который является единственным столбцом с одинаковым именем в обоих таблицах. Если бы были другие общие столбцы, они бы тоже использовались для соединения таблиц.

Второй способ соединения: JOIN USING

- Явно задаются столбцы по которым нужно выполнить соединение
- Можно использовать вместо NATURAL JOIN
- Имена таблиц и псевдонимы не указываются в ссылках на столбцы

Остаётся недостаток NATURAL JOIN:

- имена столбцов по которым нужно соединять должны быть одинаковы

6 - 6 SQL

Второй способ соединения: JOIN USING

Натуральные соединения используют все столбцы с **одинаковыми именами и типами данных** для соединения таблиц. В предложении USING могут быть указаны только те столбцы, которые следует использовать в эквисоединении. Столбцы внутри USING (...), **не** должны включать квалификаторы (имя таблицы или псевдоним), которые применяются в любом другом месте команды SQL.

Например, эта команда не содержит ошибок:

```
SELECT l.city, d.department_name
FROM   locations l JOIN departments d USING (location_id)
WHERE  location_id = 1400;
```

Эта команда не выполнится в **БД Oracle**, так как столбец LOCATION_ID содержит квалификатор в предложении WHERE:

```
SELECT l.city, d.department_name
FROM locations l JOIN departments d USING (location_id)
WHERE d.location_id = 1400;
ORA-25154: column part of USING clause cannot have qualifier
```

В БД Oracle такое же ограничение накладывается на натуральные соединения.

Использование JOIN USING

```
SELECT employee_id, last_name,
       location_id, department_id
FROM   employees JOIN departments
USING (department_id);
```

EMPLOYEE_ID	LAST_NAME	LOCATION_ID	DEPARTMENT_ID
200	Whalen	1700	10
201	Hartstein	1800	20
202	Fay	1800	20
124	Mourgos	1500	50
141	Rajs	1500	50
142	Davies	1500	50
144	Vargas	1500	50
143	Matos	1500	50

...

6 - 7 SQL

Использование JOIN USING

В приведенном примере выполняется соединение таблиц EMPLOYEES и DEPARTMENTS на основе столбца DEPARTMENT_ID. В результате выдаются сведения об идентификаторах местоположения подразделений, в которых работают служащие.

Использование псевдонимов таблиц

- Упрощает запросы
- Повышает производительность

```
SELECT e.employee_id, e.last_name,
       d.location_id, department_id
FROM   employees e JOIN departments d
USING (department_id) ;
```


Использование псевдонимов таблиц

Задание столбцов с именами таблиц требует времени, особенно если имена у таблиц длинные. Поэтому вместо имен таблиц можно использовать их псевдонимы. Как и псевдонимы столбцов, которые дают альтернативное имя столбцу, псевдонимы таблиц дают альтернативное имя таблице. Использование псевдонимов таблиц уменьшает объем кода SQL.

В примере на слайде обратите внимание на указание псевдонимов таблиц в команде `SELECT`: полное имя таблицы, пробел, а затем псевдоним таблицы. Для таблицы `EMPLOYEES` задан псевдоним *e* , а для таблицы `DEPARTMENTS` псевдоним *d*

- чем короче псевдонимы, тем удобнее читать код.
- Если в предложении `FROM` для указания таблицы используется псевдоним, этот псевдоним таблицы должен использоваться **вместо имени таблицы во всем** предложении `SELECT`.

Третий способ: JOIN ON
Лучший способ соединения

```
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);
```

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_ID	LOCATION_ID
200	Whalen	10	10	1700
201	Hartstein	20	20	1800
202	Fay	20	20	1800
124	Mourgos	50	50	1500
141	Rajs	50	50	1500
142	Davies	50	50	1500
143	Matos	50	50	1500

...

6 - 9 SQL

Третий способ: JOIN ON Лучший способ соединения

В приведенном примере с помощью выражения ON устанавливается соединение таблиц EMPLOYEES и DEPARTMENTS.

Предложение ON можно использовать для соединения столбцов с различными именами, в отличие от JOIN USING.

Вы видите явно по каким столбцам идёт соединение.

В таблицу могут быть добавлены столбцы и это не повлияет на работу команды, в отличие от NATURAL JOIN.

Имена столбцов по которым идёт соединение может быть любым, в отличие от JOIN USING.

Не будет ошибок, связанных с использованием квалификаторов (псевдонимов таблиц) в БД Oracle, в отличие от NATURAL JOIN и JOIN USING.

JOIN ON наиболее переносимый между БД разных производителей.

Соединения по нескольким столбцам

```
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)
AND    (e.manager_id = d.manager_id);
```

- JOIN ON позволяет явно задать правила соединения
- Отделение условий соединения (ON) от условий фильтрации (WHERE) упрощает чтение команды
- Предложение ON делает код более легким для понимания

Соединения по нескольким столбцам

Можно указать дополнительные условия соединения.

Для добавления условий в предложение ON используются фразу AND.

Другой способ - использование выражения WHERE для задания дополнительных условий:

```
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)
WHERE  (e.manager_id = d.manager_id);
```

Более того, можно обойтись без JOIN, результат будет тот же:

```
SELECT e.employee_id, e.last_name, e.department_id,
       d.department_id, d.location_id
FROM   employees e, departments d
WHERE  e.department_id = d.department_id
AND    (e.manager_id = d.manager_id);
```

Однако, условия соединения и фильтрации будут визуально неразличимы и смысл выражений непонятен.

Рекомендуется в ON ставить условия соединения, а в WHERE условия фильтрации.

Различение столбцов с одинаковыми именами

- Для различения одноименных столбцов из разных таблиц используются префиксы в виде имен таблиц.
- Использование префиксов таблиц увеличивает производительность.
- Одноименные столбцы из разных таблиц можно различать по их псевдонимам.
- Не используйте псевдонимы столбцов, которые указываются в предложении USING. Для таких столбцов не должны указываться псевдонимы в любом месте команды SQL.

6 - 11 SQL

Различение столбцов с одинаковыми именами

Во избежание путаницы именам столбцов в предложении WHERE должны предшествовать имена таблиц. Без такого префикса столбец DEPARTMENT_ID может принадлежать как таблице DEPARTMENTS, так и таблице EMPLOYEES. Для выполнения запроса необходимо добавить префиксы:

```
SELECT employees.employee_id, employees.last_name,
       departments.department_id, departments.location_id
FROM   employees JOIN departments
ON     employees.department_id = departments.department_id;
```

Для более компактной записи лучше использовать псевдонимы таблиц.

Если одноименных столбцов в таблицах нет, то для различения столбцов указание имен таблиц не требуется. Но благодаря префиксам увеличивается производительность, так как в этом случае БД точно знает, где искать столбцы.

Self Join
соединение таблицы самой с собой

```
SELECT e.last_name emp, m.last_name mgr
FROM   employees e JOIN employees m
ON     (e.manager_id = m.employee_id);
```

EMP	MGR
Hartstein	King
Zlotkey	King
Mourgos	King
De Haan	King
Kochhar	King

...

Соединение таблицы самой с собой

Предложение ON может быть также использовано для соединения столбцов с различными имена одной и той же или разных таблиц.

Псевдонимы таблиц обязательны.

В примере на слайде на основе столбцов EMPLOYEE_ID и MANAGER_ID выполняется соединение таблицы EMPLOYEES с собой.

Соединение нескольких таблиц

```
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;
```

EMPLOYEE_ID	CITY	DEPARTMENT_NAME
103	Southlake	IT
104	Southlake	IT
107	Southlake	IT
124	South San Francisco	Shipping
141	South San Francisco	Shipping
142	South San Francisco	Shipping
143	South San Francisco	Shipping
144	South San Francisco	Shipping

...

6 - 13 SQL

Соединение нескольких таблиц

Соединения обрабатываются слева направо. Поэтому в приведенном примере первым обрабатывается соединение между таблицами EMPLOYEES и DEPARTMENTS (EMPLOYEES JOIN DEPARTMENTS). В этом соединении указываются ссылки на столбцы таблиц EMPLOYEES и DEPARTMENTS, но в нем не может быть ссылок на столбцы таблицы LOCATIONS. Во втором условии соединения могут быть ссылки на столбцы всех трех таблиц.

Множественные соединения возможны и двумя предыдущими способами.

Не-эквисоединения

EMPLOYEES

LAST_NAME	SALARY
King	24000
Kochhar	17000
De Haan	17000
Hunold	9000
Ernst	6000
Lorentz	4200
Mourgos	5800
Rajs	3500
Davies	3100
Matos	2600
Vargas	2500
Zlotkey	10500
Abel	11000
Taylor	8600

…

JOB_GRADES

GRA	LOWEST_SAL	HIGHEST_SAL
A	1000	2999
B	3000	5999
C	6000	9999
D	10000	14999
E	15000	24999
F	25000	40000

Оклад в таблице EMPLOYEES находится между нижней и верхней границами окладов в таблице JOB_GRADES.

6 - 14 SQL

Не-эквисоединения

Не-эквисоединение - условие соединения, содержащее оператор, отличный от оператора равенства

Отношение между таблицами EMPLOYEES и JOB_GRADES не является эквисоединением, поскольку ни один столбец таблицы EMPLOYEES не соответствует точно какому-либо столбцу в таблице JOB_GRADES. Связь между двумя таблицами такова, что значение столбца SALARY в таблице EMPLOYEES находится между значениями столбцов LOWEST_SALARY и HIGHEST_SALARY таблицы JOB_GRADES. Для такого соединения оператор “=“ использоваться не может.

Выборка строк с помощью не-эквисоединений

```
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;
```

LAST_NAME	SALARY	GRA
Matos	2600	A
Vargas	2500	A
Lorentz	4200	B
Mourgos	5800	B
Rajs	3500	B
Davies	3100	B
Whalen	4400	B
Hunold	9000	C
Ernst	6000	C

...

6 - 15 SQL

Выборка строк с помощью не-эквисоединений

В примере на слайде не-эквисоединение создается для оценки категории заработной платы служащего. Она должна быть между любой парой нижней и верхней границ диапазона.

Важно отметить, что при выполнении запроса каждый служащий появляется только один раз, Ни один служащий в списке не повторяется. Для этого существует две причины:

- Ни одна строка в таблице JOB_GRADES не содержит категорий зарплат, перекрывающих друг друга. Т.е. заработная плата служащего может быть между нижним и верхнем значениями зарплат только одной строки таблицы JOB_GRADES .
- Заработная плата всех служащих находится в пределах, обозначенных таблицей JOB_GRADES. Т.е. ни один служащий не зарабатывает меньше, чем самое малое значение в столбце LOWEST_SAL , или больше, чем самое большое значение в столбце HIGHEST_SAL.

Примечание

Можно было бы использовать и другие операторы (например, <= и >=), но оператор BETWEEN является самым простым. Не забудьте, что в операторе BETWEEN сначала указывается нижняя граница, а затем – верхняя. Псевдонимы таблиц использованы не для различения столбцов, а для повышения производительности.

Внешние соединения

DEPARTMENTS

DEPARTMENT_NAME	DEPARTMENT_ID
Administration	10
Marketing	20
Shipping	50
IT	60
Sales	80
Executive	90
Accounting	110
Contracting	190

EMPLOYEES

DEPARTMENT_ID	LAST_NAME
90	King
90	Kochhar
90	De Haan
60	Hunold
60	Ernst
60	Lorentz
50	Mourgos
50	Rajs
50	Davies
50	Matos
50	Vargas
80	Zlotkey

...

В отделе 190 нет служащих.

6 - 16 SQL

Выборка записей, не имеющих прямых совпадений, с помощью внешних соединений

До сих пор мы рассматривали соединения, которые называются внутренними (INNER JOIN).

Если строка не удовлетворяет условию соединения, она не включается в результат запроса. Например, при эквисоединении таблиц EMPLOYEES и DEPARTMENTS не выводится отдел с номером 190, так как в таблице EMPLOYEES нет сотрудников с таким номером отдела. Вместо 8 записей будет выбрано 7.

Для вывода строки об отделе, в котором нет сотрудников, можно воспользоваться внешним соединением.

Сравнение внутреннего и внешнего (OUTER) соединений

- **Внутреннее соединение – это соединение двух таблиц, возвращающее только такие строки, которые соответствуют условию соединения.**
- **Соединение двух таблиц, возвращающее как строки внутреннего соединения, так и отсутствующие строки левой (правой) таблицы, – это левое (правое) внешнее соединение.**
- **Полное внешнее соединение возвращает результаты внутреннего соединения, а также левого и правого внешнего соединений.**

6 - 17 SQL

Сравнение внутреннего (INNER) и внешнего (OUTER) соединений

Внутреннее соединение (inner join) реализуется с помощью предложений NATURAL JOIN, USING или ON. Строки, не соответствующие условию соединения, не попадают в выходные результаты. Для вывода таких строк используется внешнее соединение (outer join). В результате внешнего соединения возвращаются строки, удовлетворяющие условию соединения, а также все такие строки одной таблицы, для которых нет соответствующих строк в другой таблице.

Три вида внешних соединений:

- LEFT OUTER (левое внешнее соединение);
- RIGHT OUTER (правое внешнее соединение);
- FULL OUTER (полное внешнее соединение).

Левое внешнее соединение

```
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);
```

LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
Whalen	10	Administration
Fay	20	Marketing
Hartstein	20	Marketing
...		
De Haan	90	Executive
Kochhar	90	Executive
King	90	Executive
Gietz	110	Accounting
Higgins	110	Accounting
Grant		

Пример левого внешнего соединения

В этом запросе возвращаются все строки таблицы EMPLOYEES, даже если они не соответствуют строкам таблицы DEPARTMENTS, так как таблица EMPLOYEES является левой таблицей во внешнем соединении.

"Пустые" строки дополняют строки таблицы справа.

В БД Oracle есть нестандартный унаследованный синтаксис. Эквивалент запроса на слайде в таком синтаксисе:

```
SELECT e.last_name, d.department_id, d.department_name
FROM   employees e , departments d
WHERE  e.department_id  = d.department_id (+);
```

Правое внешнее соединение

```
SELECT e.last_name, e.department_id, d.department_name
FROM   employees e RIGHT OUTER JOIN departments d
ON     (e.department_id = d.department_id) ;
```

LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
Whalen	10	Administration
Fay	20	Marketing
Hartstein	20	Marketing
Davies	50	Shipping
...		
Kochhar	90	Executive
Gietz	110	Accounting
Higgins	110	Accounting
	190	Contracting

20 rows selected.

Пример правого внешнего соединения

В этом запросе возвращаются все строки таблицы DEPARTMENTS, даже если они не соответствуют строкам таблицы EMPLOYEES, так как таблица DEPARTMENTS является правой таблицей во внешнем соединении.

"Пустые" строки дополняют строки таблицы слева.

Левое внешнее соединение и правое внешнее соединение можно преобразовать друг в друга, поменяв имена таблиц местами во фразе FROM.

В БД Oracle есть нестандартный унаследованный синтаксис. Эквивалент запроса на слайде в таком синтаксисе:

```
SELECT e.last_name, d.department_id, d.department_name
FROM   employees e , departments d
WHERE  e.department_id  (+) = d.department_id;
```

Полное внешнее соединение

```
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);
```

LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
Whalen	10	Administration
Fay	20	Marketing
Hartstein	20	Marketing
...		
King	90	Executive
Gietz	110	Accounting
Higgins	110	Accounting
Grant		
	190	Contracting

21 rows selected.

6 - 20 SQL

Пример полного внешнего соединения

В этом запросе возвращаются все строки таблицы DEPARTMENTS, даже если они не соответствуют строкам таблицы EMPLOYEES. В нем также выбираются все строки таблицы DEPARTMENTS, даже если они не соответствуют строкам таблицы EMPLOYEES.

Практическое задание 6

- Соединение таблиц с использованием эквисоединения
- Выполнение внешних соединений и рефлексивных соединений
- Включение дополнительных условий

6 - 21 SQL

Практическое задание 6

Это практическое занятие позволяет приобрести навыки выборки данных из более, чем одной таблицы с использованием стандарта SQL.

Использование групповых функций

Агрегирование

- Общие сведения об имеющихся групповых функциях
- Использование групповых функций
- Вывод данных по группам с помощью выражения `GROUP BY`
- Включение и исключение групп с помощью выражения `HAVING`

7 - 2 SQL

Агрегирование

Это еще один урок, посвященный функциям. Основное внимание сосредоточено на функциях для получения сводной информации по группам строк (например, средних значений). Обсуждаются методы группировки строк таблицы и задания критериев поиска для групп строк.

Группирование часто называют *агрегирование*, а групповые функции *агрегатами*.

Использование функций AVG и SUM

Функции AVG и SUM применяются к столбцам с числовыми данными.

```
SELECT AVG(salary), MAX(salary),
       MIN(salary), SUM(salary)
FROM   employees
WHERE  job_id LIKE '%REP%';
```

AVG(SALARY)	MAX(SALARY)	MIN(SALARY)	SUM(SALARY)
8150	11000	6000	32600

7 - 3 SQL

Использование функций AVG и SUM

Функции AVG, SUM, MIN и MAX применяются к столбцам, в которых можно хранить цифровые данные. В примере на слайде вычисляются средний, самый высокий, самый низкий оклад и сумма окладов всех торговых представителей.

Использование функций MIN и MAX

Функции MIN and MAX применяются к данным числового, символьного типа данных и датам.

```
SELECT MIN(hire_date), MAX(hire_date)
FROM   employees;
```

MIN(HIRE_	MAX(HIRE_
17-JUN-87	29-JAN-00

7 - 4 SQL

Использование функций MIN и MAX

Функции MIN и MAX применяются к данным числового, символьного типа данных и датам. Пример показывает поиск даты приема служащего, который работает дольше всех, и даты приема служащего, принятого самым последним.

В следующем примере выводятся первый и последний служащие в алфавитном списке служащих:

```
SELECT MIN(last_name), MAX(last_name)
FROM   employees;
```

MIN(LAST_NAME)	MAX(LAST_NAME)
Abel	Zlotkey

Примечание

Функции AVG, SUM, VARIANCE и STDDEV применимы только к численным данным.

Использование функции COUNT

COUNT(*) возвращает количество строк в таблице :

1

```
SELECT COUNT(*)
FROM   employees
WHERE  department_id = 50;
```

COUNT(*)
5

COUNT(*выражение*) возвращает количество строк с определенными значениями (не NULL) для *выражения*:

2

```
SELECT COUNT(commission_pct)
FROM   employees
WHERE  department_id = 80;
```

COUNT(COMMISSION_PCT)
3

Функция COUNT

Имеется три формата *функции COUNT*:

- COUNT(*)
- COUNT(*выражение*)
- COUNT(DISTINCT *выражение*)

COUNT(*) возвращает количество строк в таблице, которые удовлетворяют ограничениям, заданным в команде SELECT. При этом учитываются и строки-дубликаты и строки с неопределенными значениями. Если команда SELECT включает выражение WHERE, COUNT(*) возвращает количество строк, удовлетворяющее условию в предложении WHERE.

Напротив COUNT(*выражение*) возвращает количество строк с определенными значениями, заданными *выражением*.

COUNT(**DISTINCT** *выражение*) возвращает количество уникальных, определенных значений в столбце, заданном *выражением*.

COUNT(*) может работать менее эффективно, чем COUNT(первичный_ключ).

Примеры на слайде

1. Подсчитывается количество служащих в отделе 50.
2. Вычисляется количество служащих отдела 80, которые могут получать комиссионные.

Использование ключевого слова DISTINCT

- `COUNT(DISTINCT` *выражение*`)` возвращает количество уникальных определенных значений *выражения*
- Вывод количества уникальных номеров отделов из таблицы `EMPLOYEES`:

```
SELECT COUNT(DISTINCT department_id)
FROM   employees;
```

COUNT(DISTINCTDEPARTMENT_ID)
7

Использование ключевого слова DISTINCT

Для того, чтобы не учитывать при подсчете дублирующие значения столбца, применяется ключевое слово `DISTINCT`.

В примере на слайде подсчитывается количество отделов на основе таблицы `EMPLOYEES`.

В БД Oracle версии 12.2 есть функция APPROX_COUNT_DISTINCT и другие, выполняющиеся быстро и удобные для приблизительных вычислений в аналитических запросах.

Групповые функции и NULL

Групповые функции игнорируют неопределенные значения в столбцах:

1

```
SELECT AVG(commission_pct)
FROM   employees;
```

AVG(COMMISSION_PCT)
.2125

Функция NVL заставляет групповые функции включать неопределенные значения:

2

```
SELECT AVG(NVL(commission_pct, 0))
FROM   employees;
```

AVG(NVL(COMMISSION_PCT,0))
.0425

7 - 7 SQL

Групповые функции и NULL

Все групповые функции, кроме COUNT(*), игнорируют неопределенные значения в столбце.

Функция NVL заставляет групповые функции включать неопределенные значения.

Примеры

- В первом примере на слайде среднее вычисляется *только* по строкам, где столбец COMMISSION_PCT содержит действительное значение. Среднее вычисляется как частное от деления общей суммы комиссионных, выплаченных всем служащим, на количество служащих, получающих комиссионные (четверо).
- Во втором примере на слайде среднее вычисляется по *всем* строкам, включая строки с неопределенным значением в столбце COMMISSION_PCT. Среднее в этом случае – это частное от деления общей суммы комиссионных, выплаченных служащим, на общее количество служащих в компании (20).

Создание групп строк

EMPLOYEES

DEPARTMENT_ID	SALARY
10	4400
20	13000
20	6000
50	5800
50	3500
50	3100
50	2500
50	2600
60	9000
60	6000
60	4200
80	10500
80	8600
80	11000
90	24000
90	17000

...

4400

9500

3500

6400

10033

Средний оклад в таблице EMPLOYEES по каждому отделу

DEPARTMENT_ID	AVG(SALARY)
10	4400
20	9500
50	3500
60	6400
80	10033.3333
90	19333.3333
110	10150
	7000

7 - 8 SQL

Создание групп строк

До сих пор все групповые функции обрабатывали таблицу как одну большую группу информации. Но иногда требуется разделить таблицу на более мелкие группы. Сделать это можно с помощью выражения `GROUP BY`.

Выражение GROUP BY

```
SELECT        [столбец,] групп_функция(столбец), ...
FROM          таблица
[WHERE        условие]
[GROUP BY выражение]
[ORDER BY столбец];
```

группы.

Псевдонимы (алиасы) столбцов нельзя использовать в GROUP BY

Выражение GROUP BY

С помощью *выражения* GROUP BY можно разделить строки таблицы на группы. Затем можно использовать групповые функции для получения сводной информации по каждой группе.

выражение задает столбцы, на основе значений которых группируются строки

Указания

- Если групповая функция задана в предложении SELECT, получить одновременно индивидуальный результат можно *только в случае, если* отдельный столбец указан в предложении GROUP BY. Если список столбцов отсутствует, выдается сообщение об ошибке.
- WHERE исключает строки **до** их группирования.
- Список столбцов (выражений) в предложении GROUP BY обязателен.
- **Псевдонимы (алиасы) столбцов нельзя использовать в GROUP BY, в отличие от ORDER BY**

Использование выражения GROUP BY

Все столбцы, которые входят в список SELECT и к которым не применяются групповые функции, должны быть указаны в предложении GROUP BY.

```
SELECT   department_id, AVG(salary)
FROM     employees
GROUP BY department_id ;
```

DEPARTMENT_ID	AVG(SALARY)
10	4400
20	9500
50	3500
60	6400
80	10033.3333
90	19333.3333
110	10150
	7000

7 - 10 SQL

Использование выражения GROUP BY

Если используется выражение GROUP BY, все столбцы из списка SELECT, к которым не применяются групповые функции, должны быть включены в выражение GROUP BY. В примере на слайде выдаются номера отделов и средний оклад по каждому отделу. Команда SELECT с выражением GROUP BY делает следующее.

- выражение SELECT задает выбираемые столбцы:
  - столбец номеров отделов из таблицы EMPLOYEES ;
  - среднее всех окладов в группе, заданной с помощью выражения GROUP BY .
- выражение FROM задает таблицы, к которым должна обратиться база данных (в данном случае таблицу EMPLOYEES).
- выражение WHERE задает критерии отбора строк. Т.к. выражение WHERE отсутствует, по умолчанию выбираются все строки.
- выражение GROUP BY указывает, как должны быть сгруппированы строки. Строки группируются по отделам, чтобы функция AVG, применяемая к столбцу окладов, вычислила *средний оклад по каждому отделу.*

Использование выражения GROUP BY

Столбец, указанный в GROUP BY, может отсутствовать в списке SELECT.

```
SELECT   AVG(salary)
FROM     employees
GROUP BY department_id ;
```

AVG(SALARY)
4400
9500
3500
6400
10033.3333
19333.3333
10150
7000

7 - 11 SQL

Использование выражения GROUP BY

Столбец, заданный в предложении GROUP BY, может отсутствовать в предложении SELECT. В примере на слайде средние оклады по отделам выводятся без соответствующих номеров отделов. Но без номеров отделов результат трудно понять.

Можно использовать групповую функцию в предложении ORDER BY:

```
SELECT   department_id, AVG(salary)
FROM     employees
GROUP BY department_id
ORDER BY AVG(salary);
```

Группировка по нескольким столбцам

EMPLOYEES

DEPARTMENT_ID	JOB_ID	SALARY
90	AD_PRES	24000
90	AD_VP	17000
90	AD_VP	17000
60	IT_PROG	9000
60	IT_PROG	6000
60	IT_PROG	4200
50	ST_MAN	5800
50	ST_CLERK	3500
50	ST_CLERK	3100
50	ST_CLERK	2600
50	ST_CLERK	2500
80	SA_MAN	10500
80	SA_REP	11000
80	SA_REP	8600
...		
20	MK_REP	6000
110	AC_MGR	12000
110	AC_ACCOUNT	8300

Просуммировать оклады в таблице EMPLOYEES по каждой должности внутри каждого отдела

DEPARTMENT_ID	JOB_ID	SUM(SALARY)
10	AD_ASST	4400
20	MK_MAN	13000
20	MK_REP	6000
50	ST_CLERK	11700
50	ST_MAN	5800
60	IT_PROG	19200
80	SA_MAN	10500
80	SA_REP	19600
90	AD_PRES	24000
90	AD_VP	34000
110	AC_ACCOUNT	8300
110	AC_MGR	12000
	SA_REP	7000

7 - 12 SQL

Группировка по нескольким столбцам

Иногда требуются результаты по группам внутри групп. Отчет на слайде показывает общую сумму заработной платы, выплаченной по каждой должности в каждом отделе.

Строки таблицы EMPLOYEES сначала группируются по номерам отделов, а затем внутри этих групп – по должностям. Например, объединяются в группу четыре клерка (ST_CLERK) из отдела 50 и для них выдается единый результат: общая сумма окладов для всех клерков отдела.

Использование GROUP BY с несколькими столбцами

```
SELECT   department_id dept_id, job_id, SUM(salary)
FROM     employees
GROUP BY department_id, job_id ;
```

DEPT_ID	JOB_ID	SUM(SALARY)
10	AD_ASST	4400
20	MK_MAN	13000
20	MK_REP	6000
50	ST_CLERK	11700
50	ST_MAN	5800
60	IT_PROG	19200
80	SA_MAN	10500
80	SA_REP	19600
90	AD_PRES	24000
90	AD_VP	34000
110	AC_ACCOUNT	8300
110	AC_MGR	12000
	SA_REP	7000

Использование GROUP BY с несколькими столбцами

Чтобы получить результаты по группам и подгруппам, нужно указать несколько столбцов в предложении GROUP BY. Последовательностью столбцов в предложении GROUP BY можно задать порядок сортировки строк по умолчанию. Команда SELECT на слайде, содержащая выражение GROUP BY, делает следующее:

- выражение SELECT задает выбираемые столбцы:
  - номера отделов из таблицы EMPLOYEES ;
  - должности из таблицы EMPLOYEES ;
  - сумму всех окладов в группе, заданной с помощью выражения GROUP BY .
- выражение FROM задает таблицы, к которым должна обратиться база данных: в данном случае – EMPLOYEES.
- выражение GROUP BY указывает, как должны быть сгруппированы строки:
  - сначала по номерам отделов;
  - затем по должностям внутри отделов.

Таким образом функция SUM применяется к столбцу окладов по каждой должности внутри каждого отдела.

Ошибочные запросы с групповыми функциями

Все столбцы и выражения из списка SELECT, не являющиеся групповой функцией, должны быть включены в выражение GROUP BY :

```
SELECT department_id, COUNT(last_name)
FROM   employees;
```

```
SELECT department_id, COUNT(last_name)
       *
ERROR at line 1:
ORA-00937: not a single-group group function
```

В предложении GROUP BY недостает столбца

7 - 14 SQL

Ошибочные запросы с групповыми функциями

Если в одной и той же команде SELECT указываются отдельные элементы (DEPARTMENT_ID) и групповые функций (COUNT), выражение GROUP BY со списком отдельных элементов обязательно (в рассматриваемом случае DEPARTMENT_ID). Если выражение GROUP BY отсутствует, выдается сообщение об ошибке not a single-group group function (“групповая функция для более, чем одной группы“), и столбец, вызвавший ошибку, помечается звездочкой (*). Для устранения ошибки добавьте выражение GROUP BY:

```
SELECT   department_id, count(last_name)
FROM     employees
GROUP BY department_id;
```

Любой столбец или выражение из списка SELECT, к которым не применяется групповая функция, обязательно указывается в предложении GROUP BY.

Ошибочные запросы с групповыми функциями

- выражение WHERE для исключения групп не используется
- Для исключения некоторых групп следует пользоваться выражением HAVING
- Нельзя использовать групповые функции в предложении WHERE

```
SELECT   department_id, AVG(salary)
FROM     employees
WHERE    AVG(salary) > 8000
GROUP BY department_id;
```

```
WHERE  AVG(salary) > 8000
       *
ERROR at line 3:
ORA-00934: group function is not allowed here
```

выражение WHERE для исключения групп не используется

7 - 15 SQL

Ошибочные запросы с групповыми функциями

выражения WHERE для исключения групп не используется. Вышеуказанная команда SELECT вызовет ошибку, т.к. для вывода средних окладов только тех отделов, где средний оклад превышает 8000 долларов, используется выражение WHERE.

Для исключения групп следует использовать выражение HAVING:

```
SELECT   department_id, AVG(salary)
FROM     employees
HAVING   AVG(salary) > 8000
GROUP BY department_id;
```

Вложение групповых функций

Вывод максимального среднего оклада

Неверно:

```
SELECT   MAX(AVG(salary))
FROM     employees
GROUP BY department_id;
```

Верно:

```
SELECT MAX(avgsal)
     FROM (SELECT AVG(salary) avgsal
           FROM employees GROUP BY department_id) a;
```


Вложение групповых функций

В БД Oracle допускается вкладывать групповые функции друг в друга. В примере на слайде вычисляется максимальный средний оклад. Однако стандарт SQL запрещает это

Также стандарт запрещает использовать запрос в качестве аргумента групповой функции.

Команду на слайде можно переписать на аналогичную, соответствующую стандарту с использованием подзапроса:

```
SELECT MAX(avgsal)
   FROM (SELECT AVG(salary) avgsal
         FROM employees GROUP BY department_id) a;
```

Также можно использовать групповую функцию в качестве оконной (будут рассматриваться дальше):

```
SELECT DISTINCT MAX(AVG(salary)) OVER()
FROM     employees
GROUP BY department_id;
```

Исключение групп

EMPLOYEES

DEPARTMENT_ID	SALARY
90	24000
90	17000
90	17000
60	9000
60	6000
60	4200
50	5800
50	3500
50	3100
50	2600
50	2500
80	10500
80	11000
80	8600
...	
20	6000
110	12000
110	8300

максимальный оклад в отделе превышает 10000

DEPARTMENT_ID	MAX(SALARY)
20	13000
80	11000
90	24000
110	12000

Исключение групп

Подобно тому, как выражение WHERE используется для исключения строк, выражение HAVING используется для исключения групп. Чтобы выяснить максимальный оклад по каждому отделу, но вывести результат только по тем отделам, где он превышает 10000, необходимо сделать следующее:

1. Найти максимальный оклад по каждому отделу путем группировки строк по номерам отделов.
2. Исключить из отчета те группы, где максимальный оклад не превышает 10000.

Исключение групп: выражение HAVING

Для исключения групп пользуйтесь выражением HAVING:

1. **Строки группируются.**
2. **Применяется групповая функция.**
3. **Выводятся группы, удовлетворяющие условию в предложении HAVING.**

```
SELECT   [столбец,] групп_функция(столбец), ...
FROM          таблица
[WHERE   условие]
[GROUP BY     выражение_группировки]
[HAVING  ограничивающее_условие]
[ORDER BY     столбец];
```

7 - 18 SQL

Исключение групп: выражение HAVING

С помощью выражения HAVING из выходных данных исключаются некоторые группы. Таким образом исключение групп производится по агрегированной информации.

ограничивающее_условие вывод ограничивается группами строк, удовлетворяющими заданному условию

БД обрабатывает выражение HAVING следующим образом.

1. Строки группируются.
2. К группе применяется групповая функция.
3. Выводятся группы, удовлетворяющие критериям в предложении HAVING.

выражение HAVING может предшествовать предложению GROUP BY, но логичнее сделать выражение GROUP BY первым. Образование групп и вычисление групповых функций происходят до того, как к группам из списка SELECT применяется выражение HAVING.

Использование выражения HAVING

```
SELECT   department_id, MAX(salary)
FROM     employees
GROUP BY department_id
HAVING   MAX(salary)>10000 ;
```

DEPARTMENT_ID	MAX(SALARY)
20	13000
80	11000
90	24000
110	12000

Использование выражения HAVING

В примере на слайде выводятся номера отделов и максимальный оклад только тех отделов, где он превышает 10000.

выражение GROUP BY можно используется без групповой функции в списке SELECT.

Для исключения строк после применения групповой функции требуются выражения GROUP BY и HAVING.

Использование выражения HAVING

```
SELECT   job_id, SUM(salary) PAYROLL
FROM     employees
WHERE    job_id NOT LIKE '%REP%'
GROUP BY job_id
HAVING   SUM(salary) > 13000
ORDER BY SUM(salary);
```

JOB_ID	PAYROLL
IT_PROG	19200
AD_PRES	24000
AD_VP	34000

7 - 20 SQL

Использование выражения HAVING

Пример на слайде показывает вывод должностей и общей суммы окладов за месяц по каждой должности, если эта общая сумма превышает 13000. Из выходных данных исключаются данные о торговых представителях. Список сортируется по возрастанию общей суммы окладов

Итоги

- Использование групповых функций COUNT, MAX, MIN, AVG.
- Запросы, использующие выражение GROUP BY.
- Запросы, использующие выражение HAVING.

```
SELECT      [столбец,] групп_функция(столбец), ...
FROM        таблица
[WHERE      условие]
[GROUP BY выражение_группировки]
[HAVING    условие_группировки]
[ORDER BY столбец];
```

7 - 21 SQL

Итоги

В SQL имеются групповые функции. К ним, например, относятся AVG, COUNT, MAX, MIN, SUM, STDDEV и VARIANCE.

выражение GROUP BY позволяет создавать подгруппы. Исключать группы можно с помощью выражения HAVING.

В команде SELECT выражения HAVING и GROUP BY должны следовать за выражением WHERE. Порядок, в котором выражения HAVING и GROUP BY следуют за выражением WHERE, неважен. Располагайте выражение ORDER BY последним.

БД обрабатывает выражения в следующем порядке.

1. Если имеется выражение WHERE, БД выявляет строки-кандидаты.
2. БД выявляет группы, заданные выражением GROUP BY.
3. выражение HAVING исключает из выходных данных группы, не удовлетворяющие его критериям.

Практическое задание 7

- Написание запросов с использованием групповых функций.
- Разбиение строк на группы для получения более, чем одного результата.
- Исключение групп с помощью выражения `HAVING`.

Практическое задание 7

Это занятие позволяет приобрести навыки использования групповых функций и выборки групп данных.

8

Подзапросы

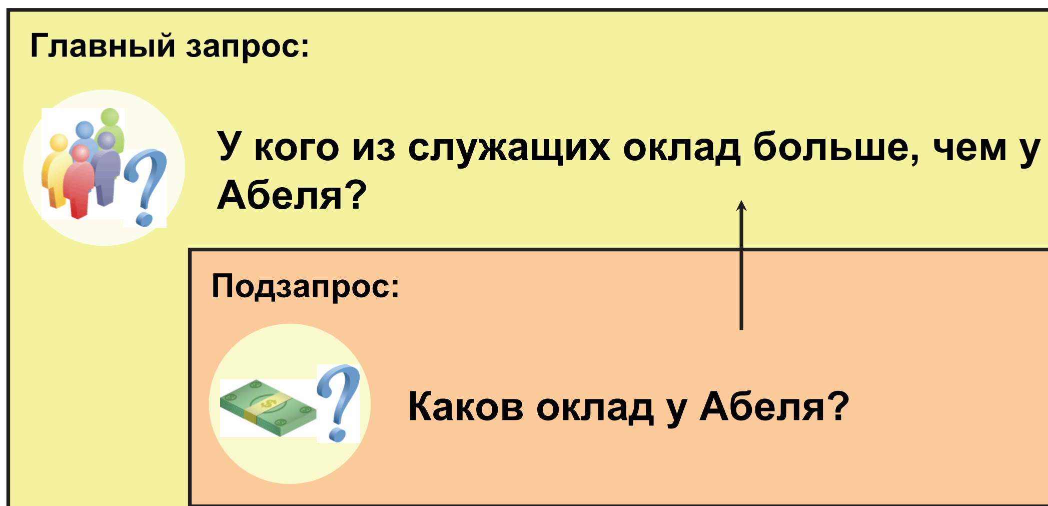
Рассматриваемые вопросы

- **Определение подзапросов**
- **Типы проблем, решаемых с помощью подзапросов**
- **Типы подзапросов**
- **Написание однострочных и многострочных подзапросов**

Цели урока

Этот урок посвящен более сложному использованию команды SELECT. Для выборки значений по неизвестному условию можно создать подзапрос в предложении WHERE команды SQL.

Задачи, решаемые подзапросами



Задачи, решаемые подзапросами

Предположим, что нужно выяснить, у кого оклад больше, чем у Абеля.

Для этого необходимы *два* запроса: один для выяснения оклада Абеля и другой для выяснения того, кто получает больше.

Для решения этой задачи можно создать один запрос внутри другого.

Значение, возвращаемое внутренним запросом (inner query) или подзапросом (subquery), используется внешним (outer query) или главным запросом (main query). Использовать подзапрос – это то же самое, что последовательно выполнить два запроса, использовав результат первого в качестве критерия поиска во втором.

Синтаксис подзапросов

```
SELECT    что
FROM      откуда
WHERE     выражение  оператор
                     (SELECT       что
                      FROM         откуда);
```

- **Подзапрос (внутренний запрос) выполняется один раз до главного запроса.**
- **Результат подзапроса используется главным запросом (внешним запросом).**

8 - 4 SQL

Синтаксис подзапросов

Подзапрос – это команда SELECT, включенная в предложение другой команды SELECT. С помощью подзапросов можно создавать очень мощные команды из простых. Они очень полезны в случае, если выборка строк из таблицы производится по условию, зависимому от данных в самой таблице.

Подзапрос можно использовать в следующих предложениях языка SQL:

- WHERE
- HAVING
- FROM

Синтаксис:

оператор оператор сравнения, например, >, = или IN

Примечание

Операторы сравнения делятся на две категории: однострочные (>, =, >=, <, <>, <=) и многострочные (IN, ANY, ALL).

Подзапрос часто называют вложенной (nested) командой SELECT, подкомандой SELECT или внутренней (inner) командой SELECT. Обычно подзапрос выполняется первым, и его результат используется для определения условия выборки в главном или внешнем запросе.

Использование подзапроса

```
SELECT last_name
FROM   employees
WHERE  salary >
               (SELECT salary
                FROM   employees
                WHERE  last_name = 'Abel');
```

11000

LAST_NAME
King
Kochhar
De Haan
Hartstein
Higgins

8 - 5 SQL

Использование подзапроса

На слайде внутренний запрос определяет оклад служащего Abel. Внешний запрос использует результат внутреннего запроса для вывода списка всех служащих, зарабатывающих больше этой суммы.

Рекомендации по использованию подзапросов

- Подзапрос должен быть заключен в скобки.
- Подзапрос должен находиться справа от оператора сравнения.
- В однострочных подзапросах используются однострочные операторы.
- В многострочных подзапросах используются многострочные операторы.

8 - 6 SQL

Рекомендации по использованию подзапросов

- Подзапрос должен быть заключен в скобки.
- Подзапрос должен находиться справа от оператора сравнения.
- `ORDER BY` может использоваться в подзапросе для проведения “TOP-N” анализа.
- В подзапросах используются операторы сравнения двух типов: однострочные и многострочные.

Типы подзапросов

- Однострочный подзапрос

- Многострочный подзапрос

8 - 7 SQL

Типы подзапросов

- Однострочные подзапросы (Single-row subqueries) – запросы, в которых внутренняя команда `SELECT` возвращает только одну строку
- Многострочные подзапросы (Multiple-row subqueries) – запросы, в которых внутренняя команда `SELECT` возвращает более одной строки

Существуют также многостолбцовые подзапросы, в которых внутренняя команда `SELECT` возвращает более одного столбца.

Однострочные подзапросы

- **Возвращают только одну строку**
- **Используют однострочные операторы сравнения**

Оператор	Значение
=	Равно
>	Больше, чем
>=	Больше либо равно
<	Меньше, чем
<=	Меньше либо равно
<>	Не равно

Однострочные подзапросы

Внутренняя команда SELECT, возвращающая только одну строку, называется однострочным подзапросом (single-row subquery). В подзапросах такого типа используются однострочные операторы (single-row operator). На слайде показан список однострочных операторов.

Пример

Вывод списка служащих с такой же должностью, как у служащего под номером 141:

```
SELECT last_name, job_id
FROM   employees
WHERE  job_id =
                (SELECT job_id
                 FROM   employees
                 WHERE  employee_id = 141);
```

Выполнение однострочных подзапросов

```
SELECT last_name, job_id, salary
FROM   employees
WHERE  job_id =
                (SELECT job_id
                 FROM   employees
                 WHERE  employee_id = 141)
AND    salary >
                (SELECT salary
                 FROM   employees
                 WHERE  employee_id = 143);
```

ST_CLERK

2600

LAST_NAME	JOB_ID	SALARY
Rajs	ST_CLERK	3500
Davies	ST_CLERK	3100

8 - 9 SQL

Выполнение однострочных подзапросов

Команду SELECT можно рассматривать как блок запроса. На слайде показан запрос для вывода списка служащих, которые работают в той же должности, что служащий с номером 141, и имеют оклад больше, чем у служащего с номером 143.

Пример состоит из трех блоков запроса: одного внешнего и двух внутренних. Сначала выполняются внутренние блоки. Их результаты – ST_CLERK и 2600. Затем выполняется внешний блок. Для создания условий поиска он использует результаты внутренних запросов.

Оба внутренних запроса возвращают по одному значению: ST_CLERK и 2600. Поэтому данная команда SQL называется однострочным подзапросом.

Внешний и внутренний подзапросы могут выбирать данные из разных таблиц.

Использование групповых функций в подзапросах

```
SELECT last_name, job_id, salary
FROM   employees
WHERE  salary =
                   (SELECT MIN(salary)
                    FROM   employees);
```


LAST_NAME	JOB_ID	SALARY
Vargas	ST_CLERK	2500

Использование групповых функций в подзапросах

В подзапросе можно использовать групповую функцию для получения результата в виде одной строки. Подзапрос заключается в круглые скобки и помещается после оператора сравнения.

В вышеуказанном примере показан вывод фамилий, должностей и окладов всех служащих, чей оклад равен минимальному. Групповая функция `MIN` возвращает во внешний запрос только одно значение (2500).

Предложение HAVING с подзапросами

- Сначала выполняется подзапрос
- Результат возвращается в предложение HAVING главного запроса

```
SELECT   department_id, MIN(salary)
FROM     employees
GROUP BY department_id
HAVING   MIN(salary) >
                        (SELECT MIN(salary)
                         FROM   employees
                         WHERE  department_id = 50);
```


Предложение HAVING с подзапросами

Подзапросы можно использовать не только в предложении WHERE, но и в предложении HAVING. БД выполняет подзапрос, и результаты возвращаются в предложение HAVING главного запроса.

Команда SQL на слайде выводит список отделов, минимальные оклады в которых превышают минимальный оклад в отделе 50.

Что неправильно в этой команде?

```
SELECT employee_id, last_name
FROM   employees
WHERE  salary =
                 (SELECT   MIN(salary)
                  FROM     employees
                  GROUP BY department_id);
```

```
ERROR at line 4:
ORA-01427: single-row subquery returns more than
one row
```

Однострочный оператор с многострочным подзапросом

Ошибки с подзапросами

Самая распространенная ошибка – это когда однострочный подзапрос возвращает более одной строки.

В команде SQL на слайде подзапрос содержит предложение `GROUP BY (department_id)`, что предполагает возврат нескольких строк – по одной на каждую найденную группу. В данном случае результатами подзапроса будут 4400, 6000, 2500, 4200, 7000, 17000 и 8300.

Внешний запрос берет результаты подзапроса (4400, 6000, 2500, 4200, 7000, 17000, 8300) и использует их в предложении `WHERE`. Предложение `WHERE` содержит оператор равенства "=" – однострочный оператор, ожидающий только одно значение. Оператор равенства не может принять от подзапроса более одного значения и, следовательно, выдаст ошибку.

Для исправления ошибки замените оператор " =" оператором `IN`.

Будет ли выполнена эта команда?

```
SELECT last_name, job_id
FROM   employees
WHERE  job_id =
                (SELECT job_id
                 FROM   employees
                 WHERE  last_name = 'Haas');
```

```
no rows selected
```

Подзапрос не возвращает никаких значений.

Проблемы с подзапросами

Самая распространенная проблема – это когда внутренний запрос не возвращает ни одной строки.

В примере на слайде команда SQL содержит предложение WHERE (last_name = 'Haas'). Очевидно, цель состояла в поиске служащего по фамилии Haas. Команда кажется правильной, но при выполнении не возвращает ни одной строки.

Проблема в том, что фамилия Haas указана неправильно. Служащего с такой фамилией нет. Поэтому подзапрос ничего не возвращает. Внешний запрос берет результаты подзапроса (неопределенное значение) и использует их в своем предложении WHERE. Внешний запрос не находит ни одного служащего с должностью, равной неопределенному значению, и ничего не возвращает. Если существует должность с неопределенным значением, внешний запрос все равно не вернет строки, так как результат сравнения двух неопределенных значений – это тоже неопределенное значение и поэтому условие WHERE неверно.

Многострочные подзапросы

- Возвращают более одной строки
- Используют многострочные операторы сравнения

Оператор	Значение
IN	Равно любому члену списка
ANY	Сравнение значения с любым значением, возвращаемым подзапросом
ALL	Сравнение значения с каждым значением, возвращаемым подзапросом

Многострочные подзапросы

Подзапросы, возвращающие более одной строки, называются *многострочными*. Вместо однострочного оператора в них используется многострочный. Многострочный оператор ожидает одно или более значений:

```
SELECT last_name, salary, department_id
FROM   employees
WHERE  salary IN (SELECT   MIN(salary)
                  FROM     employees
                  GROUP BY department_id);
```

Пример

Найдите служащих, оклад которых равен минимальному окладу в отделах.

Сначала выполняется внутренний запрос, возвращающий строки: `2500, 4200, 4400, 6000, 7000, 8300, 8600, 17000`. Затем обрабатывается главный блок. Результаты внутреннего запроса используются при этом для завершения условия поиска в главном запросе. Фактически, для БД главный запрос выглядит примерно так:

```
SELECT last_name, salary, department_id
FROM   employees
WHERE  salary IN (2500, 4200, 4400, 6000, 7000, 8300,
 8600, 17000);
```

Использование оператора ANY в многострочных подзапросах

```
SELECT employee_id, last_name, job_id, salary
FROM   employees        9000, 6000, 4200
WHERE  salary < ANY
                    (SELECT salary
                     FROM   employees
                     WHERE  job_id = 'IT_PROG')
AND    job_id <> 'IT_PROG';
```

EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
124	Mourgos	ST_MAN	5800
141	Rajs	ST_CLERK	3500
142	Davies	ST_CLERK	3100
143	Matos	ST_CLERK	2600
144	Vargas	ST_CLERK	2500

…

Многострочные подзапросы (продолжение)

Оператор ANY (и его синоним SOME) сравнивает значение с *любым* значением, возвращаемым подзапросом. Запрос на слайде возвращает список служащих, которые не являются программистами подразделения информационных технологий и оклады которых меньше, чем у любого программистами. Максимальный оклад программиста – 9000 долларов. Команда SQL возвращает список служащих, которые не являются программистами, но зарабатывают менее 9000 долларов.

<ANY означает "меньше, чем максимум".

>ANY означает "больше, чем минимум".

=ANY – эквивалент IN.

Использование оператора ALL в многострочных подзапросах

```
SELECT employee_id, last_name, job_id, salary
FROM   employees        9000, 6000, 4200
WHERE  salary < ALL
                    (SELECT salary
                     FROM   employees
                     WHERE  job_id = 'IT_PROG')
AND    job_id <> 'IT_PROG';
```

EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
141	Rajs	ST_CLERK	3500
142	Davies	ST_CLERK	3100
143	Matos	ST_CLERK	2600
144	Vargas	ST_CLERK	2500

8 - 16 SQL

Многострочные подзапросы (продолжение)

Оператор ALL сравнивает значение с *каждым* значением, возвращаемым подзапросом. Запрос на слайде возвращает список служащих, которые не являются программистами (идентификатором должности IT_PROG) и оклады которых меньше, чем у всех программистов.

>ALL означает “больше, чем максимум“.

<ALL означает “меньше, чем минимум“.

Оператор NOT не может использоваться с операторами ANY и ALL.

Неопределенные значения в подзапросе

```
SELECT emp.last_name
FROM   employees emp
WHERE  emp.employee_id NOT IN
                           (SELECT mgr.manager_id
                            FROM   employees mgr);

no rows selected
```

8 - 17 SQL

Неопределенные значения в результатах подзапроса

На слайде делается попытка вывода фамилий всех служащих, не имеющих подчиненных. Логически эта команда SQL должна была бы вернуть 12 строк, но одно из значений, возвращаемых внутренним запросом, является неопределенным значением, и, следовательно, запрос в целом не возвращает ни одной строки. Это объясняется тем, что все условия, где производится сравнение с неопределенным значением, дают неопределенное значение. Поэтому никогда не используйте оператор NOT IN, если существует вероятность того, что подзапрос вернет неопределенное значение. Оператор NOT IN эквивалентен <> ALL.

Следует отметить, что если используется оператор IN, то проблем с неопределенными значениями, возвращаемыми подзапросом, не возникает. Оператор IN эквивалентен условию =ANY. Например, для вывода всех служащих, имеющих подчиненных, используйте следующую команду SQL:

```
SELECT emp.last_name
FROM   employees emp
WHERE  emp.employee_id  IN
                           (SELECT mgr.manager_id
                            FROM   employees mgr);
```

Для того, чтобы вывести всех служащих, не имеющих подчиненных, в подзапрос можно включить предложение WHERE:

```
SELECT last_name FROM employees
WHERE  employee_id NOT IN
                        (SELECT manager_id
                         FROM   employees
                         WHERE  manager_id IS NOT NULL);
```

Inline Views или подзапрос

```
SELECT salary,
(SELECT SUM(salary) FROM employees)
FROM employees;

SELECT a.*
FROM (SELECT SUM(salary) FROM employees) a;
```


Inline Views

Запросы могут присутствовать в любом месте SELECT. Они могут возвращать, поле,строку, набор строк. Если они присутствуют во фразе FROM, то называются Inline Views, так как можно было бы создать объект View и использовать его. Синонимы Inline View: subselect (не путать с squbquery) и derived table.
В БД Postgres для Inline View во фразе FROM **обязателен псевдоним**, опционально перед ним можно указывать AS.
В БД Oracle псевдоним не обязателен и **указывать AS нельзя.**
На слайде приведен синтаксис поддерживаемый обеими БД.
Более сложный пример:

```
SELECT a.last_name, a.salary, a.department_id, b.maxsal
  FROM employees a, ( SELECT department_id, max(salary) maxsal
                      FROM employees GROUP BY department_id ) b
  WHERE a.department_id = b.department_id
    AND a.salary = b.maxsal;
```

Это коррелированный подзапрос**, акие запросы будут рассмотрены в 13 главе**

Задача

Нужно создать запрос для вывода наименований отделов, в которых сумма окладов больше, чем средняя сумма окладов сотрудников по всем отделам

8 - 19 SQL

Задача

Для решения поставленной на слайде задачи требуется вычисление следующих промежуточных данных:

1. Вычисление общей суммы окладов по каждому отделу и сохранение результатов.
2. Вычисление среднего значения по общим окладам всех отделов и сохранение результата.
3. Сравнение общего оклада, полученного на первом шаге, со средним окладом, полученным на втором шаге. Если общий фонд заработной платы отдела превышает средний оклад по всем отделам, тогда выводится наименование отдела и общая сумма окладов для этого отдела, иначе строка не выводится.

Задачу можно решить так:

```
SELECT department_name, SUM(salary)
FROM employees JOIN departments USING (department_id)
GROUP BY department_name
HAVING SUM(salary) >
        (SELECT SUM(dept_total)/COUNT(department_id)
         FROM
         (SELECT department_id, SUM(salary) dept_total
          FROM employees
          GROUP BY department_id));
```

Решение с WITH
- командой выборки языка SQL

```
WITH
   dept_costs AS (
      SELECT department_name, SUM(salary) dept_total
      FROM employees JOIN departments
      USING (department_id)
      GROUP BY department_name),
   avg_cost AS (
                SELECT SUM(dept_total)/COUNT(*) avg
                FROM dept_costs)
SELECT * FROM dept_costs
   WHERE dept_total >
      (SELECT avg FROM avg_cost)
ORDER BY department_name;
```

```
DEPARTMENT_NAME DEPT_TOTAL
--------------- ----------
Sales           304500
Shipping        156400
```

8 - 20 SQL

Решение с WITH

Это вторая команда выборки данных языка SQL, кроме SELECT. Появилась в стандарте SQL:1999.

Команда выполняет:
1)Расчет совокупного оклада для каждого отдела и сохранение результата с помощью предложения WITH.
2)Расчет среднего оклада по отделам и сохранение результата с помощью предложения WITH.
3)Сравнение совокупного оклада, рассчитанного на первом шаге, со средним окладом, рассчитанным на втором шаге. Если совокупный оклад для конкретного отдела больше среднего оклада по отделам, тогда название этого отдела и совокупный оклад для этого отдела выводятся на экран.

На 1 и 2 шагах запросами создаются имена запросов DEPT_COSTS и AVG_COST, а затем они вставляются в тело главного запроса. Внутри запроса предложение WITH раскладывается как встроенное представление или рабочая временная таблица.

Второй пример с WITH

- **Для каждого сотрудника выдать число сотрудников в его отделе**

```
WITH dept_count AS (
  SELECT department_id, COUNT(*) AS dept_count
  FROM   employees
  GROUP BY department_id)
SELECT e.last_name AS employee_name, dc.dept_count
FROM   employees e, dept_count dc
WHERE  e.department_id = dc.department_id;
```

- **эквивалент с Inline View**

```
SELECT e.last_name AS employee_name, dc.dept_count
FROM   employees e,
       (SELECT department_id, COUNT(*) AS dept_count
        FROM   employees
        GROUP BY department_id) dc
WHERE  e.department_id = dc.department_id;
```


Второй пример с WITH
упрощает написание запросов и делает их более понятными.

Также БД понимает смысл задачи и, обычно, выполняет команду более эффективно (например, использует технику временных таблиц), чем с использованием множества inline views.

Полезное свойство запросов WITH заключается в том, что они выполняются только один раз при запуске родительского запроса, даже если они вызываются из родительского запроса или вложенных WITH запросов более одного раза.

Технически это достигается путем “материализации” запроса – использованием автоматически создаваемых временных таблиц в памяти (при больших объемах используется временное табличное пространство в БД Oracle и временные файлы в postgres, точно так же, как и для объектов временные таблицы).

Таким образом, затратные по ресурсам вычисления, которые необходимы во многих случаях, могут быть размещены внутри WITH запроса, чтобы избежать избыточной обработки. Другое возможное применение состоит в предотвращении нежелательного многократного выполнения функций, которое может вызвать посторонние эффекты.

Команда WITH упрощает написание запросов.

Усложним задачу. Нужно добавить к выборке имя менеджера и количество сотрудников в отделе менеджера. С использованием **inline views** эта задача решается так:

```
SELECT e.last_name AS employee_name,
       dc1.dept_count AS emp_dept_count,
       m.last_name AS manager_name,
       dc2.dept_count AS mgr_dept_count
FROM   employees e,
       (SELECT department_id, COUNT(*) AS dept_count
        FROM   employees
        GROUP BY department_id) dc1,
       employees m,
       (SELECT department_id, COUNT(*) AS dept_count
        FROM   employees
        GROUP BY department_id) dc2
WHERE  e.department_id = dc1.department_id
AND    e.manager_id = m.employee_id
AND    m.department_id = dc2.department_id;
```

С использованием WITH не нужно повторять SELECT несколько раз:

```
WITH dept_count AS (
  SELECT department_id, COUNT(*) AS dept_count
  FROM   employees
  GROUP BY department_id)
SELECT e.last_name AS employee_name,
       dc1.dept_count AS emp_dept_count,
       m.last_name AS manager_name,
       dc2.dept_count AS mgr_dept_count
FROM employees e, dept_count dc1, employees m, dept_count dc2
WHERE  e.department_id = dc1.department_id
AND    e.manager_id = m.employee_id
AND    m.department_id = dc2.department_id;
```

Команда WITH

- Делает команду более понятной
- Используя команду WITH, можно тот же самый блок запроса применять более одного раза в команде SELECT внутри сложного запроса
- С помощью WITH извлекаются результаты блока запроса и сохраняются во временной структуре БД
- Применение WITH повышает производительность

8 - 23 SQL

Команда WITH

С помощью команды WITH можно определить блок запроса и использовать его в команде запроса. WITH позволяет повторно применять тот же самый блок запроса в команде SELECT, когда этот блок появляется более одного раза в сложном запросе. Это особенно полезно, когда в запросе много ссылок на тот же самый блок запроса, а также имеются операции соединения и агрегирования. Используя WITH, можно повторно применять тот же самый запрос, когда стоимость обработки блока запроса высокая и он появляется более одного раза внутри сложного запроса. При заданном WITH БД извлекает результаты блока запроса и сохраняет их во временных структурах. В большинстве случаев может быть получена более высокая производительность выполнения больших запросов

WITH обрабатывается только один раз, даже если оно появляется несколько раз в запросе.

Замечания по применению WITH

- Псевдоним запроса виден всеми элементами блоков запросов из WITH (в том числе их блоками подзапросов), которые определены после этого псевдонима, а также самим главным запросом (том числе его блоками подзапросов).
- Если псевдоним запроса совпадает с именем таблицы, синтаксический анализатор производит поиск в промежуточном подзапросе, учитывая, что псевдоним блока запроса имеет более высокий приоритет по сравнению с именем таблицы.
- WITH может содержать более одного запроса, которые в этом случае отделяются запятой.

Практическое задание 8

- Создание подзапросов для выборки данных по неизвестным критериям.
- Использование подзапросов для выявления значений, существующих в одном наборе данных и отсутствующих в другом.

8 - 24 SQL

Практическое задание 8

Это занятие посвящено созданию сложных запросов с подзапросами SELECT.

Возможно, что для ответов на эти вопросы лучше сначала создать подзапрос. Прежде, чем кодировать главный запрос, убедитесь в том, что подзапрос выполняется и дает нужные данные.

Объединение результатов выборки

Объединение выборок

Две таблицы:

```
CREATE TABLE a (n NUMERIC(1));
INSERT INTO  a VALUES(1);
INSERT INTO  a VALUES(1);
CREATE TABLE b (n NUMERIC(1));
INSERT INTO  b VALUES(1);
INSERT INTO  b VALUES(2);
```

Задача: есть две выборки (SELECT)

```
SELECT n FROM a;
SELECT n FROM b;
```

Количество строк в обоих таблицах может быть огромным, нужно чтобы обработку данных выполнила БД одной командой SQL

Нужно: обработать два набора строк

Объединение выборок

INTERSECT

Задача: выдать строки, которые присутствуют в обоих таблицах

Решение

```
SELECT n FROM a
INTERSECT
SELECT n FROM b;
```

Результат:

```
n
--
1
```

Количество столбцов в выборках должно быть одинаково

Типы данных должны быть совместимыми

9 - 3 SQL

INTERSECT

Операторы работы с множествами (выборками) работают с результатами двух SELECT. Для простоты в примере используется выборка столбца таблицы, однако сами SELECT могут быть достаточно сложными.

Типы данных столбцов в выборках должны быть совместимыми. Если типы несовместимы, то можно использовать приведение типов, например, CAST(n AS VARCHAR(10)) или TO_CHAR(n).

Типы столбцов в выборках (столбцы, выражения, литералы, результаты функций, подзапросов), которые перечислены после слова SELECT наследуют типы данных столбцов или определяются типом возвращаемых выражениями и функциями значений.

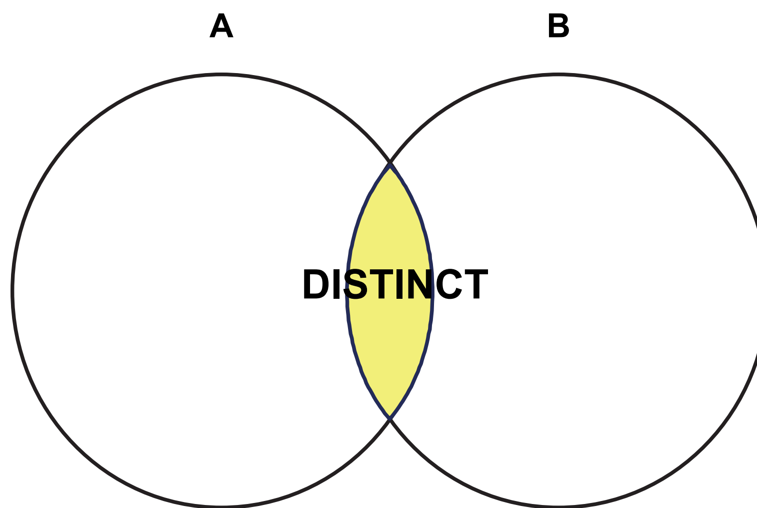
Количество столбцов в выборках должно быть одинаково.

Имена столбцов могут различаться. В результатах команды выводятся имена столбцов из первого запроса, поэтому имена столбцов во втором SELECT не важны.

Замена SELECT местами не влияет на результат.

Оператор INTERSECT исключает дубликаты ВСЕХ строк, поэтому DISTINCT ни в одном из SELECT использовать не нужно.

Оператор `INTERSECT`



Оператор `INTERSECT` возвращает строки, общие для обоих запросов

9 - 4 SQL

Оператор `INTERSECT`

MINUS/EXCEPT

Задача: выдать строки, которые отсутствуют во второй таблице

Решение:

```
SELECT n FROM a
MINUS/EXCEPT
SELECT n FROM b;
```

Результат:

```
no rows selected
```

Первый SELECT выдаст 1,1. Второй SELECT 1,2
Во втором присутствуют все строки, которые есть в первом

MINUS или EXCEPT

В БД Oracle название оператора MINUS

Используйте оператор `MINUS` для вывода строк первого запроса, которые не присутствуют во втором запросе (первая команда `SELECT MINUS` вторая команда `SELECT`).

DISTINCT использовать не нужно, дубли удаляются.

Если поменять местами SELECT результат изменится.

UNION

Задача: объединить строки обоих таблиц, исключив дублирующиеся

Решение:

```
SELECT n FROM a
UNION
SELECT n FROM b;
```

Результат:

```
n
--
1
2
```

DISTINCT использовать нет необходимости, хотя синтаксис допускает его использование

9 - 6 SQL

Использование оператора UNION

Оператор UNION исключает любые повторяющиеся записи, в том числе содержащие NULL.

Неопределенные значения (NULL), если они присутствуют в результатах SELECT также проверяются на дублирование и если в двух выборках присутствуют NULL, они считаются одинаковыми и дубль исключается.

Разные строки:

AAA NULL

NULL AAA

Одинаковые строки:

AAA NULL

AAA NULL

Одинаковые строки:

NULL NULL

NULL NULL

Поиск дублей и их исключений проводится после выполнения UNION и будут исключены все дубли из ОБЕИХ таблиц. Поэтому DISTINCT не нужно использовать ни в одном SELECT, хотя синтаксис это допускает.

UNION ALL

Задача: объединить ВСЕ строки обоих таблиц, не исключая дублирующиеся
Решение:

```
SELECT n FROM a
UNION ALL
SELECT n FROM b;
```

Результат:

```
n
--
1
1
1
2
```

повторяющиеся строки не исключаются

9 - 7 SQL

UNION ALL

Ключевое слово DISTINCT можно использовать на уровне любого SELECT, оно будет действовать на этот SELECT до выполнения объединения (UNION ALL) результатов.

В отличие от оператора `UNION` повторяющиеся строки не исключаются, в том числе содержащие NULL.

Данные не сортируются ни на одном из уровней

Объединение нескольких выборок

Можно объединять выборки, используя скобки ()

```
(SELECT n FROM a
UNION
SELECT n FROM b)
UNION ALL
(SELECT n FROM a
INTERSECT
SELECT n FROM b);
```

Результат:

```
n
--
1
2
1
```

9 - 8 SQL

Объединение нескольких выборок

Сортировка результата

Для сортировки результата используется ORDER BY

```
(SELECT n AS z FROM a
UNION
SELECT n FROM b)
UNION ALL
(SELECT n FROM a
INTERSECT
SELECT n FROM b)
ORDER BY z;
```

Который сортирует результат:

```
n
--
1
1
2
```


Сортировка результата

Для сортировки результата объединения можно использовать ORDER BY, который указывается в конце и относится ко всей выборке.

Использовать ORDER BY на уровне отдельных SELECT нельзя.

В БД Oracle выполняется неявно сортировка результатирующей выборки для всех операторов над множествами, кроме UNION ALL. Сортировка технически используется для исключения дублей строк, поэтому данные выдаются в отсортированном виде. Сортировка выполняется по первому столбцу(выражению) в выборке. Поэтому более эффективно указывать первым столбцом тот, столбец, который выдаёт большее число возможных значений, например, столбец с идентификаторами строк исходных таблиц.

Сортировка результата

ORDER BY нельзя использовать внутри, так как это не имеет смысла

```
(SELECT n AS z FROM a ORDER BY z
UNION
SELECT n FROM b)
UNION ALL
(SELECT n FROM a
INTERSECT
SELECT n FROM b);
```

```
SQL error: missing right parenthesis
```

9 - 10 SQL

Сортировка результата

Допускается использовать круглые скобки для ограничения отдельных SELECT:

```
(SELECT n FROM a)
UNION
(SELECT n FROM b);
```

Это позволяет в **БД postgres** использовать ORDER BY на уровне одного из SELECT:

```
(SELECT n FROM a ORDER BY n)
UNION
(SELECT n FROM b);
```

В **БД Oracle** ORDER BY можно использовать только на результатирующей выборке. Смысла использовать ORDER BY на уровне одного из SELECT нет.

Другие опции WHERE, GROUP BY в круглых скобках не нуждаются.

Использование операторов объединения в подзапросах

```
SELECT employee_id, department_id
 FROM   employees
 WHERE  (employee_id, department_id)
        IN (SELECT  employee_id, department_id
            FROM    employees
            UNION
            SELECT  employee_id, department_id
            FROM    job_history);
```


Использование операторов объединения в подзапросах

В **БД Oracle** также нельзя добавить ORDER BY куда-либо в подзапрос, так как это не имеет смысла. Сортироваться должна результатирующая выборка. ORDER BY может присутствовать только там, где это имеет смысл, например, в окнах, которые будут рассмотрены позже.

```
SELECT employee_id, department_id
 FROM   employees
 WHERE  (employee_id, department_id)
        IN (SELECT  employee_id, department_id
            FROM    employees
            UNION ORDER BY 1);
```

Пример согласования команд SELECT

Используя оператор UNION, для всех сотрудников выводится их номер, идентификаторы должностей и оклад.

```
SELECT employee_id, job_id,salary
FROM   employees
UNION
SELECT employee_id, job_id,0
FROM   job_history;
```

EMPLOYEE_ID	JOB_ID	SALARY
100	AD_PRES	24000
101	AC_ACCOUNT	0
101	AC_MGR	0

...

Пример согласования команд SELECT

Число столбцов в SELECT должно быть одинаковым, однако иногда в какой-то таблице отсутствует нужный столбец. Можно использовать литерал для того, чтобы добавить значение в выборку.

В примере на слайде таблицы EMPLOYEES и JOB_HISTORY имеют общие столбцы EMPLOYEE_ID, JOB_ID и DEPARTMENT_ID.

Что необходимо сделать, чтобы вывести столбцы EMPLOYEE_ID, JOB_ID и SALARY, используя оператор UNION, если SALARY отсутствует в таблице JOB_HISTORY?

Можно использовать численный литерал 0 и указать её в позиции столбца SALARY первого запроса.

Также можно использовать любые другие литералы, подходящие по смыслу. Например, NULL.

Команда WITH и рекурсивный запрос

```
WITH RECURSIVE t1(idn, parent_id) AS (
  -- нерекурсивный запрос
  SELECT idn, parent_id
  FROM   tab1
  WHERE  parent_id IS NULL
  UNION ALL
  -- рекурсивный запрос
  SELECT a.idn, a.parent_id
  FROM   tab1 a, t1
  WHERE  a.parent_id = t1.idn
)
SELECT idn, parent_id FROM t1;  -- родительский запрос
```

tab1

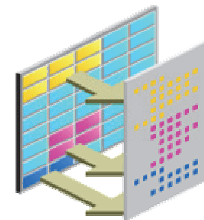
IDN	PARENT_ID
1	(null)
2	1
3	2

```
 IDN        PARENT_ID
---------- ----------
         1
         2          1
         3          2
```


Команда WITH и рекурсивный запрос
Рекурсивные запросы обычно используются при работе с иерархическими данными или данными, которые имеют древовидную структуру.
В БД postgres обязательно использование ключевого слова RECURSIVE. Синтаксис опций для иерархических запросов в Oracle и postgres различен. В postgres WITH может содержать операции DML, что является расширением стандарта SQL. В БД Oracle есть расширение к команде INSERT с аналогичным функционалом - INSERT ALL/FIRST.
Общая форма рекурсивного запроса WITH всегда нерекурсивный запрос, затем **UNION** или **UNION ALL**, затем рекурсивный запрос, где только рекурсивный запрос может содержать ссылку на свой собственный вывод результата.
Шаги выполнения рекурсивного WITH:
Выполняется нерекурсивный запрос. Для UNION (но не для UNION ALL), отбрасываются дублирующиеся строки. Включаются все оставшиеся строки из результата рекурсивного запроса и размещаются в памяти (временную " рабочую таблицу").
Пока "рабочая таблица" не окажется пустой, повторяются следующие шаги:
Выполняется рекурсивный запрос, подстановка текущего содержимого рабочей таблицы для рекурсивной ссылки на саму себя. Для UNION (но не для UNION ALL), отбрасываются дублирующиеся строки и строки, которые дублируют любые строки в предыдущих результатах. Включаются все оставшиеся строки из результата рекурсивного запроса и также размещаются во временную промежуточную таблицу.
Замещается содержимое рабочей таблицы на содержимое промежуточной таблицы, затем промежуточная таблица очищается.
При работе с рекурсивными запросами, важно убедиться, что рекурсивная часть запроса исчерпает строки, иначе запрос станет бесконечным циклом и выдастся ошибка cycle detected while executing recursive WITH query. Для тестирования запросов, когда вы не знаете точно, может ли случиться зацикливание, можно добавить опцию LIMIT к родительскому запросу.

GROUPING SETS

- **GROUPING SETS используется для объединения нескольких GROUP BY в одном запросе**
- **Преимущества:**
  - **Обрабатывает данные за один проход**
  - **Не нужно использовать UNION ALL**
  - **Выполняется эффективнее, чем UNION ALL**



GROUPING SETS - расширение функционала GROUP BY позволяющее объединить несколько группировок в одном запросе и сканировать данные за один проход. При замене на эквивалентные конструкции с UNION ALL БД будет сканировать данные несколько раз.

GROUPING SETS: пример

```
SELECT   department_id, job_id, manager_id,AVG(salary)
FROM     employees
GROUP BY GROUPING SETS
((department_id,job_id), (job_id,manager_id));
```

эквивалент

```
SELECT department_id, job_id, NULL as manager_id, AVG(salary)
FROM employees
GROUP BY department_id, job_id
UNION ALL
SELECT NULL, job_id, manager_id, AVG(salary)
FROM employees
GROUP BY job_id, manager_id;
```


GROUPING SETS: пример

В примере на слайде подсчитываются итоговые значения для двух группировок. Рассчитывается средний оклад сотрудников каждой из этих групп.

БД не увязывает отдельные блоки второй команды с UNION ALL и поэтому выстроит план выполнения, в соответствие с которым команда будет выполнять дважды сканирование базовой таблицы EMPLOYEES. Это очень неэффективно. Поэтому рекомендуется использовать команду с GROUPING SETS.

GROUPING SETS

- **Добавим третью группу**

```
SELECT   department_id, job_id, manager_id, AVG(salary)
FROM     employees
GROUP BY GROUPING SETS
   ((department_id,job_id), (job_id,manager_id), ());
```

- **К результату добавится строка со средней зарплатой по всем сотрудникам**

department_id	job_id	manager_id	avg
(null)	ST_CLERK	123	3000.0000000000000000
(null)	ST_CLERK	124	2925.0000000000000000
(null)	ST_MAN	100	7280.0000000000000000
(null)	(null)	(null)	6461.8317757009345794
10	AD_ASST	(null)	4400.0000000000000000
20	MK_MAN	(null)	13000.0000000000000000
20	MK_REP	(null)	6000.0000000000000000

- **Порядок вывода строк произвольный**
- **Для упорядочивания используется ORDER BY**

GROUPING SETS

Добавим к предыдущему примеру третью группировку: по всем строкам таблицы. Будет вычислена средняя зарплата по всем сотрудникам и результатирующая строка добавлена к выборке логикой UNION ALL.

Порядок выдачи группировок в результате может быть произвольным и не соответствовать порядку группировок в выражении GROUPING SETS. Поэтому порядок указания группировок в GROUPING SETS не важен и в примере на слайде можно переставить (department_id,job_id), (job_id,manager_id) местами.

Также по той же причине суммарную строку можно будет увидеть где угодно в результате - строка возвращается в примере на слайде в середине выборки.

Для упорядочивания результата нужно использовать ORDER BY в конце команды.

ROLLUP и CUBE

```
SELECT   department_id, job_id, SUM(salary)
FROM     employees
WHERE    department_id < 60
GROUP BY ROLLUP(department_id, job_id);
```

	DEPARTMENT_ID	JOB_ID	SUM(SALARY)
1	10	AD_ASST	4400
2	10		4400
3	20	MK_MAN	13000
4	20	MK_REP	6000
5	20		19000
6	30	PU_MAN	11000
7	30	PU_CLERK	13900
8	30		24900
9	40	HR_REP	6500
10	40		6500
11	50	ST_MAN	36400
12	50	SH_CLERK	64300
13	50	ST_CLERK	55700
14	50		156400
15			211200

1 Группировка по DEPARTMENT_ID и JOB_ID

2 GROUP BY DEPARTMENT_ID

3 По всей выборке

ROLLUP и CUBE

Если групп много, то для более компактной записи вместо GROUPNG SETS можно использовать ROLLUP и CUBE. Пример для трех столбцов:

CUBE(a, b, c) эквивалентен	GROUPING SETS ((a, b, c), (a, b), (a, c), (b, c), (a), (b), (c), ())
ROLLUP(a, b,c) эквивалентен	GROUPING SETS ((a, b, c), (a, b),(a), ())

Можно указывать что группировать:
ROLLUP(department_id,(job_id,manager_id)) – эквивалентен группировке (department_id, job_id, manager_id) (department_id) ()
Можно комбинировать операторы:
GROUP BY department_id, ROLLUP(job_id), CUBE(manager_id);
Однако эти возможности затрудняют понимание того, что получится в результате и не дают преимуществ в производительности, по сравнению с GROUPING SETS.

Использование операторов ROLLUP и CUBE вместе с GROUP BY

- **Используйте ROLLUP и CUBE вместе с GROUP BY для получения обобщенных строк перекрестного отчета.**
- **ROLLUP выдает результат группировки, содержащий обычные сгруппированные строки и промежуточные итоговые значения.**
- **CUBE выдает результат группировки , содержащий строки после применения операции ROLLUP и строки с перекрестными итоговыми данными.**

9 - 18 SQL

GROUP BY с ROLLUP и CUBE

В команде запроса операторы ROLLUP и CUBE задаются в предложении GROUP BY. Операция ROLLUP выдает результат группировки, содержащий обычные сгруппированные строки и строки промежуточных итогов. При использовании операции CUBE в предложении GROUP BY выбранные строки группируются на основе значений всех возможных комбинаций заданных выражений. В результате возвращается по одной строке итоговой информации для каждой группы.

Примечание

Для использования операторов ROLLUP и CUBE необходимо, чтобы столбцы, указанные в предложении GROUP BY, были по смыслу связаны друг с другом, чтобы получились осмысленные результаты.

ROLLUP

- ROLLUP расширяет применение предложения GROUP BY
- Используйте операцию ROLLUP для получения промежуточных итогов

```
SELECT        [столбец,] групп_функция(столбец), ...
FROM          таблица
[WHERE        условие]
[GROUP BY     [ROLLUP] выражение_группировки]
[HAVING       ограничивающее_условие]
[ORDER BY     столбец];
```


Оператор ROLLUP

Оператор ROLLUP вырабатывает строки с общими и промежуточными итогами на основе выражений, заданных в предложении GROUP BY. Оператор ROLLUP может использоваться для написания отчетов, в которых из результирующих наборов извлекаются статистические данные и итоговая информация. Полученные обобщенные данные могут быть использованы в отчетах, диаграммах и графиках.

Оператор ROLLUP создает группы, перемещаясь в одном направлении справа налево вдоль списка столбцов в предложении GROUP BY. Затем он применяет обобщенную функцию к этим сгруппированным наборам данных (groupings).

Примечание

- Чтобы получить промежуточные итоги для n размерностей (которые соответствуют n столбцам в предложении GROUP BY) без оператора ROLLUP, $n+1$ команда SELECT должна быть объединена с помощью оператора UNION ALL. Такой запрос выполняется неэффективно, так как каждая команда SELECT обращается к таблице. Оператор ROLLUP собирает результаты, обращаясь к таблице только один раз. Оператор ROLLUP полезен, если много столбцов группируется для получения промежуточных итоговых данных.
- С помощью предложения ROLLUP получаются промежуточные и обобщенные итоговые данные. Операция CUBE также позволяет получить общие итоговые данные, а также перекрестные итоги, получаемые в результате операции ROLLUP по каждому возможному направлению.

ROLLUP: пример

```
SELECT   department_id, job_id, SUM(salary)
FROM     employees
WHERE    department id < 60
GROUP BY ROLLUP(department_id, job_id);
```

DEPARTMENT_ID	JOB_ID	SUM(SALARY)
10	AD_ASST	4400
10		4400
20	MK_MAN	13000
20	MK_REP	6000
20		19000
30	PU_MAN	11000
30	PU_CLERK	13900
30		24900
40	HR_REP	6500
40		6500
50	ST_MAN	36400
50	SH_CLERK	64300
50	ST_CLERK	55700
50		156400
		211200

1
2
3

15 rows selected.

ROLLUP: пример

В примере на слайде:

- общая сумма окладов по каждой должности внутри отделов, номера которых меньше 60, выводится на основе предложения GROUP BY (метка 1);
- оператор ROLLUP выдает:
  - общую сумму окладов сотрудников отделов, номера которых меньше 60 (метка 2)
  - общую сумму окладов сотрудников всех отделов , номера которых меньше 60, независимо от должности (метка 3)

В этом примере все строки, отмеченные меткой 1, – это итоги по группе, задаваемой столбцами DEPARTMENT_ID и JOB_ID, строки, отмеченные как 2, – итоги по группе, определяемой столбцом DEPARTMENT_ID, и 3 – общий итог. Оператор ROLLUP вырабатывает промежуточные итоговые данные, которые сворачиваются (roll up) с наименьшего уровня детализации до самого высокого уровня в порядке, заданном списком группируемых столбцов в предложении GROUP BY. Во-первых, подсчитывается стандартные итоговые значения для групп, определенных в предложении GROUP BY (в примере сумма окладов по каждой должности внутри отдела). Затем в ходе перемещения справа налево по списку группируемых столбцов постепенно создаются итоги более высокого уровня. В приведенном примере подсчитывается сумма окладов по каждому отделу и затем общая сумма по всем отделам.

- В предложении GROUP BY для n выражений в операторе ROLLUP, было выполнено
  n + 1 = 2 + 1 = 3 операции группировки.
- Строки, основанные на первых n выражениях, называют строками или обычными строками, остальные строки называют строками с обобщенными данными (superaggregate rows).

CUBE

- CUBE is CUBE расширяет применение предложения GROUP BY
- Используйте операцию CUBE для получения в одной команде SELECT сводных пересекающихся данных

```
SELECT       [столбец,] групп_функция(столбец), ...
FROM         таблица
[WHERE       условие]
[GROUP BY    [CUBE] выражение_группировки]
[HAVING      ограничивающее_условие]
[ORDER BY    столбец];
```


Оператор CUBE

Оператор CUBE может быть применен ко всем агрегированным функциям, включая AVG, SUM, MAX, MIN и COUNT. CUBE создает промежуточные итоговые данные на основе всех возможных группировок столбцов и выражений из предложения GROUP BY, а также общий итог.

Оператор CUBE используется вместе с групповыми функциями для генерации дополнительных строк результирующих наборов. Столбцы из предложения GROUP BY взаимно пересекаются для выработки расширенного набора групп. Агрегированные функции, указанные в списке SELECT, применяются ко всем эти группам и вырабатывают итоговые значения для дополнительных *строк с общими итогами*. Числа столбцов, входящих в фразу GROUP BY, определяет количество дополнительных групп в результирующих наборах.

Фактически каждая возможная комбинация столбцов и выражений из предложения GROUP BY используется для выработки общих итогов. Если в предложении GROUP BY имеется n столбцов или выражений, тогда возможно 2^n комбинаций для получения общих итогов. Математически эти комбинации формируют куб с n размерностями. Поэтому эта операция называется CUBE.

Полученные в приложениях и с помощью средств программирования такие общие значения могут быть затем использованы для построения диаграмм и графиков, которые отражают полученные результаты и взаимосвязи.

CUBE: пример

```
SELECT    department_id, job_id, SUM(salary)
FROM      employees
WHERE     department_id < 60
GROUP BY CUBE (department_id, job_id) ;
```

DEPARTMENT_ID	JOB_ID	SUM(SALARY)
		211200
	HR_REP	6500
	MK_MAN	13000
	MK_REP	6000
	PU_MAN	11000
	ST_MAN	36400
	AD_ASST	4400
	PU_CLERK	13900
	SH_CLERK	64300
	ST_CLERK	55700
10		4400
10	AD_ASST	4400
20		19000
20	MK_MAN	13000
20	MK_REP	6000
30		24900
30	PU_MAN	11000

1 2 3 4

CUBE: пример

Результат выполнения команды SELECT, приведенной в примере можно проинтерпретировать следующим образом.

- Общие суммы окладов по каждой должности внутри отделов, номера которых меньше 60, выводятся на основе предложения GROUP BY.
- Общие суммы окладов сотрудников по каждому отделу, номер которого меньше 60.
- Общие суммы окладов сотрудников по всем должностям независимо от отдела, полученные на основе данных, в которых номер отдела меньше 60.
- Общие суммы окладов сотрудников всех отделов, номера которых меньше 60, независимо от должности.

В предыдущем примере строка, отмеченная меткой 1, – общий итогами, строки, отмеченные как 2, – итоги по группе, определяемой столбцом JOB_ID, 3 – итоги по группе DEPARTMENT_ID и JOB_ID и 4 – итоги по группе DEPARTMENT_ID .

Оператор CUBE также выполняет операцию ROLLUP для выработки промежуточных итогов по отделам, номера которых меньше 60, независимо от наименования должности. Дополнительно оператор CUBE выводит общий оклад по каждой должности независимо от отделов.

Примечание

Аналогично оператору ROLLUP, чтобы получить промежуточные итоги для n размерностей (которые соответствуют n столбцам в предложении GROUP BY) без оператора CUBE, 2^n команд SELECT должны быть объединены с помощью оператора UNION ALL. Так для отчета с тремя размерностями требуется $2^3 = 8$ команд SELECT, объединенных с помощью оператора UNION ALL.

Функция GROUPING

- **Используется чтобы отличить NULL в результатах от NULL создаваемых группировками**
- **возвращает 0 or 1**

```
SELECT    department_id DEPTID, job_id JOB, SUM(salary),
          GROUPING(department_id) GRP_DEPT,
          GROUPING(job_id) GRP_JOB
FROM      employees WHERE department_id < 50
GROUP BY ROLLUP(department_id, job_id);
```

	DEPTID	JOB	SUM(SALARY)	GRP_DEPT	GRP_JOB
1	10	AD_ASST	4400	0	0
2	10		4400	0	1
3	20	MK_MAN	13000	0	0
4	20	MK_REP	6000	0	0
5	20		19000	0	1
6	30	PU_MAN	11000	0	0
7	30	PU_CLERK	13900	0	0
8	30		24900	0	1
9	40	HR_REP	6500	0	0
10	40		6500	0	1
11			54800	1	1

1 2 3

Ссылки на группирующие столбцы или выражения заменяются в результирующих строках значениями NULL для тех группирующих наборов, в которых эти столбцы отсутствуют. Чтобы можно было понять, результатом какого группирования стала конкретная выходная строка можно использовать функцию GROUPING. Позволяет отличить NULL в данных от пустых значений которые выдаются при группировании.

1. В примере на слайде значение 4400 в первой строке получается группировкой (DEPARTMENT_ID, JOB_ID). Функции GROUPING(department_id)и GROUPING(job_id) возвращают 0.
2. Во второй строке значение 4400 получается группировкой по (DEPARTMENT_ID). Функция GROUPING(department_id) возвращает 0.
3. В последней строке выдаётся 54800. Это сумма зарплат всех сотрудников WHERE department_id < 50 и GROUPING(department_id) GROUPING(job_id)выдают 1.

Функция GROUPING

- Функция GROUPING может быть использована с операторами CUBE или ROLLUP.
- Используя функцию GROUPING, можно найти группы, формирующие промежуточные итоги для строки.
- Использование функции GROUPING позволяет отличить хранимые значения NULL от созданных операциями ROLLUP или CUBE.
- Функция GROUPING возвращает 0 или 1.

```
SELECT      столбец,] групп_функция(столбец), ...,
            GROUPING(выражение)
FROM        таблица
[WHERE      условие]
[GROUP BY [ROLLUP][CUBE] выражение_группировки]
[HAVING     ограничивающее_условие]
[ORDER BY столбец];
```

9 - 24 SQL

Функция GROUPING

Функция GROUPING может быть использована вместе с оператором CUBE или оператором ROLLUP. Она помогает понять, как были получены итоговые значения. Функция GROUPING использует единственный столбец в качестве аргумента. *Выражение* в функции GROUPING должно соответствовать одному из *выражений_группировки* предложения GROUP BY clause. Функция GROUPING возвращает 0 или 1.

Значения, возвращаемые функцией GROUPING, полезны для:

- определения уровня агрегирования данного промежуточного итога, т.е. для нахождения группы или групп, на которых основывается промежуточная итоговая сумма;
- определения того, неопределенное значение (NULL) в выражении столбца – это результат, который отражает факт, что:
  - значение NULL получено из базовой таблицы (хранящей значение NULL);
  - значение NULL создано ROLLUP/CUBE (как результат групповой функции, примененной к этому выражению).

Значение 0, возвращенное функцией GROUPING, основанной на *выражении*, отражает одну из следующих ситуаций:

- выражение использовалось для подсчета агрегированного значения;
- значение NULL в выражении – это хранимое значение NULL.

Значение 1, возвращенное функцией GROUPING, основанной на *выражении*, отражает одну из следующих ситуаций:

- выражение не использовалось для подсчета агрегированного значения.
- значение NULL в выражении столбца создано функциями ROLLUP или CUBE как результат группирования.

Функция GROUPING: пример

```
SELECT   department_id DEPTID, job_id JOB,
         SUM(salary),
         GROUPING(department_id) GRP_DEPT,
         GROUPING(job_id) GRP_JOB
FROM     employees
WHERE    department_id < 50
GROUP BY ROLLUP(department_id, job_id);
```

DEPTID	JOB	SUM(SALARY)	GRP_DEPT	GRP_JOB
10	AD_ASST	4400	0	0
10		4400	0	1
20	MK_MAN	13000	0	0
20	MK_REP	6000	0	0
20		19000	0	1
30	PU_MAN	11000	0	0
30	PU_CLERK	13900	0	0
30		24900	0	1
40	HR_REP	6500	0	0
40		6500	0	1
		54800	1	1

1 2 3

11 rows selected.

Пример использования функции GROUPING

Рассмотрим в примере на слайде суммарное значение 4400 из первой строки (метка 1). Это итоговое значение представляет собой общий оклад сотрудников, занимающих должность AD_ASST в отделе 10. Для подсчета этого значение принимаются во внимание оба столбца DEPARTMENT_ID и JOB_ID. Поэтому значение 0 возвращается для обоих выражений GROUPING(department_id) и GROUPING(job_id).

Рассмотрим суммарное значение 4400 во второй строке (метка 2). Это итоговое значение представляет собой общий оклад сотрудников отдела 10. При его подсчете принимался во внимание столбец DEPARTMENT_ID; поэтому значение 0 возвращается для выражения GROUPING(department_id). Так как столбец JOB_ID не учитывается при подсчете этого значения, функция GROUPING(job_id) возвращает значение 1. Заметим, что такой же результат получен в пятой строке.

Рассмотрим в последней строке суммарное значение 54800 (метка 3). Это итоговое значение представляет собой общий оклад сотрудников с любой должностью из отделов, номер которых меньше 50. При подсчете итогового значения ни столбец DEPARTMENT_ID, ни столбец JOB_ID не принимаются во внимание. Поэтому значение 1 возвращается для обоих выражений GROUPING(department_id) и GROUPING(job_id).

Составные столбцы

- **Составной столбец –это совокупность столбцов, которые обрабатываются как единое целое.**
  `ROLLUP (a, (b,c) ,d)`
- **Составные столбцы определяются в предложении `GROUP BY`. Требуемые столбцы заключаются в круглые скобки и БД обрабатывает их как единое целое при выполнении операторов `ROLLUP` и `CUBE`.**
- **Использование составных столбцов вместе с операторами `ROLLUP` и `CUBE` приводит к пропуску определенных уровней агрегирования при подсчете итоговых значений.**

9 - 26 SQL

Составные столбцы – это совокупность столбцов, которые обрабатываются как единое целое при выполнении вычислений над группировками. В следующем операторе задаются столбцы в круглых скобках:

```
ROLLUP (a, (b, c), d)
```

В этом операторе `(b,c)` образуют составной столбец, который обрабатывается как отдельная компонента. Составные столбцы обычно полезно применять в `ROLLUP`, `CUBE` и `GROUPING SETS`. Например, составные столбцы в `CUBE` или `ROLLUP` означают пропуск определенных уровней агрегирования при подсчете итоговых значений.

Таким образом предложение `GROUP BY ROLLUP(a, (b, c))` эквивалентно:

```
GROUP BY a, b, c UNION ALL
GROUP BY a UNION ALL
GROUP BY ()
```

В этом примере `(b, c)` обрабатывается как отдельная компонента и подсчет итогов (rollup) не выполняется для пересечений с отдельными столбцами из `(b, c)`. Это то же самое, что присвоить псевдоним , например `z` для `(b, c)` и сократить выражение в `GROUP BY` до `GROUP BY ROLLUP(a, z)`.

Примечание. `GROUP BY( )` – это предложение в команде `SELECT`, в которой значения для столбцов `a` и `b` неопределенны `(NULL)` и указана только агрегированная функция. Она обычно используется для генерации обобщенных итогов.

```
SELECT   NULL, NULL, агрегируемый_столбец
FROM     <имя_таблицы>
GROUP BY ( );
```

Составные столбцы (продолжение)

Сравните предыдущий пример с обычным использованием ROLLUP:

```
GROUP BY ROLLUP(a, b, c)
```

Это означает выполнение:

```
GROUP BY a, b, c UNION ALL
GROUP BY a, b UNION ALL
GROUP BY a UNION ALL
GROUP BY ()
```

Подобным образом

```
GROUP BY CUBE((a, b), c)
```

будет эквивалентно:

```
GROUP BY a, b, c UNION ALL
GROUP BY a, b UNION ALL
GROUP BY c UNION ALL
GROUP By ()
```

В следующей таблице приведены описания группируемых наборов и эквивалентные описания предложений GROUP BY.

Предложения GROUPING SETS	**Эквивалентные предложения GROUP BY**
GROUP BY GROUPING SETS(a, b, c)	GROUP BY a UNION ALL GROUP BY b UNION ALL GROUP BY c
GROUP BY GROUPING SETS(a, b,(b, c)) (The GROUPING SETS expression has a composite column.)	GROUP BY a UNION ALL GROUP BY b UNION ALL GROUP BY b, c
GROUP BY GROUPING SETS((a, b, c))	GROUP BY a, b, c
GROUP BY GROUPING SETS(a, (b), ())	GROUP BY a UNION ALL GROUP BY b UNION ALL GROUP BY ()
GROUP BY GROUPING SETS (a,ROLLUP(b, c)) (The GROUPING SETS expression has a composite column.)	GROUP BY a UNION ALL GROUP BY ROLLUP(b, c)

Составные столбцы: пример

```
SELECT   department_id, job_id, manager_id,
         SUM(salary)
FROM     employees
GROUP BY ROLLUP( department_id,(job_id, manager_id));
```

DEPARTMENT_ID	JOB_ID	MANAGER_ID	SUM(SALARY)
	SA_REP	149	7000
			7000
10	AD_ASST	101	4400
10			4400
20	MK_MAN	100	13000
20	MK_REP	201	6000
20			19000

...

DEPARTMENT_ID	JOB_ID	MANAGER_ID	SUM(SALARY)
100	FI_MGR	101	12000
100	FI_ACCOUNT	108	39600
100			51600
110	AC_MGR	101	12000
110	AC_ACCOUNT	205	8300
110			20300
			691400

46 rows selected.

Составные столбцы: пример

Рассмотрим пример:

```
SELECT department_id, job_id,manager_id, SUM(salary)
 FROM   employees
 GROUP BY ROLLUP( department_id,job_id, manager_id);
```

В ходе обработки предыдущего примера БД выполняет вычисления над следующими группировками:

1. (department_id, job_id, manager_id)
2. (department_id, job_id)
3. (department_id)
4. ( )

Если необходимы только группировки в строках 1, 3 и 4, вы не сможете ограничить вычисления только этими группировками без использования составных столбцов. Применение составного столбца позволяет обрабатывать столбцы JOB_ID и MANAGER_ID как единую компоненту при свертывании. Столбцы, заключенные в круглые скобки, обрабатываются как отдельный параметр в вычислениях по операторам ROLLUP и CUBE. Это показано в примере на слайде. Заключая столбцы JOB_ID и MANAGER_ID в круглые скобки, вы предписываете БД обрабатывать JOB_ID и MANAGER_ID как единую компоненту, как составной столбец.

Составные столбцы: пример (продолжение)

В примере на слайде выполняются вычисления над следующими группировками:

- (department_id, job_id, manager_id)
- (department_id)
- ( )

В примере на слайде выводится следующее:

- общий оклад сотрудников для каждого отдела, должности и менеджера (метки 1);
- общий оклад сотрудников каждого отдела (метка 2);
- общий итог для всех сотрудников компании(метка 3).

Пример на слайде может быть также записан в виде:

```
SELECT   department_id, job_id, manager_id, SUM(salary)
FROM     employees
GROUP    BY department_id,job_id, manager_id
UNION    ALL
SELECT   department_id, TO_CHAR(NULL),TO_NUMBER(NULL), SUM(salary)
FROM     employees
GROUP BY department_id
UNION ALL
SELECT TO_NUMBER(NULL),TO_CHAR(NULL),TO_NUMBER(NULL), SUM(salary)
FROM     employees
GROUP BY ();
```

БД не увязывает отдельные блоки приведенной команды и поэтому выстроит план выполнения, в соответствие с которым представленный запрос будет выполнять трижды сканирование базовой таблицы EMPLOYEES. Это очень неэффективно. Поэтому рекомендуется использовать составные столбцы.

Сцепленные группировки

- **Сцепленные (комбинированные) группировки позволяют простым способом генерировать полезные комбинации группировок.**
- **Чтобы определить наборы сцепленных группировок, необходимо указать через запятую несколько группируемых наборов, операции `ROLLUP` и `CUBE` так, чтобы БД скомбинировала их в одно предложение `GROUP BY`.**
- **Результатом будет векторное произведение группировок на основе группируемых наборов.**

```
GROUP BY GROUPING SETS(a, b), GROUPING SETS(c, d)
```

9 - 30 SQL

Сцепленные группировки

Сцепленные группировки (*concatenated groupings*) позволяют простым способом генерировать полезные комбинации группировок. Чтобы определить наборы сцепленных группировок, необходимо указать через запятую несколько группируемых наборов, операции `ROLLUP` и `CUBE`. Пример сцепленных группировок:

```
GROUP BY GROUPING SETS(a, b), GROUPING SETS(c, d)
```

В предыдущей записи на SQL определяются следующие группировки:

```
(a, c), (a, d), (b, c), (b, d)
```

Наборы сцепленных группировок очень полезны, так как:

- просто создавать запрос (не надо вручную перечислять все группировки);
- они используются в приложениях оперативной аналитической обработки (OLAP) для генерации команд на SQL и определяют в них применяемые размерности.

Векторные произведения: примеры

- Примеры векторных произведений со сцепленными группировками

```
SELECT  department_id, AVG(salary) FROM employees
```

```
BY GROUPING SETS(department_id) = BY department_id;
12 строк
BY GROUPING SETS(department_id),GROUPING SETS(department_id);
тоже 12 строк
GROUPING SETS((department_id), ());
13 строк
BY GROUPING SETS(department_id,()), GROUPING SETS(());
тоже 13 строк
BY GROUPING SETS(department_id,()),
   GROUPING SETS(department_id,());
37 строк (12+12+12+1)
```

9 - 31 SQL

Векторные произведения: примеры

Векторное произведение это не UNION ALL.

Сцепленные группировки: пример

```
SELECT    department_id, job_id, manager_id,
          SUM(salary)
FROM      employees
GROUP BY department_id,
          ROLLUP(job_id),
          CUBE(manager_id);
```

DEPARTMENT_ID	JOB_ID	MANAGER_ID	SUM(SALARY)
	SA_REP	149	7000
10	AD_ASST	101	4400
20	MK_MAN	100	13000
20	MK_REP	201	6000
...			
90	AD_VP	100	34000
90	AD_PRES		24000
		149	7000
			7000
...			
	SA_REP		7000
10	AD_ASST		4400
...			
110		101	12000
110		205	8300
110			20300

93 rows selected.

Сцепленные группировки: пример

В сцепленных группировках можно применять ROLLUP и CUBE.

В примере на слайде получаются результаты для следующих группировок:

- `(department_id, job_id, manager_id)` (1)
- `(department_id, job_id)` (2)
- `(department_id, manager_id)` (3)
- `(department_id)` (4)

Подсчитывается общая сумма окладов для каждой из этих групп.

Практическое задание 9

- Использование оператора UNION
- Использование оператора INTERSECTION
- Использование оператора MINUS/EXCEPT
- Использование оператора ROLLUP
- Использование функции GROUPING
- Использование GROUPING SETS

Практическое задание 9

В этом практическом занятии в запросах используются операторы над множествами, операторы ROLLUP и CUBE , которые расширяют возможности предложения GROUP BY. Вы также будете использовать выражение GROUPING SETS.

10

Аналитические функции

Окна и разделы

Оконная функция выполняет вычисления по группе строк, связанным с текущей строкой

Оконная функция вычисляется по строкам, попадающим в один раздел с текущей строкой

Зарплата каждого сотрудника и средняя зарплата его отдела:

```
SELECT department_id, last_name, salary,
trunc(avg(salary) OVER (PARTITION BY department_id)) AS
   average
FROM employees WHERE department_id=90;
```

```
DEPARTMENT_ID LAST_NAME                    SALARY    AVERAGE
------------- ------------------------- ---------- ----------
           90 King                           24000      19333
           90 Kochhar                        17000      19333
           90 De Haan                        17000      19333
```


Окна и разделы

Набор строк делится на **разделы** фразой PARTITION BY. Разбиение аналогично разбиению на группы фразой GROUP BY.

В разделе существует окно. По умолчанию окно равно разделу. Однако, оно может быть меньше раздела и тогда вводится понятие **рамки окна**.

С окнами работают оконные функции.

Если предложение PARTITION BY отсутствует, то групповые функции применяются ко всему результирующему набору строк запроса. В отличие от группировки, где мы получаем на каждую группу только одну строку, которая может содержать результат групповых функций, подсчитанные для каждой такой группы, здесь мы можем добавить агрегированный результат к несгруппированным строкам.

Оконная функция выполняет вычисления по группе строк, связанных с **текущей строкой**. Можно сравнить её с групповой функцией, но, в отличие от обычной групповой функции, при использовании оконной функции несколько строк не группируются в одну, а продолжают существовать отдельно. Внутри же, оконная функция, как и групповая, может обращаться не только к текущей строке в рамках окна.

Фактически среднее вычисляет групповая функция AVG, но предложение OVER превращает её в оконную, так что она обрабатывает лишь заданный окном набор строк.

Вызов оконной функции всегда содержит предложение OVER, следующее за названием и аргументами оконной функции. Это синтаксически отличает её от обычной или групповой функции. Предложение OVER определяет, как именно нужно разделить строки запроса для обработки оконной функцией. Предложение PARTITION BY, дополняющее OVER, указывает, что строки нужно разделить по группам или разделам, объединяя одинаковые значения выражений PARTITION BY. Оконная функция вычисляется по строкам, попадающим в один раздел с текущей строкой.

Оконные функции разрешается использовать в запросе только в списке SELECT и предложении ORDER BY. Во всех остальных предложениях, включая GROUP BY, HAVING и WHERE, они запрещены. Это объясняется тем, что логически они выполняются после обычных групповых функций, а значит групповую функцию можно вызвать из оконной, но не наоборот. Если вам нужно отфильтровать или сгруппировать строки после вычисления оконных функций, вы можете использовать запрос как вложенный в другой SELECT.

Следующие запросы эквивалентны:

```
SELECT DISTINCT department_id,
  avg(salary) OVER (PARTITION BY department_id)
FROM employees;

SELECT department_id, avg(salary)
FROM employees GROUP BY department_id;
```

ORDER BY в окне

- **Для каждой строки существует набор строк в её разделе, называемый рамкой окна**
- **без ORDER BY рамка состоит из всех строк раздела**
- **с ORDER BY рамка по умолчанию состоит из всех строк от начала раздела до текущей строки и строк, равных текущей по значению выражения в ORDER BY**

```
SELECT department_id, last_name, salary,
COUNT(salary) OVER (PARTITION BY department_id ORDER BY
    salary DESC) AS co
FROM employees WHERE department_id=60 ORDER BY salary;
```

```
DEPARTMENT_ID LAST_NAME                      SALARY         CO
------------- ------------------------- ---------- ----------
           60 Lorentz                         4200          5
           60 Austin                          4800          4
           60 Pataballa                       4800          4
           60 Ernst                           6000          2
           60 Hunold                          9000          1
```


Для каждой строки существует набор строк в её разделе, называемый ***рамкой окна***. По умолчанию, с указанием ORDER BY рамка состоит из всех строк от начала раздела до текущей строки и строк, равных текущей по значению выражения ORDER BY. Без ORDER BY рамка по умолчанию состоит из всех строк раздела. Также можно задавать порядок (ASC/DESC) в ORDER BY для окна.

Список сотрудников 60 отдела

```
SELECT department_id, last_name, salary FROM employees WHERE
department_id=60 ORDER BY salary;
DEPARTMENT_ID LAST_NAME                     SALARY
------------- ------------------------- ----------
           60 Lorentz                         4200
           60 Austin                          4800
           60 Pataballa                       4800
           60 Ernst                           6000
           60 Hunold                          9000
```

Без ORDER BY в рамку включаются все строки раздела, так как все они считаются родственными текущей.
Число строк в окнах, рамка окна – все строки раздела:

```
SELECT department_id, last_name, salary,
COUNT(salary) OVER (PARTITION BY department_id) AS co
FROM employees WHERE department_id=60 ORDER BY salary;
DEPARTMENT_ID LAST_NAME                     SALARY         CO
------------- ------------------------- ---------- ----------
           60 Lorentz                         4200          5
           60 Austin                          4800          5
           60 Pataballa                       4800          5
           60 Ernst                           6000          5
           60 Hunold                          9000          5
```

С указанием ORDER BY это означает, что рамка будет включать все строки от начала раздела до последней строки, родственной текущей.
Добавляем сортировку. Здесь в сумме вычитаются (DESC) зарплаты от первой (самой высокой) до текущей, включая повторяющиеся текущие значения. Обратите внимание на результат 4 в строках с одинаковой зарплатой.

```
SELECT department_id, last_name, salary,
COUNT(salary) OVER (PARTITION BY department_id ORDER BY salary
DESC) AS co
FROM employees WHERE department_id=60 ORDER BY salary;
DEPARTMENT_ID LAST_NAME                     SALARY         CO
------------- ------------------------- ---------- ----------
           60 Lorentz                         4200          5
           60 Austin                          4800          4
           60 Pataballa                       4800          4
           60 Ernst                           6000          2
           60 Hunold                          9000          1
```

Рамка окна

- подмножество строк текущего окна
- используется для оконных функций, которые могут работать с рамкой окна, а не со всем разделом
- значение по умолчанию: RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
- Пример: рамка начинается с первой строки раздела

```
SELECT department_id, last_name, salary,
COUNT(salary) OVER (PARTITION BY department_id ORDER BY
   salary DESC ROWS UNBOUNDED PRECEDING) AS co
FROM employees WHERE department_id=60 ORDER BY salary;
```

```
DEPARTMENT_ID LAST_NAME                    SALARY         CO
------------- ------------------------- ---------- ----------
           60 Lorentz                        4200          5
           60 Austin                         4800          3
           60 Pataballa                      4800          4
           60 Ernst                          6000          2
           60 Hunold                         9000          1
```


Рамка окна

{ RANGE | ROWS } *начало_рамки*
{ RANGE | ROWS } BETWEEN *начало_рамки* AND *конец_рамки*
начало_рамки и *конец_рамки* задаются одним из выражений:
UNBOUNDED PRECEDING
значение PRECEDING
CURRENT ROW
значение FOLLOWING
UNBOUNDED FOLLOWING
начинается с положения начало_рамки и заканчивается положением конец_рамки. Если конец_рамки опущен, подразумевается CURRENT ROW (текущая строка). Если начало_рамки задано как UNBOUNDED PRECEDING, рамка начинается с первой строки раздела, а если конец_рамки определён как UNBOUNDED FOLLOWING, рамка заканчивается последней строкой раздела.
В режиме RANGE начало_рамки, заданное как CURRENT ROW, определяет в качестве начала первую родственную строку (строку, которую ORDER BY считает равной текущей), тогда как конец_рамки, заданный как CURRENT ROW, определяет концом рамки последнюю родственную (для ORDER BY) строку. В режиме ROWS вариант CURRENT ROW просто обозначает текущую строку.

По умолчанию рамка определяется как RANGE UNBOUNDED PRECEDING, что равносильно расширенному определению RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW. С указанием ORDER BY это означает, что рамка будет включать все строки от начала раздела до последней строки, родственной текущей (для ORDER BY). Без ORDER BY в рамку включаются все строки раздела, так как все они считаются родственными текущей.

Варианты значение PRECEDING и значение FOLLOWING допускаются только в режиме ROWS. Они указывают, что рамка начинается или заканчивается со сдвигом на заданное число строк перед или после заданной строки. Здесь значение должно быть целочисленным выражением, не содержащим переменные, агрегатные или оконные функции, и может быть нулевым, что будет означать выбор текущей строки.

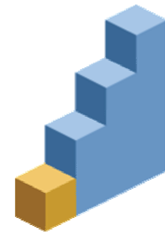
Действуют также ограничения: начало_рамки не может определяться как UNBOUNDED FOLLOWING, а конец_рамки — UNBOUNDED PRECEDING, и конец_рамки не может определяться раньше, чем начало_рамки — например, запись RANGE BETWEEN CURRENT ROW AND значение PRECEDING недопустима.

Когда в качестве оконной функции используется агрегатная, она обрабатывает строки в рамке текущей строки. Агрегатная функция с ORDER BY и определением рамки окна по умолчанию будет вычисляться как «бегущая сумма», что может не соответствовать желаемому результату. Чтобы агрегатная функция работала со всем разделом, следует опустить ORDER BY или использовать ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING.

Функции ранжировки

Вычисляют место (rank) строки по отношению к другим строкам.

- RANK и DENSE_RANK
- CUME_DIST
- PERCENT_RANK
- NTILE
- ROW_NUMBER



ROW_NUMBER()

- Выдаёт номер текущей строки в её разделе, начиная с 1

```
SELECT job_id, last_name, salary, ROW_NUMBER()
 OVER (PARTITION BY job_id ORDER BY salary) AS CD
FROM employees WHERE job_id LIKE 'PU%';
```

- Полезна, чтобы понимать как распределены (ранжированы) строки в разделах

```
PU_CLERK   Colmenares    2500.00   1
PU_CLERK   Himuro        2600.00   2
PU_CLERK   Tobias        2800.00   3
PU_CLERK   Baida         2900.00   4
PU_CLERK   Khoo          3100.00   5
PU_MAN     Raphaely     11000.00   1
```


Можно использовать эту функцию для того, чтобы лучше понять, как работают аналитические функции, изучаемые дальше, так как она даёт представление о месте строк в разделах.

Также функция может использоваться для быстрого удаления строк с дублирующимися значениями в столбцах.

```
DELETE FROM таблица
 WHERE ключевой_столбец IN (SELECT ключевой_столбец
 FROM (SELECT row_number() OVER (PARTITION BY столбец_с_дубликатами),
  ключевой_столбец FROM таблица) x
  WHERE x.row_number > 1);
```

Функция RANK()

```
SELECT department_id, last_name, salary,
   RANK() OVER (PARTITION BY department_id
   ORDER BY salary DESC) "Rank"
FROM employees
WHERE department_id = 60
ORDER BY department_id, "Rank", salary;
```

DEPARTMENT_ID	LAST_NAME	SALARY	Rank
60	Hunold	9000	1
60	Ernst	6000	2
60	Austin	4800	3
60	Pataballa	4800	3
60	Lorentz	4200	5

10 - 10 SQL

Функция RANK()

- выдаёт порядковый номер в разделе текущей строки для каждого уникального значения, по которому выполняет сортировку предложение ORDER BY. У функции RANK()

нет параметров, так как её поведение полностью определяется предложением OVER.

Строки, обрабатываемые оконной функцией, представляют собой «виртуальные таблицы», созданные из предложения FROM и затем прошедшие через фильтрацию и группировку WHERE и GROUP BY и, возможно, условие HAVING. Например, строка, отфильтрованная из-за нарушения условия WHERE, не будет видна для оконных функций. Запрос может содержать несколько оконных функций, разделяющих данные по-разному с помощью разных предложений OVER, но все они будут обрабатывать один и тот же набор строк этой виртуальной таблицы.

DENSE_RANK()

```
SELECT last_name, salary,
    RANK() OVER (PARTITION BY
    department_id
    ORDER BY salary DESC) "Rank"
FROM employees
WHERE department_id = 60
ORDER BY salary DESC, "Rank"
    DESC;
```

	LAST_NAME	SALARY	Rank
1	Hunold	9000	1
2	Ernst	6000	2
3	Austin	4800	3
4	Pataballa	4800	3
5	Lorentz	4200	5

DENSE_RANK выдает ранжировку без пропусков

RANK оставляет пропуски в последовательности

	LAST_NAME	SALARY	Drank
1	Hunold	9000	1
2	Ernst	6000	2
3	Austin	4800	3
4	Pataballa	4800	3
5	Lorentz	4200	4

```
SELECT last_name, salary,
    DENSE_RANK() over (PARTITION BY
    department_id
    ORDER BY salary DESC) "Drank"
FROM employees
WHERE department_id = 60
ORDER BY salary DESC, "Drank" DESC;
```


DENSE_RANK()

- выдаёт ранг текущей строки без пропусков; эта функция считает группы родственных строк

CUME_DIST() и PERCENT_RANK()

- CUME_DIST относительный ранг текущей строки: число строк, предшествующих или родственных текущей деленное на общее число строк

```
CUME_DIST() OVER (PARTITION BY .. ORDER BY ..)
```

- PERCENT_RANK относительный ранг текущей строки: (ранг текущей строки - 1) / (общее число строк - 1)

```
PERCENT_RANK() OVER (PARTITION BY .. ORDER BY ..)
```

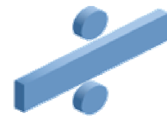

PERCENT_RANK()

- схожа с функцией кумулятивного распределения CUME_DIST. Диапазон выдаваемых значений от 0 до 1 включительно.

CUME_DIST()

```
SELECT job_id, last_name, salary,
   CUME_DIST()
   OVER (PARTITION BY job_id ORDER BY salary) AS CD
FROM employees
WHERE job_id LIKE 'PU%';
```

	JOB_ID	LAST_NAME	SALARY	CD
1	PU_CLERK	Colmenares	2500	0.2
2	PU_CLERK	Himuro	2600	0.4
3	PU_CLERK	Tobias	2800	0.6
4	PU_CLERK	Baida	2900	0.8
5	PU_CLERK	Khoo	3100	1
6	PU_MAN	Raphaely	11000	1



CUME_DIST()

- выдаёт относительное место строки в окне. В примере на слайде вычисляется процентиль для каждого сотрудника отделов PU%.

Процентиль - мера, в которой процентное значение общих значений равно этой мере или меньше ее. Например, 90 % значений данных находятся ниже 90-го процентиля, а 10 % значений данных находятся ниже 10-го процентиля.

- Для первой строки в выборке на слайде есть только одна строка с зарплатой <=2500 (так как ORDER_BY salary) среди других пяти PU_CLERK результат `CUME_DIST` = 1/5 или 0.2
- Для второй строки в выборке есть уже две строки с зарплатой <=2500 (так как ORDER_BY salary) среди других пяти PU_CLERK. `CUME_DIST` = 2/5 или 0.4.

...

- Для 6 строки есть только одна строка где `job_id=PU_MAN` и зарплата<=11000. `CUME_DIST` = 1/1 или 1

PERCENT_RANK()

```
SELECT job_id, last_name, salary,
  RANK()
    OVER (PARTITION BY job_id ORDER BY salary ) AS rank,
  PERCENT_RANK()
    OVER (PARTITION BY job_id ORDER BY salary ) AS pr
FROM employees
WHERE job_id LIKE 'PU%';
```

	JOB_ID	LAST_NAME	SALARY	RANK	PR
1	PU_CLERK	Colmenares	2500	1	0
2	PU_CLERK	Himuro	2600	2	0.25
3	PU_CLERK	Tobias	2800	3	0.5
4	PU_CLERK	Baida	2900	4	0.75
5	PU_CLERK	Khoo	3100	5	1
6	PU_MAN	Raphaely	11000	1	0

В примере на слайде вычисляется относительный ранг каждого сотрудника в PU% с окнами по значению столбца `job_id`.

Для первой строки `PERCENT_RANK` = (1-1)/(5-1) = 0/4 = 0

Для второй строки `PERCENT_RANK` = (2-1)/(5-1) = 1/4 = 0.25

Для шестой строки окно содержит единственную строку, у которой `job_id=PU_MAN` и `PERCENT_RANK` = (1-1)/(1-1) = 0.

Когда в запросе вычисляются несколько оконных функций для одинаково определённых окон, нужно писать для каждой из них отдельное предложение OVER, но при этом оно будет дублироваться.

В БД Postgres есть синтаксис, позволяющий избежать дублирования:

```
SELECT job_id, last_name, salary,
  rank() OVER w, percent_rank() OVER w
  FROM employees
  WHERE job_id LIKE 'PU%'
  WINDOW w AS (PARTITION BY job_id ORDER BY salary);
```

NTILE

```
NTILE (выражение) OVER (PARTITION BY .. ORDER BY ..)
```

```
SELECT last_name, salary, NTILE(4)
   OVER (ORDER BY salary DESC) AS quartile
FROM employees
WHERE department_id = 100
ORDER BY last_name, salary, quartile;
```

```
LAST_NAME                     SALARY   QUARTILE
------------------------- ---------- ----------
Chen                            8200          2
Faviet                          9000          1
Greenberg                      12008          1
Popp                            6900          4
Sciarra                         7700          3
Urman                           7800          2

 6 rows selected
```

1

2

NTILE

- ранжирует по целым числам от 1 до значения аргумента так, чтобы размеры групп были максимально близки.

Функция вычисляет квартили, децили и другие статистические показатели. Функция делит окна на набор групп и назначает номер группы каждой строке в окне. Например, для 62 строк в окне `NTILE(3)` включит в три группы 21, 21, 20 строк.

В примере на слайде значения столбца `salary` сотрудников 100 отдела попадают в 4 группы. В столбце `salary` имеется 6 значений. По одному значению попадает в каждую группу, а оставшиеся 2 значения попадут в группы номер один и два. 1 1 2 2 3 4.

LAG и LEAD

```
{LAG | LEAD}(значение [, смещение ] [, по умолчанию ])
 OVER (PARTITION BY .. ORDER BY ..)
```

- LAG и LEAD используются для сравнения значений полей текущей строки с полями другой строки, если изначально не известен порядок строк.
- Используют число, задающее насколько другая строка отстоит от текущей после формирования раздела.

10 - 16 SQL

LAG возвращает *значение* для строки, положение которой задаётся *смещением* от текущей строки к началу окна.

LEAD возвращает *значение* для строки, положение которой задаётся *смещением* от текущей строки к концу окна.

Если такой строки нет, возвращается значение *по_умолчанию* (оно должно иметь тот же тип, что и *значение*). Оба
параметра *смещение* и *по_умолчанию* вычисляются для текущей строки. Если они не указываются, то *смещение* по умолчанию равно 1, а *по_умолчанию* NULL

по_умолчанию задаёт возвращаемое значение функции, если *смещение* указывает за границу окна.

Функции работают быстрее, чем использования self-join.

В *значении* нельзя использовать аналитические функции, а выражения можно.

LAG и LEAD

```
SELECT time_id, SUM(amount_sold) AS SALES,
LAG(SUM(amount_sold),1) OVER (ORDER BY time_id) AS LAG1,
LEAD(SUM(amount_sold),1) OVER (ORDER BY time_id) AS LEAD1
FROM sales
WHERE time_id >= '2000-10-10' AND
      time_id <= '2000-10-14'
GROUP BY time_id;
```

```
TIME_ID                       SALES       LAG1       LEAD1
------------------------- ---------- ---------- ----------
10-OCT-00                    238,479                23,183
11-OCT-00                     23,183    238,479     24,616
12-OCT-00                     24,616     23,183     76,516
13-OCT-00                     76,516     24,616     29,795
14-OCT-00                     29,795     76,516
```


В первой строке используется смещение назад на 1. Но такой строки нет, так как текущая строка первая в окне, поэтому выдаётся NULL.

Функции могут использоваться для поиска пропусков в данных собранных по временным интервалам (gaps in time series).

SELECT * FROM (

SELECT gropid, year, month, seq, seq-lag(seq,1)

OVER (PARTITION BY groupid, year, month ORDER BY seq) AS gap FROM таблица) AS t

WHERE NOT (t.gap=1)

ORDER BY groupid, year, month, seq

Пример использования LAG и LEAD с несколькими таблицами:

```
SELECT t.calendar_month_desc,
MAX (t.calendar_month_name) MONTH_NAME ,SUM(s.quantity_sold) AS
   quantity_sold,
LAG (SUM(s.quantity_sold),12) OVER (ORDER BY t.calendar_month_desc )
   AS year_before,
LEAD(SUM(s.quantity_sold),12) OVER (ORDER BY t.calendar_month_desc )
   AS year_after
FROM sh.sales s, sh.times t
WHERE s.time_id=t.time_id
GROUP BY t.calendar_year, t.calendar_month_desc;
```

FIRST_VALUE и LAST_VALUE

- Выбирают первую или последнюю строку окна
- В следующем примере PARTITION BY не используется и раздел один:

```
SELECT department_id, last_name, salary,
FIRST_VALUE(last_name) OVER (ORDER BY salary ) AS F
FROM (SELECT * FROM employees
      WHERE department_id = 60
      ORDER BY employee_id) a;
```

```
DEPARTMENT_ID LAST_NAME                     SALARY FIRST_VALUE1
------------- ------------------------- ---------- -------------------------
           60 Lorentz                         4200 Lorentz
           60 Austin                          4800 Lorentz
           60 Pataballa                       4800 Lorentz
           60 Ernst                           6000 Lorentz
           60 Hunold                          9000 Lorentz
```

10 - 18 SQL

В примере на слайде для сотрудников 60 отдела выдаётся имя сотрудника с наименьшей зарплатой.

В стандарте SQL функциям lead, lag, first_value, last_value и nth_value можно указать параметр IGNORE NULLS.

FIRST_VALUE (last_name) IGNORE NULLS **OVER (..);**

Если результат функции NULL (last_name пустое), то она выдает следующую строку с непустым значением. Если все строки пусты, то выдает NULL.

В БД Postgres IGNORE NULLS не реализован.

В БД Oracle IGNORE NULLS реализован.

NTH_VALUE

возвращает *значение*, вычисленное в *н-ой* строке в рамке окна (считая с 1), или NULL, если такой строки нет

```
SELECT department_id, manager_id, MIN(salary),
NTH_VALUE(MIN(salary), 2)
OVER (PARTITION BY department_id ORDER BY manager_id) NV
FROM employees GROUP BY department_id, manager_id;
```


NTH_VALUE

возвращает *значение*, вычисленное в *н-ой* строке в рамке окна (считая с 1), или NULL, если такой строки нет.

В Postgres функция nth_value не поддерживает предусмотренные стандартом SQL параметры FROM FIRST и FROM LAST: реализовано только поведение по умолчанию (с подразумеваемым параметром FROM FIRST). (Получить эффект параметра FROM LAST можно, изменив порядок ORDER BY на обратный.)

В БД Oracle полный синтаксис:

```
NTH_VALUE (<expr>, <n>)} [FROM FIRST | FROM LAST]
[RESPECT NULLS | IGNORE NULLS] OVER (..)
```

LISTAGG/STRING_AGG

- Oracle `LISTAGG`:

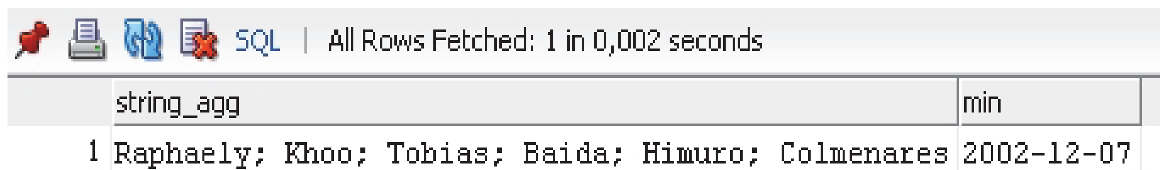
```
SELECT LISTAGG(last_name, '; ')
   WITHIN GROUP (ORDER BY hire_date,
   last_name),
 MIN(hire_date)
 FROM employees WHERE department_id=30;
```

- PostgreSQL `STRING_AGG`:

```
SELECT STRING_AGG(last_name, '; ' ORDER
 BY hire_date,last_name),
 MIN(hire_date)
 FROM employees WHERE department_id=30;
```


Второй параметр функций – разделитель. `ORDER BY` – порядок конкатенации. Значения `NULL` в первом аргументе (`last_name`) при конкатенации игнорируются.

В первом примере на слайде выдаёт сотрудников в порядке hire date и last name columns. Результат команд на слайде:

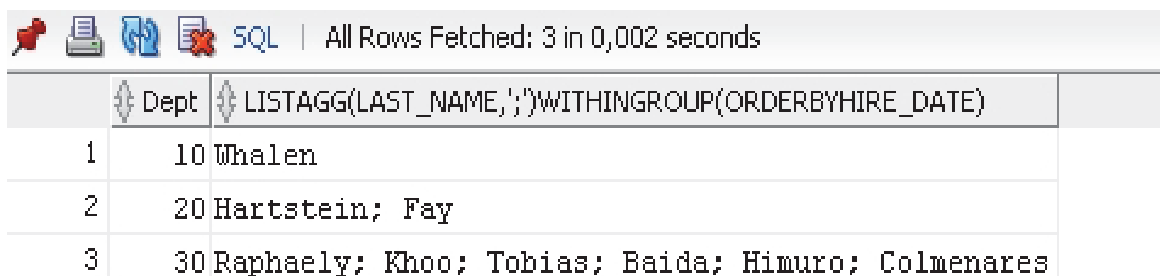


All Rows Fetched: 1 in 0,002 seconds

	string_agg	min
1	Raphaely; Khoo; Tobias; Baida; Himuro; Colmenares	2002-12-07

В следующем примере выдаёт сотрудников в порядке hire_date:

```
SELECT department_id "Dept",
LISTAGG(last_name, '; ') WITHIN GROUP (ORDER BY hire_date)
FROM employees
WHERE department_id IN(10, 20, 30)
GROUP BY department_id;
```



All Rows Fetched: 3 in 0,002 seconds

	Dept	LISTAGG(LAST_NAME,';')WITHINGROUP(ORDERBYHIRE_DATE)
1	10	Whalen
2	20	Hartstein; Fay
3	30	Raphaely; Khoo; Tobias; Baida; Himuro; Colmenares

Практическое задание 10

- Использование аналитических функций

10 - 21 SQL

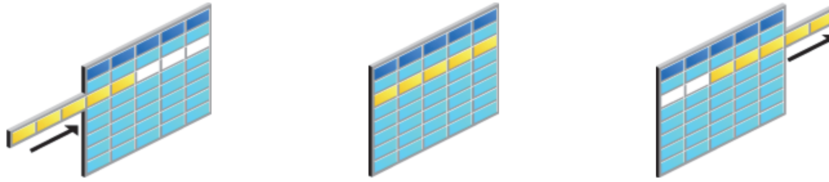
Практическое задание 10

Использование аналитических функций

11

Внесение изменений в данные

Изменение данных



```
INSERT INTO employees VALUES
(9999,'Bob','Builder','bob@fors.su',NULL,CURRENT_DATE,
 'IT_PROG',NULL,NULL,100,90);
1 row created.

UPDATE employees SET SALARY=8000
 WHERE EMPLOYEE_ID = 777;
1 row updated.

DELETE from employees WHERE EMPLOYEE_ID = 777;
1 row deleted.
```


Изменение данных

Манипулирование данными в базе данных можно производить с помощью команд INSERT, UPDATE, DELETE, MERGE. Эти команды будут рассмотрены на этом занятии.

Команды изменения данных

- Команды изменения выполняются при следующих операциях:
  - Вставка новых строк в таблицу
  - Изменение существующих строк в таблице
  - Удаление существующих строк из таблицы

11 - 3 SQL

Команды изменения данных

DML – обозначение команд INSERT, UPDATE, DELETE, MERGE в терминах БД Oracle.

Синтаксис команды INSERT

- **Для вставки новых строк в таблицу используется команда INSERT:**

```
INSERT INTO   таблица [(столбец [, столбец...])]
VALUES        (значение [, значение...]);
```

- **Этот синтаксис позволяет вставлять в таблицу только по одной строке**

11 - 4 SQL

Синтаксис команды INSERT

Вставить новые строки в таблицу можно с помощью команды INSERT.

Команда INSERT с предложением VALUES позволяет вставлять в таблицу только по одной строке.

Вставка новых строк

- Вставка новой строки, содержащей значения для каждого из столбцов.
- Значения указываются в стандартном порядке столбцов таблицы (используемом по умолчанию).
- Перечисление столбцов в предложении INSERT необязательно.

```
INSERT INTO departments(department_id,
       department_name, manager_id, location_id)
VALUES (70, 'Public Relations', 100, 1700);
1 row created.
```

- Символьные значения и даты заключаются в апострофы.

11 - 5 SQL

Вставка новых строк

Если вставляемая строка содержит значения для всех столбцов, то перечисление столбцов в предложении INSERT необязательно. Следует, однако, помнить о том, что последовательность самих значений должна соответствовать последовательности столбцов в этой таблице.

Для ясности лучше указывать список столбцов в команде INSERT.
Символьные значения и даты обрамляются апострофами; числовые значения в апострофы не заключаются, так как при этом будет выполняться неявное преобразование в тип данных NUMERIC.

Вставка строк с неопределенными значениями

- **Неявный метод: столбец не указывается в списке столбцов.**

```
INSERT INTO   departments (department_id,
                           department_name  )
VALUES        (30, 'Purchasing');
1 row created.
```

- **Явный метод: использование ключевого слова NULL в списке VALUES.**

```
INSERT INTO   departments
VALUES        (100, 'Finance', NULL, NULL);
1 row created.
```

11 - 6 SQL

Вставка строк с неопределенными значениями

Метод	Описание
Неявный	Исключение столбца из списка вставляемых столбцов
Явный	Задание ключевого слова NULL в списке VALUES. Задание пустой строки ('') в списке VALUES; только для символьных значений и дат .

Убедитесь в том, что столбец допускает неопределенные значения и не нарушаются другие ограничения целостности.

Столбец, не заданный явно, получает в новой строке неопределенное значение или значение по умолчанию, установленное на уровне таблицы для этого столбца.

Распространенные ошибки, которые могут возникнуть при вводе пользовательских данных:

- Опущено обязательное значение для столбца с ограничением NOT NULL.
- Дублирующее значение нарушает ограничение уникального ключа.
- Нарушение ограничения внешнего ключа.
- Нарушение ограничения CHECK.
- Несовпадение типа данных столбца таблицы и вставляемого значения.
- Вставляемое значение не помещается по ширине столбца.

Вставка специальных значений

Функция CURRENT_DATE записывает текущие дату и время.

```
INSERT INTO employees (employee_id,
                 first_name, last_name,
                 email, phone_number,
                 hire_date, job_id, salary,
                 commission_pct, manager_id,
                 department_id)
VALUES           (113,
                 'Louis', 'Popp',
                 'LPOPP', '515.124.4567',
                 CURRENT_TIMESTAMP, 'AC_ACCOUNT', 6900,
                 NULL, 205, 100);
1 row created.
```


Вставка специальных значений

Для вставки специальных значений в таблицу можно использовать функции.

В примере на слайде в таблицу EMPLOYEES заносится информация о служащем по фамилии Popp. В столбец HIRE_DATE заносятся текущая дата и время. Для получения текущей даты и времени используется функция CURRENT_DATE.

При вставке строк в таблицу можно использовать и другие функции, например, функцию USER . Она записывает текущее имя пользователя, подсоединившегося к БД.

Вставка конкретных значений даты и времени

- **Добавление нового служащего**

```
INSERT INTO employees
VALUES      (500,
             'Den', 'Raphealy',
             'DRAPHEALA', '515.127.4561',
             TIMESTAMP '1999-02-03 20:38:40',
             'AC_ACCOUNT', 11000, NULL, 100, 30);
1 row created.
```

- **Проверка вставки**

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_P
114	Den	Raphealy	DRAPHEAL	515.127.4561	03-FEB-99	AC_ACCOUNT	11000	

11 - 8 SQL

Вставка конкретных значений даты и времени

Пример на слайде показывает включение в таблицу EMPLOYEES служащего по фамилии Raphealy. Для столбца HIRE_DATE задано значение 3 февраля 1999 г.

Использование подзапросов при внесении изменений

В командах DML можно использовать подзапросы:

- **Копирование данных из одной таблицы в другую**
- **Выборка данных из вложенного представления**
- **Изменение данных в одной таблице на основе значений в другой таблице**
- **Удаление данных в одной таблице на основе значений в другой таблице**

11 - 9 SQL

Использование подзапросов при манипулировании данными

Подзапросы можно использовать в команде `INSERT` для выбора данных из одной таблицы и вставки в другую таблицу. Таким простым способом с помощью одной вложенной команды `SELECT` копируются большие объемы данных из одной таблицы в другую.

Подобным образом при массовых изменениях и удалениях подзапросы указываются в предложении `WHERE` команд `UPDATE` и `DELETE`.

Кроме того, подзапросы применяются в предложении `FROM` команды `SELECT`. Такие подзапросы называются *вложенными представлениями* (*inline views*).

Примечание

При обработке больших объемов данных БД могут использовать оптимизации, существенно ускоряющие команду. Например, в БД Oracle используется режим прямых вставок (direct-path insert). Также часть данных могут не удовлетворять ограничениям целостности и в БД есть опции команд, которые позволяют игнорировать ошибки. Например, в **БД Oracle** опция INSERT..LOG_ERRORS INTO.., но при её использовании режим прямых вставок не работает.

Копирование строк из другой таблицы

- **Команда `INSERT` должна включать подзапрос**

```
INSERT INTO sales_reps(id, name, salary, commission_pct)
  SELECT employee_id, last_name, salary, commission_pct
  FROM   employees
  WHERE  job_id LIKE '%REP%';

4 rows created.
```

- **Предложение `VALUES` не используется.**
- **Количество столбцов, указанных в предложении `INSERT`, должно совпадать с количеством столбцов в подзапросе.**

11 - 10 SQL

Копирование строк из другой таблицы

Для вставки строк в таблицу с помощью команды `INSERT` можно использовать значения из другой таблицы. Для этого вместо предложения `VALUES` используется подзапрос.

Количество столбцов и типы данных в списке столбцов в команде `INSERT` должны совпадать с количеством значений и соответствующими типами данных в подзапросе. Чтобы скопировать строки в таблицу, используйте в подзапросе предложение `SELECT *` :

```
INSERT INTO copy_emp
  SELECT *
  FROM   employees;
```

Синтаксис команды UPDATE

- Для обновления существующих строк используется команда UPDATE:

```
UPDATE   таблица
SET      столбец = значение [, столбец = значение, ...]
[WHERE   условие];
```

- В случае необходимости можно одновременно обновлять несколько строк

11 - 11 SQL

Синтаксис команды UPDATE

Изменять существующие строки можно с помощью команды UPDATE.

Синтаксис:

таблица	имя таблицы
столбец	имя обновляемого столбца таблицы
значение	соответствующее значение или подзапрос для столбца
условие	показывает, какие строки должны быть изменены; состоит из имен столбцов, выражений, констант, подзапросов и операторов сравнения.

Проверьте результаты обновления таблицы путем выборки измененных строк.

Примечание

Обычно для идентификации отдельной строки используется главный ключ. Использование с этой целью других столбцов может привести к неожиданному обновлению нескольких строк вместо одной. Например, в таблице EMPLOYEES опасно распознавать строки по именам служащих, т.к. они могут повторяться.

Обновление строк в таблице

- Предложение WHERE позволяет изменить конкретную строку или строки:

```
UPDATE employees
SET    department_id = 70
WHERE  employee_id = 113;
1 row updated.
```

- Если предложение WHERE отсутствует, обновляются все строки таблицы

```
UPDATE   copy_emp
SET      department_id = 110;
22 rows updated.
```


Обновление строк (продолжение)

Если задано предложение WHERE , то команда UPDATE изменяет конкретную строку или строки. В вышеуказанном примере служащий под номером 113 (Popp) переводится в отдел 70.

Если предложение WHERE отсутствует, в таблице изменяются все строки.

Обновление двух столбцов с помощью подзапроса

Изменение должности и оклада служащего под номером 114, чтобы они стали такими же, как у служащего под номером 205.

```
UPDATE   employees
SET      job_id   = (SELECT  job_id
                     FROM    employees
                     WHERE   employee_id = 205),
         salary   = (SELECT  salary
                     FROM    employees
                     WHERE   employee_id = 205)
WHERE    employee_id    =  114;
1 row updated.
```


Обновление двух столбцов с помощью подзапроса

Подзапросы можно использовать в предложении `SET` команды `UPDATE` для изменения нескольких столбцов.

Если не было изменено ни одной строки, выдается сообщение "`0 rows updated.`"

Обновление строк на основе значений из другой таблицы

Для изменения строк таблицы на основе значений из другой таблицы используйте подзапросы в командах UPDATE

```
UPDATE   copy_emp
SET      department_id  =  (SELECT department_id
                             FROM employees
                             WHERE employee_id = 100)
WHERE    job_id         =  (SELECT job_id
                             FROM employees
                             WHERE employee_id = 200);
1 row updated.
```


Обновление строк на основе значений из другой таблицы

Для обновления строк таблицы можно использовать подзапросы в команде UPDATE. В примере на слайде таблица COPY_EMP обновляется на основе значений из таблицы EMPLOYEES. Номер отдела всех служащих, имеющих такую же должность, как служащий под номером 200, меняется на текущий номер отдела служащего номер 100.

Явное задание значения DEFAULT

- При использовании значений по умолчанию, можно задавать ключевое слово DEFAULT в качестве устанавливаемого значения столбца
- Это возможность решать, где и когда следует использовать значение по умолчанию
- Ключевое слово DEFAULT можно указывать в командах INSERT и UPDATE.

11 - 15 SQL

Явное задание атрибута значения по умолчанию

Ключевое слово DEFAULT используется в командах INSERT и UPDATE для ссылки на определенное по умолчанию значение столбца. Если значение по умолчанию не установлено, применяется неопределенное значение.

Без опции DEFAULT значение по умолчанию необходимо либо явно указывать в коде программы, либо выбирать из словаря данных. В первом случае при изменении значения по умолчанию следует последовательно вносить исправления в код. Вторая возможность (выбор из словаря данных) обычно не предоставляется программам приложения. Поэтому опция DEFAULT – важная возможность.

Явное задание значения с помощью DEFAULT

- DEFAULT в команде INSERT:

```
INSERT INTO deptm3
  (department_id, department_name, manager_id)
VALUES (300, 'Engineering', DEFAULT);
```

- DEFAULT в команде UPDATE:

```
UPDATE deptm3
SET manager_id = DEFAULT
WHERE department_id = 10;
```

11 - 16 SQL

Явноге задание значения с помощью DEFAULT

Ключевое слово DEFAULT используется для задания значения, которое было ранее определено в качестве значения по умолчанию для столбца. Если значение по умолчанию не определено для соответствующего столбца, Oracle устанавливает неопределенное значение.

В первом приведенном примере в команде INSERT используется значение по умолчанию для столбца MANAGER_ID. Если значение по умолчанию не определено для этого столбца, будет введено неопределенное значение.

Во втором приведенном примере используется команда UPDATE для изменения значения столбца MANAGER_ID в строках отдела 10 на значение по умолчанию. Если значение по умолчанию не определено для этого столбца, при изменение будут использованы неопределенные значения.

Примечание

При создании таблицы можно определить значение по умолчанию для столбца.

Команда DELETE

Для удаления строк используется команда DELETE

```
DELETE [FROM]   таблица
[WHERE          условие];
```


Удаление строк

Удалить существующие строки можно с помощью команды DELETE.

Синтаксис:

таблица	имя таблицы
условие	определяет строки, которые должны быть удалены; состоит из имен столбцов, выражений, констант, подзапросов и операторов сравнения

Примечание

Если ни одна строка не была удалена, выдается сообщение “0 rows deleted.”

Удаление строк из таблицы

- **Конкретная строка или строки удаляются с помощью предложения WHERE**

```
 DELETE FROM departments
 WHERE  department_name = 'Finance';
1 row deleted.
```

- **Если предложение WHERE отсутствует, удаляются все строки таблицы**

```
DELETE FROM  copy_emp;
22 rows deleted.
```

11 - 18 SQL

Удаление строк из таблицы

Удалить из таблицы конкретную строку или строки можно с помощью предложения WHERE в команде DELETE. В примере на слайде из таблицы DEPARTMENTS удаляется отдел Finance. Проверить результаты операции удаления можно путем выборки строк по команде SELECT.

```
SELECT   *
 FROM     departments
 WHERE    department_name = 'Finance';

 no rows selected.
```

Если предложение WHERE отсутствует, удаляются все строки таблицы. Во втором примере на слайде удаляются все строки из таблицы COPY_EMP, так как предложение WHERE не указано.

Пример

Удаление строк, заданных в предложении WHERE.

```
DELETE FROM  employees WHERE employee_id = 114;
1 row deleted.

DELETE FROM  departments WHERE department_id IN (30, 40);
2 rows deleted.
```

Удаление строк на основе значений из другой таблицы

Для удаления строк на основе значений из другой таблицы используйте подзапросы в командах DELETE

```
DELETE FROM employees
WHERE  department_id =
                (SELECT department_id
                 FROM   departments
                 WHERE  department_name
                        LIKE '%Public%');
1 row deleted.
```

11 - 19 SQL

Удаление строк на основе значений из другой таблицы

Для удаления строк таблицы на основе значений из другой таблицы можно использовать подзапросы. В вышеуказанном примере удаляются все служащие отдела, в наименовании которого имеется строка “Public.” Сначала подзапрос просматривает таблицу DEPARTMENTS для выяснения номера отдела, имя которого содержит строку “Public.” Затем номер отдела передается в главный запрос, который удаляет данные из таблицы EMPLOYEES в зависимости от полученного номера отдела.

Команда TRUNCATE

- Удаляет все строки из таблицы, оставляя таблицу пустой с сохранением структуры таблицы
- Освобождает место, занимаемое таблицей
- Опция CASCADE очищает зависимые таблицы

- Синтаксис:

```
TRUNCATE TABLE table_name [CASCADE];
```

- Пример:

```
TRUNCATE TABLE copy_emp;
```

11 - 20 SQL

Команда TRUNCATE

Команда TRUNCATE (усечение) предоставляет быстрый способ удаления всех строк таблицы. Структура таблицы, ограничения целостности и другие служебные объекты не удаляются, в отличие от команды DROP.

Удаление строк с помощью команды TRUNCATE происходит быстрее, чем по команде DELETE. При выполнении этой команды не запускаются действия, связанные с командой DELETE (триггеры на удаление).

Опция CASCADE на родительской таблице усекает ссылающиеся на неё таблицы.

TRUNCATE на зависимой от другой таблице (foreign key) не выполнится даже с CASCADE. Перед усечением можно удалить ограничение целостности foreign key.

В БД Oracle - эта команда, как и все команды DDL не откатывается. Восстановить строки в БД Oracle после команды TRUNCATE выполненной ошибочно сложно, поэтому нужно аккуратно применять эту команду. Удалить часть строк командой TRUNCATE нельзя, если только это не связано со структурой таблицы: в секционированной таблице можно усечь секцию.

Команда MERGE

MERGE выполненяет операции INSERT и UPDATE в одной команде

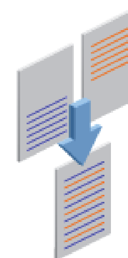
Workspace

Enter SQL, PL/SQL and SQL*Plus statements.

```
MERGE INTO jobs j
  USING (SELECT * FROM jobs_acquisition) a
  ON (j.job_id = a.job_id)
  WHEN MATCHED THEN UPDATE SET j.job_title = a.job_title
  WHEN NOT MATCHED THEN INSERT
    (j.job_id, j.job_title, j.min_salary, j.max_salary)
  VALUES (a.job_id, a.job_title, a.min_salary, a.max_salary)
```

Execute | Load Script | Save Script | Cancel

5 rows merged.



Команда MERGE

Команда MERGE появилась в стандарте SQL:2003 и реализована **в БД Oracle**.

Команда MERGE выполняет операции INSERT и UPDATE за один проход, что полезно при обработке большого количества строк. Использование этой команды позволяет объединять один источник с другим. При этом новые строки вставляются, а в уже существующих строках обновляются значения определенных столбцов.

Рассмотрим следующий пример, в котором показаны некоторые строки таблицы JOBS:

```
JOB_ID      JOB_TITLE   MIN_SALARY MAX_SALARY
---------- ------------ ---------- ----------
AD_PRES    President       20000      40000
FI_ACCOUNT Accountant       4200       9000
ST_CLERK   Stock Clerk      2000       5000
IT_PROG    Programmer       4000      10000
```

Команда MERGE (продолжение)

Ниже приведены данные таблицы JOBS_ACQUISITION:

```
JOB_ID     JOB_TITLE      MIN_SALARY MAX_SALARY
---------- -------------- ---------- ----------
AD_PRES    VP             20000      40000
DBA        DB Admin        4200       9000
SA         Sys Admin       2000       5000
```

Команда MERGE вставляет в таблицу JOBS строки с новыми должностями (JOB_ID) и обновляет столбец JOB_TITLE в существующей строке таблицы JOBS, если уже есть строка с такой должностью (JOB_ID). В результате название должности “President” будет изменено на “VP” и, кроме того, добавятся две строки для новых должностей “SA” и “DBA”.

Команда MERGE

- **Предоставляет возможность при определенных условиях изменять или вставлять данные в таблицу базы данных.**
- **Выполняет изменение (UPDATE), если строка существует и вставку (INSERT) для новой строки:**
  - **Позволяет избежать внесения отдельных изменений.**
  - **Повышает производительность и упрощает использование.**
  - **Эффективна для приложений, работающих с хранилищами данных.**

Команда MERGE

Использование этой команды позволяет изменять, вставлять или удалять строку при определенных условиях, что позволяет избежать применения нескольких команд UPDATE. Решение об изменении, вставке или удалении в целевую таблицу основывается на условии в предложении ON.

Так как команда MERGE объединяет команды INSERT и UPDATE, пользователю необходимы привилегии INSERT и UPDATE для целевой таблицы и привилегия SELECT для исходной таблицы. Чтобы задать фразу DELETE в предложении merge_update_clause, необходима объектная привилегия DELETE для целевой таблицы.

Команда MERGE обеспечивает определенность выполнения. Вы не можете при помощи одной и той же команды MERGE изменить несколько раз одну и ту же строку в целевой таблице.

Противоположный подход состоит в использовании циклов на процедурном языке и нескольких команд DML. В то же время команда MERGE – это одна команда SQL, которая удобна в использовании и имеет более простой вид.

Команда MERGE подходит для приложений, работающих с хранилищами данных. Например, в таких приложениях вам может понадобиться обработка данных, поступающих из многих источников, некоторые из которых повторяются. С помощью команды MERGE вы можете добавлять и изменять строки при определенных условиях.

Синтаксис команды MERGE

Команда MERGE позволяет вставлять или изменять строки при определенных условиях.

```
MERGE INTO имя_таблицы псевдоним_таблицы
  USING (таблица|представление|подзапрос) псевдоним
  ON (условие_соединения)
  WHEN MATCHED THEN
    UPDATE SET
    столбец1 = значение_столбца1,
    столбец2 = значение_столбца2
  WHEN NOT MATCHED THEN
    INSERT (список_столбцов)
    VALUES (список_столбцов);
```

11 - 24 SQL

Слияние строк

Изменить или вставить новые строки при определенных условиях можно, используя команду MERGE.

Синтаксис:

предложение INTO	определяет целевую таблицу, в которую производится вставка или изменение
предложение USING	определяет источники данных, которые используются для изменения или вставки; это может быть таблица, представление или подзапрос
предложение ON	условие, определяющее действие (изменение или вставка), которое выполняется по команде MERGE
WHEN MATCHED | WHEN NOT MATCHED	указывает БД, как реагировать на результаты условия соединения

Слияние строк

Вставка или изменение строк таблицы EMPL3 для установления соответствия с значениями таблицы EMPLOYEES.

```
MERGE INTO empl3  c
  USING employees e
  ON (c.employee_id = e.employee_id)
WHEN MATCHED THEN
  UPDATE SET
     c.first_name     = e.first_name,
     c.last_name      = e.last_name,
     ...
     c.department_id  = e.department_id
WHEN NOT MATCHED THEN
 INSERT VALUES(e.employee_id, e.first_name, e.last_name,
          e.email, e.phone_number, e.hire_date, e.job_id,
          e.salary, e.commission_pct, e.manager_id,
          e.department_id);
```


Пример слияния строк

```
MERGE INTO empl3 c
  USING employees e
  ON (c.employee_id = e.employee_id)
WHEN MATCHED THEN
  UPDATE SET
    c.first_name     = e.first_name,
    c.last_name      = e.last_name,
    c.email          = e.email,
    c.phone_number   = e.phone_number,
    c.hire_date      = e.hire_date,
    c.job_id         = e.job_id,
    c.salary         = e.salary,
    c.commission_pct = e.commission_pct,
    c.manager_id     = e.manager_id,
    c.department_id  = e.department_id
WHEN NOT MATCHED THEN
  INSERT VALUES(e.employee_id, e.first_name, e.last_name,
       e.email, e.phone_number, e.hire_date, e.job_id,
       e.salary, e.commission_pct, e.manager_id,
       e.department_id);
```

Слияние строк

```
TRUNCATE TABLE empl3;
SELECT *
FROM empl3;
no rows selected
```

```
MERGE INTO empl3 c
  USING employees e
  ON (c.employee_id = e.employee_id)
WHEN MATCHED THEN
  UPDATE SET
     ...
WHEN NOT MATCHED THEN
 INSERT VALUES...;
```

```
SELECT *
FROM empl3;

107 rows selected.
```

11 - 26 SQL

Пример слияния строк (продолжение)

В примере на слайде сопоставляются значения столбцов EMPLOYEE_ID в таблицах EMPL3 и EMPLOYEES. При совпадении значений строка в таблице EMPL3 обновляется так, чтобы соответствовать строке в таблице EMPLOYEES. В противном случае, когда нет совпадающих значений, строка вставляется в таблицу EMPL3.

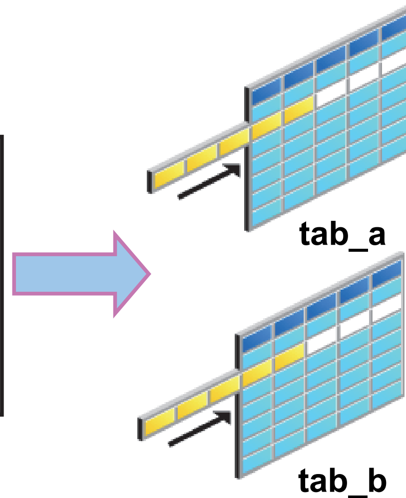
Проверяется условие c.employee_id = e.employee_id. Так как таблица EMPL3 не заполнена, условие возвращает значение ложно (false), т.е. нет соответствий. Выполняется предложение WHEN NOT MATCHED и по команде MERGE вставляются строки из таблицы EMPLOYEES в таблицу EMPL3.

Если существуют строки в таблице EMPL3 и номер служащего (employee_id) совпадает в обоих таблицах (EMPL3 и EMPLOYEES), существующие строки таблицы EMPL3 изменяются так, чтобы соответствовать строкам таблицы EMPLOYEES.

Обзор многотабличных команд

- **Синтаксис Oracle**

```
INSERT ALL
WHEN sal>10000 THEN
 INTO tab_a VALUES(emp_id, sal)
WHEN mgr>200 THEN
 INTO tab_b VALUES(emp_id, mgr)
SELECT * FROM emp;
```



Обзор многотабличных команд

В многотабличной команде `INSERT` можно в подзапросах производить вычисления значений вставляемых строк и добавлять строки в одну или несколько таблиц.

Многотабличная команда `INSERT` полезна в различных ситуациях при эксплуатации хранилищ данных. Так для поддержки решения задач анализа бизнеса требуется регулярно загружать информацию в хранилище данных. Эта информация должна быть извлечена из одной или нескольких оперативных систем и перенесена в хранилище данных.

Синтаксис многотабличных команд в разных БД различны.

Обзор многотабличных команд

- **Команда может быть использована для вставки в несколько таблиц с помощью одной команды**
- **Многотабличные команды могут использоваться в системах поддержки хранилищ данных для передачи информации из одного или более источников данных в набор целевых таблиц**
- **Такие команды обеспечивают существенное повышение производительности за счет:**
  - **использования одной команды вместо нескольких команд `INSERT...SELECT` ;**
  - **использования одной команды DML вместо процедуры, в которой выполняются вставки при условии (`IF...THEN`).**

11 - 28 SQL

Обзор многотабличных команд

Преимущество многотабличных команд состоит в возможности использования операции для внесения изменений в несколько целевых таблиц. Например, без них нужно выполнять *n* независимых команд `INSERT ... SELECT`, что требовало обработки одного и того же источника данных *n* раз и увеличивало рабочую нагрузку при преобразовании в *n* раз.

Каждая исходная строка может быть преобразована в несколько строк таблиц БД.

Практическое задание 11

- Вставка строк в таблицы
- Обновление и удаление строк в таблице

Практическое задание 11

На этом практическом занятии вы будете добавлять строки в таблицу, обновлять данные, удалять данные из таблицы..

12 Транзакции

Транзакции базы данных

Содержат набор команд, меняющих данные в БД

Обладают свойствами “ACID”:

- **Atomicity (атомарность) – будут либо выполнены (commited - зафиксированы) все команды, либо не выполнено ни одной (rollback - отменены)**
- **Consistency (согласованность) - фиксация транзакции сохраняет согласованность базы данных**
- **Isolation (изоляцию) - транзакции выполняющиеся в то же время не должны оказывать влияние друг на друга**
- **Durability (устойчивость к сбоям) - если транзакция получила подтверждение от БД, что она выполнена, результат не будет отменён из-за сбоя БД**

12 - 2 SQL

Транзакции базы данных

Обеспечивают наиболее надёжную и предсказуемую работу с данными. Свойства ACID были сформулированы Джеймсом Греем в конце 70-х.

Транзакции открываются в сессиях (соединениях) с БД.

Транзакция, фиксирующая свои результаты, сохраняет согласованность БД (не нарушаются ограничения целостности БД). В банковской системе может существовать требование равенства суммы, списываемой с одного счёта, сумме, зачисляемой на другой. Это бизнес-правило можно реализовать объединив две операции списания и зачисления в одну транзакцию.

В ходе выполнения транзакции согласованность не требуется. Списание и зачисление будут, скорее всего, двумя разными командами. Внутри транзакции будет видно несогласованное состояние данных. Однако благодаря изоляции, никаким другим сессиям эта несогласованность не будет видна. Атомарность гарантирует, что транзакция либо будет полностью завершена, либо ни одна из операций транзакции не будет выполнена. Тем самым эта промежуточная несогласованность является скрытой.

Атомарность

Атомарность в БД реализуется командами COMMIT и ROLLBACK. Транзакция завершается одной из этих команд

- **Commit: делает изменения постоянными**
- **Rollback: отменяет изменения**

12 - 3 SQL

Атомарность

Атомарность в БД реализуется командами COMMIT и ROLLBACK. Транзакция завершается одной из этих команд.

Перед выполнением `COMMIT` или `ROLLBACK` изменения находятся в отложенном состоянии. Только внутри сессии, делающей изменения, позволено их видеть и только в рамках (от момента начала до момента фиксации или отката) транзакции. Остальные сессии, запрашивающие данные, получают их в том виде, в котором они были перед внесением изменений. Остальные сессии не могут выполнить команды DML и менять те же самые данные, которые уже меняет какая-либо транзакция.

По умолчанию сессия, пытающийся изменить ту же самую строку, которую меняет другая сессия, будет ждать, пока начавшая менять строку сессия либо зафиксирует, либо откатит изменения. Это контролируется автоматически механизмом блокирования в БД.

Согласованность чтения

- Согласованность гарантирует непротиворечивость данных друг другу и ограничениям целостности БД в любой момент времени
- Изменения, сделанные одной транзакцией, не вступают в противоречие с изменениями, сделанными другой транзакцией
- Гарантируется, что для одних и тех же данных:
  - “Читатели“ не блокируют “писателей“.
  - “Писатели“ не блокируют “читателей“.

12 - 4 SQL

Согласованность чтения

В сессиях с БД могут выполняться:

- Операция чтения - одиночная команда SELECT.
- Операции чтения - набор команд SELECT, желающих получить данные на один момент времени (целостный образ данных)
- Одна или несколько операций записи (команды INSERT, UPDATE, DELETE).
- Операции чтения и записи, выполняющиеся в рамках одной транзакции.

Согласованность чтения (*read consistency*) обеспечивает следующее:

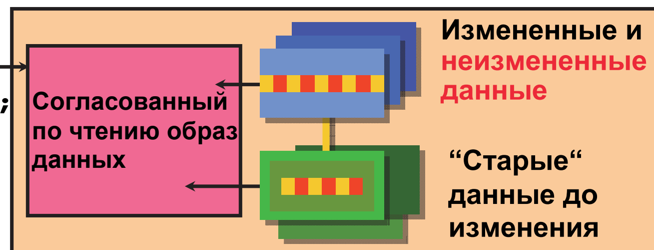
- Операции чтения видит образ данных на один момент времени
- Сессии, осуществляющие чтение, не видят данных, которые находятся в процессе изменения другими сессиями.
- Транзакция, осуществляющая запись, вносит свои изменения согласованно между операциями внутри неё.

Реализация согласованности

Пользователь A

```
UPDATE employees
SET    salary = 7000
WHERE  last_name = 'Grant';
```

```
SELECT  *
FROM userA.employees;
```



Пользователь B

Реализация согласованности

Данные согласованы, если они выдаются на один момент времени, независимо от того сколько времени обрабатываются команды, обратившиеся к данным.

Для одиночных команд SQL согласованность чтения (выдача данных на один момент) обеспечивается всегда. SELECT может работать долго и не блокирует писателей, они могут внести изменения в прочитанные и не прочитанные SELECTом данные. Например, создать отдел в таблице DEPT и внести в него сотрудников в таблице EMPLOYEES. SELECT при соединении таблиц может увидеть, что в DEPT отдела нет, а в EMPLOYEES есть сотрудники в отделе, которого нет. Это нарушение целостности данных. БД такого не допускают.

БД реализуют согласованность чтения с помощью внутреннего механизма, называющегося многоверсионность (MVCC - MvultiVersion Concurrency Control). Образ данных на один момент времени называется моментальный снимок - snapshot, по аналогии с фотографией, на которой запечатлён меняющийся предмет.

Пока изменения не будут зафиксированы в базе данных, измененные данные видит только сессия, которая эти изменения осуществляет. Благодаря этому сессии, читающие данные, видят непротиворечивые данные.

После фиксации транзакции изменение, внесенное в базу данных, становится видимым для других сессий.

Уровни изоляции транзакций

Уровни изоляции транзакций

Свойство изоляции транзакций в БД имеет несколько уровней. Высокая изоляция снижает производительность – возможность одновременной работы большому количеству пользователей, так как: возникают задержки и БД тратит много ресурсов на реализацию высокого уровня изоляции. Низкая изоляция приводит к побочным эффектам и ошибкам, которые выявляются после сдачи системы в эксплуатацию. Стандарт SQL определяет четыре уровня изоляции транзакций.

По умолчанию БД Oracle и БД postgres использует уровень READ COMMITTED.

Уровень можно выбрать на уровне транзакции:

```
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
```

БД позволяют установить уровень на уровне сессии, который будет использоваться по умолчанию для последующих транзакций. Синтаксис таких команд нестандартизован и различается в БД разных производителей.

В БД Oracle:

```
ALTER SESSION SET ISOLATION_LEVEL SERIALIZABLE;
```

Уровень READ COMMITED

Наиболее часто используемый

Достаточен для большинства действий

Устанавливается программно или командой

```
SET TRANSACTION ISOLATION LEVEL READ
COMMITTED;
```

Не обеспечивает повторяемость чтения и отсутствие фантомных строк

12 - 7 SQL

Уровень READ COMMITTED

В транзакции, работающей на уровне READ COMMITTED, команды SQL видят только те данные, которые были зафиксированы в других транзакциях до начала этой команды. Команда может выполняться долгое время, за которое транзакции в других сессиях успеют зафиксировать данные, но команда выдаст образ данных, зафиксированных до момента её начала. Образ данных это моментальный снимок (snapshot). Изменения, которые были выполнены, но не зафиксированы транзакцией, в которой выполняется команда, будут ей видны, даже если они не были зафиксированы.

БД Oracle и PostgreSQL обеспечивают возможность выборки старых данных, сохраняя образ старых данных в служебных структурах. Такая реализация позволяет не устанавливать какие-либо блокировки при выполнении команд выборки данных.

Если в этом режиме выполнить один и тот же SELECT два раза в той же транзакции, результат выполнения может отличаться. Это называется проблемой неповторяющегося чтения, когда при повторном чтении в рамках одной транзакции ранее прочитанные строки содержат другие данные. Также может выдаваться разное количество строк (строки, удовлетворяющие критерию выборки SELECT были удалены или вставлены другими транзакциями), такое поведение называется фантомное чтение.

Поэтому, выполнять на уровне READ COMMITTED последовательно команды SELECT, сохранять (кэшировать) результаты в приложении и использовать сохраненные в приложении данные не проверяя, изменились ли они с момента выборки не нужно. Все необходимые данные лучше выбирать одной командой, тогда образ данных из разных таблиц будет согласованным на один момент времени, либо нужно использовать уровень изоляции транзакции SERIALIZABLE.

Примечание

в БД может быть реализован ещё один уровень изоляции - REPEATABLE READ. Уровень REPEATABLE READ не защищает от фантомного чтения. Уровень REPEATABLE READ может быть реализован на уровне READ COMMITED путем блокировки всех читаемых в этой транзакции строк (добавление FOR UPDATE ко всем SELECT).

Уровень SERIALIZABLE

- Наиболее строгий уровень изоляции транзакций друг от друга
- Устанавливается программно или командой
  `SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;`
- Транзакция начинается на момент выдачи первой команды, в том числе SELECT
- Все команды видят данные на момент начала транзакции
- Транзакция завершается командой COMMIT или ROLLBACK
- Транзакция может состоять только из SELECTов

12 - 8 SQL

Уровень SERIALIZABLE

Это наиболее строгий уровень изоляции транзакций друг от друга. На этом уровне всем командам внутри транзакции виден образ данных на момент начала транзакции.
Без необходимости использовать этот уровень не нужно, так как попытка внести изменение в данные, которые были зафиксированы в других транзакциях приведёт к ошибке:

```
- Cannot serialize access for this transaction
- не удалось сериализовать доступ из-за зависимостей чтения/записи
между транзакциями
```

БД проверяет будут ли данные после выполнения транзакций этого уровня такими же, если перекрывающиеся по времени транзакции этого уровня выполнять последовательно в разных порядках. Если результат будет разным, БД выдаст эту же ошибку. Обнаружить это БД может при выполнении любой команды в любой транзакции этого уровня, не только при фиксации.
В логике приложения нужно предусматривать обработку такой ошибки: откатывать транзакцию, которой БД выдала ошибку и, возмоно, пытаться повторить транзакцию снова.
БД проверяет побочные эффекты (аномалии) сериализации только между транзакциями этого уровня, перекрывающиеся по времени.
Перекрытие по времени может обнаружиться под нагрузкой: когда много параллельных транзакций и они выполняются дольше.
Уровень Serializable используют в редких случаях для реализации специфической логики приложения. При этом, отслеживание зависимостей чтения-записи в режиме Serializable создаёт дополнительную нагрузку на БД, как и повторение приложением транзакций, не зафиксированных из-за ошибки сериализации, которая приведена выше.
В JDBC autocommit=true по умолчанию и отключается программно “conn.setAutoCommit(false);”

Аномалии сериализации

Если в транзакциях используется считывание значений строк и использование этих значений при изменении строк таблиц и эти транзакции перекрываются по времени, то есть вторая транзакция началась до момента фиксации первой транзакции, то уровень SERIALIZABLE, установленный в обеих транзакциях защищает от "аномалии сериализации" - одна из транзакций получит ошибку.
БД могла бы выполнить такие одновременно начавшиеся транзакции: вторую за первой или первую за второй, результат был бы различным.
Например:

```
create table mytab (class numeric, value numeric);
insert into mytab values (1,10);
insert into mytab values (2,100);
```

Транзакция A:

```
INSERT INTO MYTAB VALUES (2,SELECT SUM(value) FROM mytab WHERE
class = 1);
```

Транзакция B:

```
INSERT INTO MYTAB VALUES (1,(SELECT SUM(value) FROM mytab WHERE
class = 2));
```

Если A выполнится раньше B, то вставятся данные (2,10).
Если B выполнится раньше A, то вставятся данные (1,110).

Если обе транзакции работают на уровне SERIALIZABLE, то БД выдаст ошибку при фиксации транзакции, которая фиксируется позже.
Таким образом, уровень SERIALIZABLE эквивалентен выстраиванию перекрывающихся по времени транзакций в очередь: начинается и фиксируется первая транзакция, потом начинается и фиксируется вторая транзакция.
БД Oracle под названием SERIALIZABLE реализует нестандартный уровень SNAPSHOT ISOLATION, подверженный аномалиям.

Автономные транзакции

Есть в БД Oracle

Это транзакции, которые открываются когда уже открыта транзакция и фиксируются независимо от того зафиксируется или откатится родительская (внешняя) транзакция

Выполняются при вызове программного кода: функций и процедур

12 - 10 SQL

Автономные транзакции

- это транзакции, которые открываются когда уже открыта транзакция и фиксируются независимо от того зафиксируется или откатится родительская (внешняя) транзакция

Выполняются при вызове программного кода: функций и процедур.

В коде SQL нет команд открытия автономных транзакций.

В БД postgres не реализованы, но могут эмулироваться через соединения с базами данных.

Блокировки

- Препятствуют одновременному изменению одних и тех же данных разными транзакциями
- Автоматически устанавливаются на самом низком уровне для данной команды
- Не расширяют конфликты с другими транзакциями.

Транзакция 1

```
SQL> UPDATE employees
  2   SET salary=salary+100
  3   WHERE employee_id=100;
```

Транзакция 2

```
SQL> UPDATE employees
  2   SET salary=salary*1.1
  3  WHERE employee_id=100;
```


Блокировки

Перед тем, как база данных разрешит сеансу обновить данные, они должны быть сначала заблокированы этим сеансом. С помощью блокировки сеанс получает монопольный контроль над данными и никакие другие транзакции не могут изменить заблокированные данные, пока блокировка не будет снята.

Транзакции могут блокировать отдельные строки данных, несколько строк и даже целые таблицы. БД поддерживают как ручное, так и автоматическое блокирование. Автоматически накладываемые блокировки всегда выбирают наименьший уровень, чтобы минимизировать потенциальные конфликты с другими транзакциями.

Механизм блокирования

- Высокий уровень одновременной обработки данных
  - Блокировки DML на уровне строк
  - Запросы выполняются без блокировок
- Автоматическое управление очередями
- Блокировки удерживаются до окончаний транзакций (до операций фиксации или отката)

Транзакция 1

Транзакция 2

```
SQL> UPDATE employees
  2  SET salary=salary+100
  3  WHERE employee_id=100;
```

```
SQL> UPDATE employees
  2  SET salary=salary*1.1
  3  WHERE employee_id=101;
```


Механизм блокирования

Механизм блокирования спроектирован так, чтобы обеспечивать высокий уровень конкурентной обработки данных внутри базы данных. Транзакции, модифицирующие данные, получают блокировки на уровне строки, а не на уровне таблицы. При модификации объектов (например, перемещении таблицы) накладываются блокировки на уровне объектов, а не всей базы данных или схемы.

Запросы данных не требуют блокировки. Они удачно завершаются, даже если кто-то блокирует данные (всегда выводятся первоначальные, существовавшие до блокирования данные, восстановленные на основе информации отмены).

Когда нескольким транзакциям необходимо заблокировать один и тот же ресурс, первая запрашивающая блокировку транзакция получает ее. Остальные транзакции ждут в очереди, пока первая транзакция завершится. Механизм очередей автоматический.

Все блокировки освобождаются в конце транзакции. Транзакции завершаются, когда происходит фиксация или откат. В случае аварийного завершения транзакции тот же самый фоновый процесс, который автоматически откатывает любые изменения этой транзакции, также освобождает все удерживаемые ее блокировки.

Одновременный доступ

Время:	Транзакция 1	UPDATE hr.employees SET salary=salary+100 WHERE employee_id=100;
	Транзакция 2	UPDATE hr.employees SET salary=salary+100 WHERE employee_id=101;
09:00:00	Транзакция 3	UPDATE hr.employees SET salary=salary+100 WHERE employee_id=102;
	...	...
	Транзакция *x*	UPDATE hr.employees SET salary=salary+100 WHERE employee_id=xxx;

Одновременный доступ

Механизм блокирования по умолчанию работает режиме блокирования на уровне строки. Различные транзакции могут обновлять разные строки внутри одной и той же таблицы, не мешая друг другу.

Хотя по умолчанию действует модель блокирования на уровне строк, поддерживается ручное блокирование таблицы:

```
SQL> LOCK TABLE employees IN EXCLUSIVE MODE;
Table(s) Locked.
```

Режим EXCLUSIVE – наиболее ограничивающий режим блокирования. Есть и другие режимы.

Команды ручного блокирования так же, как и другие запрашиваемые блокировки, ждут в очереди, пока все сеансы, уже удерживающие или запросившие требуемые блокировки раньше, их освободят. Команда LOCK имеет опциональный аргумент NOWAIT. Когда NOWAIT задан, управление немедленно возвращается обратно, если таблица уже заблокирована другим сеансом:

```
LOCK TABLE employees IN SHARE MODE NOWAIT;
LOCK TABLE employees IN SHARE MODE NOWAIT
               *
ERROR at line 1:
ORA-00054: resource busy and acquire with NOWAIT specified
```

Обычно не возникает необходимости в ручном блокировании объектов.

В БД Oracle кроме NOWAIT есть опция WAIT n, где n - максимальное количество секунд ожидания получения блокировки.

Механизм формирования очередей

Механизм очередей следит за:

- **Сеансами, ожидающими блокировок**
- **Требуемым режимом блокировки**
- **Очередностью запроса сеансами блокировок**

12 - 14 SQL

Механизм формирования очередей

Запросы блокировок автоматически ставятся в очередь. Как только удерживающая блокировку транзакция завершается, следующий в очереди сеанс получает блокировку.

Механизм очередей отслеживает порядок, в котором блокировки были запрошены и режимы запрашиваемых блокировок.

Сеансы, уже удерживающие блокировку, могут запросить преобразование (convert) блокировки, и при этом они не ставятся в конец очереди. Например, предположим сеанс удерживает разделяемую блокировку таблицы (SHARE) и запрашивает преобразование этой блокировки в монопольную (EXCLUSIVE). После того, как ни у кого не останется монопольной (EXCLUSIVE) или разделяемой (SHARE) блокировки таблицы, сеанс, удерживающий разделяемую блокировку, получит монопольную блокировку без повторного ожидания в очереди.

Конфликты блокировок

Транзакция 1	Время	Транзакция 2
`UPDATE hr.employees SET salary=salary+100 WHERE employee_id=100;` `1 row updated.`	9:00:00	`UPDATE hr.employees SET salary=salary+100 WHERE employee_id=101;` `1 row updated.`

`UPDATE hr.employees SET COMMISION_PCT=2 WHERE employee_id=101;`
Сеанс ждет в очереди из-за конфликта блокировок.
	9:00:05	`SELECT sum(salary) FROM hr.employees;`
		`SUM(SALARY)`
		`-----------`
		`692634`

Сеанс все еще ждет!	16:30:00	**Много запросов, вставок, изменений и удалений в течение последних 7.5 часов, но ни одной фиксации или отката!**
`1 row updated.` **Сеанс продолжается.**	16:30:01	`commit;`

12 - 15 SQL

Конфликты блокировок

Конфликты блокировок происходят часто. Обычно они со временем разрешаются через механизм очередей. В очень редких случаях конфликт блокировок может потребовать вмешательства администратора. В приведенном на слайде случае транзакция 2 получила блокировку строки в 9:00:00 и не зафиксировала изменение, оставив за собой блокировку. Транзакция 1 попыталась изменить всю таблицу и запросила блокировку всех строк в 9:00:05. В результате транзакция 1 была заблокирована транзакцией 2 до 16:30:01, когда транзакция 2 выполнила фиксацию.

Пользователь, пытающийся выполнить транзакцию 1, конечно будет обращаться к администратору за помощью и АБД должен обнаружить и разрешить конфликт.

Причины конкуренции блокировок

- Незафиксированные изменения
- Длительно выполняемые транзакции
- Неоправданно высокий уровень блокирования

12 - 16 SQL

Возможные причины конкуренции блокировок

Незафиксированные изменения – это наиболее общая причиной конфликта блокировок. Но возможны и некоторые другие причины.

- **Длительно выполняемые транзакции.** Во многих приложениях используется пакетная обработка при внесении *массовых изменений* (*bulk updates*). Выполнение таких пакетных заданий обычно запланировано в часы низкой активности или во время, когда пользователи не работают. Однако в некоторых случаях задания могут не успеть завершиться до возобновления работы пользователей, или же они слишком долго выполняются в период низкой активности. Конфликты блокировок обычно возникают, когда транзакция и пакетная обработка выполняются одновременно.
- **Неоправданно высокий уровень блокирования**. Разработчики могут вручную устанавливать блокировки искусственно завышая их уровень. Например, использовать SELECT FOR UPDATE в целях только выборки данных.

Взаимоблокировки

- **Наличие указывает на ошибки в архитектуре приложения**

Транзакция 1		Транзакция 2
UPDATE employees SET salary = salary * 2 WHERE employee_id = 100;	9:00	UPDATE employees SET manager_id = 102 WHERE employee_id = 200;
UPDATE employees SET salary = salary * 2 WHERE employee_id = 200;	9:15	UPDATE employees SET manager_id = 102 WHERE employee_id = 100;
ERROR: Deadlock detected while waiting for resource	9:16	

- **Устранить возникновение можно только изменением логики работы приложения**

Взаимоблокировки

Взаимоблокировка (*deadlock*) – специальный случай конфликта блокировок. Взаимоблокировка возникают, когда два или несколько сеансов ожидают доступа к данным, блокированным друг у друга. Несколько сеансов могут образовывать “кольца”: 1 ждет 2, 2 ждет 3, 3 ждет 1. Поскольку каждый ждет другого, ни один не может завершить свою транзакцию, чтобы разрешить конфликт.

БД автоматически обнаруживают и разрешают взаимоблокировку путем прерывания последней команды в одной из транзакций. Транзакция выбирается произвольно.

Последняя команда в транзакции, которая была прервана автоматически при обнаружении взаимоблокировки, отменяется и снимаются её блокировки. При этом строки, которые были заблокированы командами (не приведшими к взаимоблокировке) в автоматически прерванной транзакции останутся заблокированными до тех пор, пока прерванная транзакция не даст команды rollback, commit или не разорвет соединение. Если прерванная транзакция даст:

а) ROLLBACK, то она будет отменена полностью

б) COMMIT , то она будет отменена полностью

в) ROLLBACK TO SAVEPOINT к точке сохранения, которую она до этого создала, то отменены будут изменения до точки сохранения и эта транзакция сможет продолжить работать. При откате к точке сохранения строки, которые были заблокированы до точки сохранения останутся заблокированными.

В БД Oracle, в отличии от **PostgreSQL** в автоматически прерванной транзакции возможно давать другие команды, не выполняя отката или фиксации. В **PostgreSQL** до выдачи команд ROLLBACK(COMMIT), ROLLBACK TO SAVEPOINT другие команды будут возвращать ошибку.

SELECT FOR UPDATE

- **Блокирует строки в таблице EMPLOYEES, в которых job_id имеет значение SA_REP.**

```
SELECT employee_id, salary, commission_pct, job_id
FROM employees
WHERE job_id = 'SA_REP'
ORDER BY employee_id
FOR UPDATE;
```

- **Разблокирование происходит после ROLLBACK или COMMIT.**
- **Если SELECT пытается заблокировать строку, заблокированную другой транзакцией, то она ожидает разблокирования строки**

12 - 18 SQL

Предложение FOR UPDATE в инструкции SELECT

При отправке SELECT в базу данных для запроса нескольких записей выбранные строки не блокируются. Блокируются только измененные, но еще не зафиксированные записи. Даже в этом случае другие транзакции могут считывать эти записи в том виде, в котором они существовали до начала изменения. Однако иногда возникает необходимость заблокировать набор записей даже до изменения. Для выполнения такой блокировки используется предложение FOR UPDATE инструкции SELECT.

При запуске SELECT...FOR UPDATE БД блокирует строки по мере их выборки командой SELECT. Другие транзакции будут ждать, пока транзакция не выполнит ROLLBACK или COMMIT, которые снимут блокировку.

FOR UPDATE NOWAIT выдаст ошибку, если столкнется с заблокированной строкой.

В БД Oracle вместо NOWAIT можно указать время ожидания получения блокировки: WAIT *число_секунд*.

Опция FOR UPDATE SKIP LOCKED позволяет команде SELECT пропустить строки, которые заблокированы и не ждать снятия блокировки, однако, эти строки будут отсутствовать в результате выполнения SELECT, даже если они удовлетворяют условию выборки.

Примеры SELECT FOR UPDATE

- **Предложение FOR UPDATE в инструкции SELECT можно использовать для нескольких таблиц.**

```
SELECT e.employee_id, e.salary, e.commission_pct
FROM employees e JOIN departments d
USING (department_id)
WHERE job_id = 'ST_CLERK'
AND location_id = 1500
ORDER BY e.employee_id
FOR UPDATE;
```

- **Блокируются строки в обеих таблицах EMPLOYEES и DEPARTMENTS.**
- **При использовании предложения FOR UPDATE OF *column_name* для определения столбца, который требуется изменить, блокируются только строки из указанной таблицы.**

12 - 19 SQL

Примеры предложения FOR UPDATE

В примере на слайде команда блокирует в таблице EMPLOYEES строки, в которых столбец JOB_ID имеет значение ST_CLERK, а LOCATION_ID равен 1500, а также строки в таблице DEPARTMENTS, в которых столбец LOCATION_ID имеет значение 1500.

FOR UPDATE OF *таблица* (где *таблица* – имя или псевдоним таблицы из фразы FROM) указывает в какой таблице нужно заблокировать строки.

Показанная ниже команда блокирует строки в таблице EMPLOYEES, в таблице DEPARTMENTS строки не блокируются:

```
SELECT e.employee_id, e.salary, e.commission_pct
FROM employees e JOIN departments d
USING (department_id)
WHERE job_id = 'ST_CLERK' AND location_id = 1500
ORDER BY e.employee_id
FOR UPDATE OF e;
```

В БД Oracle после имени таблицы нужно указать имя столбца из неё (FOR UPDATE OF e.*salary*). При этом также будет заблокирована вся строка.

FOR UPDATE и изоляция транзакций

Создадим таблицу и добавим одну строку:

```
CREATE TABLE t (n numeric);
INSERT INTO t VALUES (1); COMMIT;
```

сессия 1	сессия 2
SELECT * FROM t FOR UPDATE;	
	INSERT INTO t VALUES (2); 1 row created. COMMIT;
	SELECT * FROM t WHERE n=2 FOR UPDATE;
UPDATE t SET n=0; сессия заблокирована	
	ROLLBACK;
2 rows updated; ROLLBACK;	

FOR UPDATE и изоляция транзакций

В режиме READ COMMITTED, который используется по умолчанию, SELECT FOR UPDATE блокирует только те строки, которые существовали на момент выполнения команды. Поэтому не стоит ожидать, что команда DML (UPDATE, DELETE, MERGE) , выполненная после SELECT FOR UPDATE и имеющая тот же предикат для фильтрации нужных строк не столкнется с новой строкой, которая была вставлена другой транзакцией.

Если нужно обновить только заблокированные строки, то можно дать команду SET TRANSACTION ISOLATION LEVEL SERIALIZABLE; в первой сессии перед SELECT FOR UPDATE. В этом случае, команда UPDATE обновит 2 строки, третью строку транзакции в первой сессии не увидит.

Таким образом, разбивать действия по блокировке строк и их обновлению на несколько команд в режиме READ COMMITED нежелательно и SELECT FOR UPDATE используется нечасто.

В БД Oracle SELECT FOR UPDATE часто используется в процедурном расширении pl/sql. Например, при создании процедур, триггеров. В процедурном расширении у команды SELECT есть фраза INTO, которая указывает ссылку на объект (курсор), сохраняющий ссылку на выбранные строки. По этой ссылке можно работать с выборкой строк в последующих командах без изменения режима изоляции транзакций.
Процедурное расширение удобно для реализации многошаговой алгоритмической логики, но оно:
•менее производительно, чем использование SQL
•может отсутствовать в БД других производителей
•имеет множество особенностей, побочных эффектов
•оптимизация логики для повышения производительности более трудоемка, чем в SQL
Для реализации нужной логики, обычно, можно подобрать более эффективные методы обработки данных в рамках SQL. Например, использовать временные таблицы вместо курсоров для промежуточного сохранения данных; inline view; команды WITH, MODEL.

Фиксация и откат транзакций

- В JDBC-соединениях и psql autocommit по умолчанию включён
- Завершить транзакцию можно командами COMMIT; или ROLLBACK;
- в БД Oracle выполнение команды DDL или DCL фиксирует открытую транзакцию

12 - 21 SQL

Когда начинается и завершается транзакция?

В JDBC-соединениях autocommit по умолчанию включён и отключается программно “conn.setAutoCommit(false);”

Базы данных оптимизированы для быстрого выполнения команды COMMIT. Команды ROLLBACK используются редко, в случае непредвиденных ситуациях, поэтому БД не оптимизированы для выполнения ROLLBACK и эта команда может выполняться относительно долго.

в **БД Oracle** выполнение команд DDL фиксирует открытую транзакцию.

Завершение транзакци

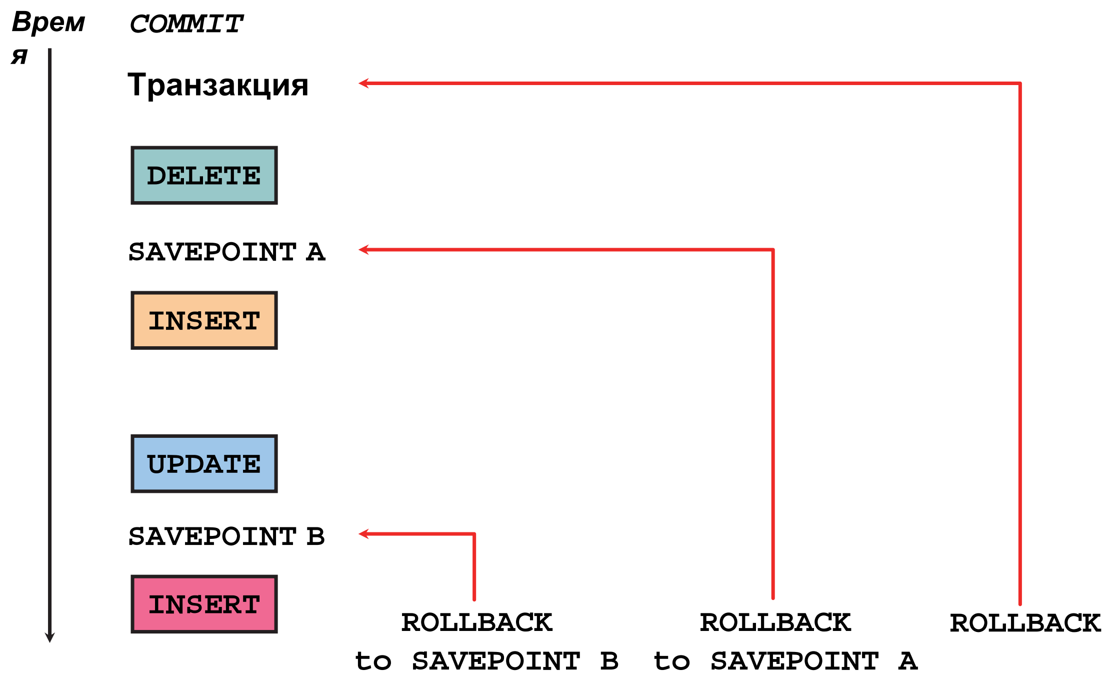
- **используйте команды `COMMIT` или `ROLLBACK`**
- **Позволяют проверить изменения в данных прежде, чем сделать их постоянными.**
- **Логически группируют взаимосвязанные операции.**

`COMMIT` и `ROLLBACK`

Лучше явно использовать команды `COMMIT` или `ROLLBACK` для завершения транзакции, а не полагаться на автоматическую фиксацию.

В JDBC-клиентах используются методы conn.commit(); conn.rollback();

При создании приложений с незнакомыми БД и JDBC-драйверами рекомендуется проверить то, что транзакция будет отменена при возникновении ошибки при выполнении команд. Не все версии JDBC-драйверов или БД могут работать ожидаемым образом. Например, метод conn.rollback(); может выполниться без ошибки, при этом отката транзакции не произойдёт.



Команды явного управления транзакциями

Управление логикой транзакций осуществляется с помощью команд COMMIT, SAVEPOINT, RELEASE SAVEPOINT, ROLLBACK.

Команда	**Описание**
COMMIT	Завершает текущую транзакцию, делая все незафиксированные изменения постоянными.
SAVEPOINT *имя*	Создает точку сохранения в текущей транзакции.
ROLLBACK	Команда ROLLBACK прекращает текущую транзакцию, отменяя все произведенные изменения в данных.
ROLLBACK TO *SAVEPOINT имя*	Команда ROLLBACK TO *SAVEPOINT имя* отменяет все произведенные изменения до точки сохранения.Если опущено предложение TO SAVEPOINT, команда ROLLBACK отменяет все изменения транзакции. Просмотреть список созданных точек сохранения нельзя.
RELEASE SAVEPOINT имя	Удаляет точку сохранения в текущей транзакции

Откат изменений до точки сохранения

- С помощью команды SAVEPOINT создайте точку сохранения в текущей транзакции.
- Выполните откат к точке сохранения, используя команду ROLLBACK TO SAVEPOINT.

```
UPDATE...
SAVEPOINT update_done;
Savepoint created.
INSERT...
ROLLBACK TO update_done;
Rollback complete.
```

12 - 24 SQL

Откат изменений до точки сохранения

С помощью команды SAVEPOINT можно создать точку сохранения в текущей транзакции, которая делит транзакцию на более мелкие части. Далее по команде ROLLBACK TO SAVEPOINT можно отменить незаконченные изменения, сделанные после точки сохранения.

Если создается вторая точка сохранения с таким же именем, предыдущая точка уничтожается.

В java создание точки сохранения, откат до точки, открытие и фиксация транзакций сохранения осуществляется программно - методами.

Состояние данных до выполнения команды COMMIT или ROLLBACK

- Предыдущее состояние данных может быть восстановлено.
- Текущий пользователь может просмотреть результаты своих операций DML с помощью команды SELECT.
- Другие пользователи *не могут* видеть результаты команд DML, выполняемых текущим пользователем.
- Измененяемые строки *блокируются*, и другие пользователи не могут обновлять их содержимое.

12 - 25 SQL

Фиксация изменений

Каждое изменение данных, выполненное в ходе транзакции, является временным до тех пор, пока транзакция не будет зафиксирована.

Состояние данных до команды `COMMIT` или `ROLLBACK`:

- Операции манипулирования данными изменяют только буфер базы данных; следовательно, предыдущее состояние данных может быть восстановлено.
- Текущий пользователь может проверить результаты своих операций манипулирования данными путем выполнения запросов к таблицам.
- Другие пользователи не могут видеть результаты операций манипулирования данными, выполняемых текущим пользователем.
- Строки, с которыми в данный момент проводятся операции, блокируются; другие пользователи изменять их не могут.

Практическое задание 12

- Вставка строк в таблицы
- Обновление и удаление строк в таблице
- Управление транзакциями

12 - 26 SQL

Практическое задание 12

На этом практическом занятии вы будете добавлять строки в таблицу, обновлять данные, удалять данные из таблицы и осуществлять управление своими транзакциями.

13

Коррелированные подзапросы

Рассматриваемые вопросы

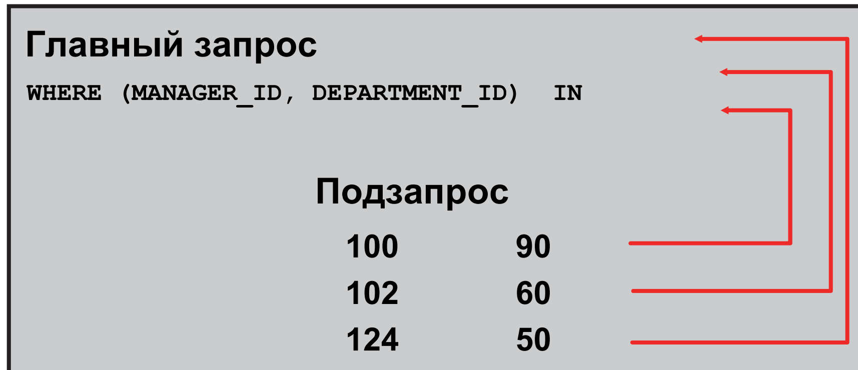
- Создание многостолбцовых подзапросов
- Использование скалярных подзапросов в SQL
- Типы проблем, которые могут быть решены с помощью связанных подзапросов
- Изменение и удаление строк с использованием связанных подзапросов
- Изменение и удаление строк с использованием связанных подзапросов
- Использование операторов `EXISTS` и `NOT EXISTS`

13 - 2 SQL

Цели урока

На этом уроке вы научитесь создавать многостолбцовые подзапросы (multiple-column subqueries) и подзапросы в предложении `FROM` команды `SELECT`. Вы также научитесь решать проблемы с помощью скалярных и связанных (correlated) подзапросов, а также предложения `WITH`.

Многостолбцовые подзапросы



Каждая строка главного запроса сравнивается с значениями из многострочного и многостолбцового подзапроса

Многостолбцовые подзапросы

До сих пор рассматривались только *однострочные* (*single-row*) и *многострочные* (multiple-row) *позапросы*, в которых внутренняя команда SELECT возвращает только один столбец, значение которого используется в выражении родительской команды выбора. Если требуется сравнение значений двух или более столбцов, необходимо сложное предложение WHERE с логическими операторами. Многостолбцовые подзапросы позволяют объединять дублируемые условия WHERE в единое предложение WHERE.

Синтаксис

```
SELECT    столбец, столбец, ...
 FROM     таблица
 WHERE    (столбец, столбец, ...) IN
            (SELECT столбец, столбец, ...
                            FROM    таблица
                            WHERE   условие);
```

В примере на слайде показывается, что значения столбцов MANAGER_ID и DEPARTMENT_ID главного запроса сравниваются с значениями столбцов MANAGER_ID и DEPARTMENT_ID, возвращаемых подзапросом. В примере выполняется сравнение более одного столбца. Поэтому приведенный подзапрос – это *многостолбцовый подзапрос* (*multiple-column subquery*).

Сравнения столбцов

Сравнения столбцов в многостолбцовом подзапросе могут быть:

- **парными;**
- **непарными.**

13 - 4 SQL

Сравнения столбцов

Сравнения столбцов в многостолбцовом подзапросе могут быть *парными* (*pairwise*) и *непарными* (*nonpairwise comparisons*).

В примере на слайде следующей страницы в предложении WHERE производится парное сравнение. Каждая строка-кандидат в команде SELECT должна *одновременно* иметь такой же номер менеджера (MANAGER_ID) и такой же номер отдела (DEPARTMENT_ID), как у сотрудников с номерами (EMPLOYEE_ID) 199 или 174.

Вместе с многостолбцовым подзапросом может использоваться непарное сравнение. Каждый столбец из предложения WHERE родительской команды SELECT отдельно сравнивается с несколькими значениями, возвращаемыми внутренним запросом. Но для каждой выводимой строки главного запроса должны выполняться все условия в целом. Далее в уроке приводится пример, который демонстрирует непарное сравнение.

Подзапрос с парным сравнением

Вывод сведений о сотрудниках, которые одновременно подчиняются такому же менеджеру и работают в одном отделе, что и сотрудники с номерами (EMPLOYEE_ID) 199 или 174.

```
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 (199,174))
AND     employee_id NOT IN (199,174);
```


Подзапрос с парным сравнением

Подзапрос в примере на слайде является *многостолбцовым*, т.к. возвращает значения нескольких столбцов. Он сравнивает значения всех строк из таблицы EMPLOYEES в столбцах MANAGER_ID и DEPARTMENT_ID со значениями этих столбцов в строках сведений о сотрудниках с номерами (EMPLOYEE_ID) 199 или 174.

Первым выполняется подзапрос, который выдает пары значений MANAGER_ID и DEPARTMENT_ID для сотрудников с номерами 199 или 174. Эти значения сравниваются с значениями столбцов MANAGER_ID и DEPARTMENT_ID каждой строки таблицы EMPLOYEES. При совпадении значений выводится строка. В выходных результатах нет строк о сотрудниках с номерами 199 и 174.

Подзапрос с непарным сравнением

Вывод сведений о сотрудниках, которые подчиняются такому же менеджеру, что и сотрудники с номерами (EMPLOYEE_ID) 174 или 199, *и* работают в одном отделе, что и сотрудники с номерами 174 или 199.

```
SELECT   employee_id, manager_id, department_id
FROM     employees
WHERE    manager_id IN
                   (SELECT   manager_id
                    FROM     employees
                    WHERE    employee_id IN (174,199))
AND      department_id IN
                   (SELECT   department_id
                    FROM     employees
                    WHERE    employee_id IN (174,199))

AND     employee_id NOT IN(174,199);
```

13 - 6   SQL

Подзапрос с непарным сравнением

В примере на слайде производится непарное сравнение столбцов. Выводятся номера сотрудника, менеджера и отдела для тех сотрудников, менеджер которых – любой из менеджером сотрудников с номерами 174 и 199, и номер отдела совпадает с любым из отделов, в которых работают сотрудники с номерами 174 и 199.

Первым выполняется подзапрос, который выдает значения MANAGER_ID для сотрудников с номерами 174 или 199. Подобным образом второй запрос выбирает значения DEPARTMENT_ID для сотрудников с EMPLOYEE_ID 174 или 199. Полученные значения столбцов MANAGER_ID и DEPARTMENT_ID сравниваются с значениями таких же столбцов таблицы EMPLOYEES. Если столбец MANAGER_ID строки таблицы EMPLOYEES совпадает с любым из значений столбца MANAGER_ID внутреннего подзапроса и если столбец DEPARTMENT_ID строки таблицы EMPLOYEES совпадает с любым из значений столбца DEPARTMENT_ID второго подзапроса, тогда строка выводится.

Скалярные подзапросы в SQL

- **Скалярный подзапрос – это подзапрос, возвращающий ровно одно значение столбца из одной строки**
- **Скалярные подзапросы могут быть использованы:**
  - **условии и части выражения DECODE и CASE;**
  - **во всех предложениях команды SELECT кроме GROUP BY.**

13 - 7 SQL

Скалярные подзапросы в SQL

На подзапрос, который возвращает ровно одно значение столбца из одной строки, ссылаются также как на скалярный подзапрос. Многостолбцовые запросы, созданные для сравнения двух или более столбцов, использующие сложное предложение WHERE и логические операторы, не являются скалярными подзапросами.

Скалярный подзапрос возвращает значение элемента из списка выбора подзапроса. Если подзапрос возвращает 0 строк, выражение, которое образует скалярный запрос, имеет неопределенное значение (NULL). Если подзапрос возвращает более одной строки, БД выводит сообщение об ошибке. Они также могут использоваться:

- в условии и части выражения CASE;
- во всех предложениях команды SELECT кроме GROUP BY;
- в предложениях SET и WHERE команды UPDATE.

Однако скалярные подзапросы не применяются:

- как значения по умолчанию;
- в предложении RETURNING команд DML;
- для создания индекса, основанного на функции;
- в предложении GROUP BY, ограничении CHECK и условии WHEN;
- в предложении CONNECT BY;

Примеры скалярных подзапросов

- **Скалярный подзапрос в выражениях CASE**

```
SELECT employee_id, last_name,
       (CASE
        WHEN department_id =          20
                  (SELECT department_id
                   FROM departments
                   WHERE location_id = 1800)
         THEN 'Canada' ELSE 'USA' END) location
FROM   employees;
```

- **Скалярный подзапрос в предложении ORDER BY**

```
SELECT    employee_id, last_name
FROM      employees e
ORDER BY (SELECT department_name
           FROM departments d
            WHERE e.department_id = d.department_id);
```


Примеры скалярных подзапросов

В первом примере на слайде приведен скалярный подзапрос в выражении CASE. Внутренний подзапрос возвращает значение 20, которое представляет собой номер отдела, расположенного в местоположении с номером 1800. Выражение CASE во внешнем запросе использует результат внутреннего запроса для вывода номеров и фамилий сотрудников, а также страны (США или Канада) размещения отдела в зависимости от номера (20 или не 20), который получает внешний запрос.

Во втором примере на слайде приведен скалярный подзапрос, используемый в предложении ORDER BY. Выходные данные упорядочиваются на основе наименования отдела DEPARTMENT_NAME путем сопоставления столбцов номеров отделов (DEPARTMENT_ID) из таблиц EMPLOYEES и DEPARTMENTS. Это сравнение выполняется с помощью скалярного подзапроса в предложении ORDER BY. В этом примере используется ***связанный*** *он же* ***коррелированный*** *подзапрос* (*correlated subquery*). В таких подзапросах столбец связывается с столбцом таблицы, на которую ссылается родительская команда.

Такие подзапросы могут выполняться неэффективно с использованием циклов переборки результата подзапроса в процессе выполнения команды. Если можно переписать такой запрос с использованием соединения таблиц, то есть фактически перенести подзапрос во фразу WHERE или JOIN, то БД сможет использовать более эффективные методы соединения таблиц (hash join). На большом количестве строк это может быть более эффективно.

Связанные подзапросы

Связанные подзапросы используются при построчной обработке. Каждый подзапрос выполняется один раз для каждой строки внешнего запроса.

13 - 9 SQL

Связанные подзапросы

БД выполняет *связанный подзапрос* (*correlated subquery*), когда подзапрос связывается с столбцом таблицы, на который ссылается родительская команда. Связанный подзапрос выполняется один раз для каждой строки, обрабатываемой родительской командой. Такими командами могут быть `SELECT`, `UPDATE` и `DELETE`.

Вложенные подзапросы или связанные подзапросы

При использовании обычного *вложенного подзапроса* (*nested subquery*) внутренняя команда `SELECT` выполняется в первую очередь и только один раз, возвращая значения, которые используются главным запросом. С другой стороны, связанный подзапрос выполняется один раз для каждой строки-кандидата внешнего запроса. Другими словами, внутренний запрос управляется внешним запросом.

Выполнение вложенного подзапроса

- Первым выполняется внутренний запрос и находит значение.
- Внешний запрос выполняется один раз, используя значение внутреннего запроса.

Выполнение связанного подзапроса

- Извлечение строки-кандидата (выборку производит внешний запрос).
- Выполнение внутреннего запроса, используя значение строки-кандидата.
- Использование значений, полученных из внутреннего запроса, для принятия или отбраковки строки-кандидата.
- Повторять выполнение, пока не останется строк-кандидатов.

Связанные подзапросы

Подзапрос ссылается на столбец таблицы, из которой выбирает родительский запрос.

```
SELECT столбец1, столбец2, ...
FROM   таблица1 outer
WHERE  столбец1 оператор
                      (SELECT  столбец1, столбец2
                       FROM    таблица2
                       WHERE   выражение1 =
                                  outer.выражение2);
```

13 - 10 SQL

Связанные подзапросы

Связанный подзапрос выполняет чтение строки и сравнение значений с соответствующими данными. Он используется, когда подзапрос должен возвращать различный результат или набор результатов на основе каждой строки-кандидата главного запроса. Другими словами, связанный подзапрос используется для ответа на составной вопрос, и это ответ зависит от значения в каждой строке, обрабатываемой родительской командой.

БД выполняет связанный подзапрос, когда подзапрос ссылается на столбец таблицы, из которой выбирает родительский запрос.

Примечание

В связанном подзапросе можно использовать операторы ANY и ALL.

Связанные подзапросы во фразе WHERE, обычно, выполняются эффективно, в отличите от связанных подзапросов, указанных на уровне SELECT или FROM.

Использование связанных подзапросов

Поиск всех сотрудников, зарабатывающих больше среднего оклада по отделу, в котором они работают.

```
SELECT last_name, salary, department_id
FROM   employees outer
WHERE  salary >
            (SELECT AVG(salary)
             FROM   employees
             WHERE  department_id =
             outer.department_id) ;
```

Каждый раз, когда обрабатывается строка внешнего запроса , выполняется внутренний запрос.

Использование связанных подзапросов

В примере на слайде определяются сотрудники, зарабатывающие больше среднего оклада по отделу, в котором они работают. Связанный запрос выполняет подсчет среднего оклада для каждого конкретного отдела.

Так как *внешний* (*outer*) и *внутренний* (*inner*) запросы используют таблицу `EMPLOYEES` в предложении `FROM`, во внешнем запросе для этой таблицы задается псевдоним *`outer`*. Это делается не только для обеспечения читабельности всей команды `SELECT`, но и потому что без этого внутренняя команда не сможет отличить столбцы внутренней и внешней таблиц.

Использование связанных подзапросов

Вывод сведений о сотрудниках, которые меняли занимаемую должность хотя бы дважды.

```
SELECT e.employee_id, last_name,e.job_id
FROM   employees e
WHERE  2 <= (SELECT COUNT(*)
             FROM   job_history
             WHERE  employee_id = e.employee_id);
```

EMPLOYEE_ID	LAST_NAME	JOB_ID
101	Kochhar	AD_VP
176	Taylor	SA_REP
200	Whalen	AD_ASST

Использование связанных подзапросов (продолжение)

В примере на слайде выводятся сведений о сотрудниках, которые меняли занимаемую должность хотя бы дважды. БД обрабатывает связанный подзапрос следующим образом.

1. Выбор строки из таблицы внешнего запроса. Она становится текущей строкой-кандидатом.
2. Сохранение значения столбца строки-кандидата, на который ссылается подзапрос (в примере на слайде в подзапросе используется ссылка `E.EMPLOYEE_ID`).
3. Выполнение подзапроса при условии, использующем ссылку на значение столбца строки-кандидата внешнего запроса (в примере вычисляется групповая функция `COUNT(*)` на основе значения столбца `E.EMPLOYEE_ID`, полученного на шаге 2).
4. Обработка предложения `WHERE` внешнего запроса на основе результатов подзапроса, полученных на шаге 3. В итоге определяется, будет ли строка-кандидат выведена внешним запросом. В примере в предложении `WHERE` внешнего запроса количество переходов на другую работу, вычисляемое подзапросом, сравнивается с 2. Если условие выполняется, тогда запись о сотруднике выводится.
5. Процедура повторяется для следующих строк-кандидатов, пока все строки не будут выбраны из таблицы.

Связь (*correlation*) устанавливается с помощью использования элемента внешнего запроса в подзапросе. В примере связь определяется как `EMPLOYEE_ID = E.EMPLOYEE_ID`. Этот оператор сравнивает столбец `EMPLOYEE_ID` из таблицы подзапроса с столбцом `EMPLOYEE_ID` внешнего запроса.

Использование оператора EXISTS

- Оператор EXISTS проверяет существование строк в наборе результатов подзапроса.
- Если значение строки подзапроса найдено:
  – поиск во внутреннем запросе прерывается;
  – условие отмечается как верное (TRUE).
- Если значение строки подзапроса не найдено:
  – условие отмечается как ложное (FALSE);
  – поиск во внутреннем запросе продолжается.

13 - 13 SQL

Оператор EXISTS

Во вложенных командах SELECT все логические операторы проверяются. Кроме того, можно использовать оператор EXISTS. Этот оператор часто применяется вместе с связанными подзапросами для проверки, существует ли значение, выбранное внешним запросом в наборе результатов внутреннего запроса. Если подзапрос находит хотя бы одну строку, оператор EXISTS возвращает значение TRUE. Если значение не найдено, возвращается значение FALSE. Соответственно NOT EXISTS проверяет условие, что значение внешнего запроса не является частью набора результатов внутреннего запроса.

Пример: поиск служащих, которым подчиняется хотя бы один служащий

```
SELECT employee_id, last_name, job_id, department_id
FROM   employees outer
WHERE  EXISTS ( SELECT 'X'
                FROM   employees
                WHERE  manager_id =
                       outer.employee_id);
```

EMPLOYEE_ID	LAST_NAME	JOB_ID	DEPARTMENT_ID
100	King	AD_PRES	90
101	Kochhar	AD_VP	90
102	De Haan	AD_VP	90
103	Hunold	IT_PROG	60
108	Greenberg	FI_MGR	100
114	Raphaely	PU_MAN	30
120	Weiss	ST_MAN	50
121	Fripp	ST_MAN	50
122	Kaufling	ST_MAN	50
123	Vollman	ST_MAN	50
124	Mourgos	ST_MAN	50
145	Russell	SA_MAN	80
146	Partners	SA_MAN	80
147	Errazuriz	SA_MAN	80
148	Cambrault	SA_MAN	80
149	Zlotkey	SA_MAN	80
201	Hartstein	MK_MAN	20
205	Higgins	AC_MGR	110

18 rows selected.

13 - 14 SQL

Пример: поиск служащих, которым подчиняется хотя бы один служащий

Оператор EXISTS обеспечивает, что поиск во внутреннем запросе не будет продолжаться, как только будет найдена хотя бы одна строка, удовлетворяющая условию совпадения номеров менеджера и служащего:

```
WHERE manager_id = outer.employee_id.
```

Заметим, что внутренний запрос не должен возвращать конкретное значение. Поэтому может быть выбрана константа в команде SELECT.

Пример: поиск отделов, в которых нет служащих

```
SELECT department_id
FROM departments d
WHERE NOT EXISTS (SELECT 'X'
                  FROM   employees
                  WHERE  department_id = d.department_id);
```

13 - 15 SQL

Использование оператора NOT EXISTS

Альтернативное решение

Конструкция NOT IN может быть использована в качестве альтернативы для оператора NOT EXISTS, как показано в следующем примере.

```
SELECT department_id
FROM   departments
WHERE  department_id NOT IN (SELECT department_id
                             FROM   employees);
No rows selected.
```

Однако NOT IN вырабатывает FALSE, если любой элемент множества имеет неопределенное значение (NULL). Поэтому запрос не будет возвращать строк, даже если существуют строки в таблице отделов, удовлетворяющие условию в предложении WHERE. Запросы с EXISTS/NOT EXISTS выполняются эффективнее эквивалентных запросов с использованием операторов над множествами (MINUS/EXCEPT). Например эквивалентный запрос тому, что представлен на слайде на больших объемах менее эффективен

```
SELECT department_id FROM departments
MINUS
SELECT department_id FROM employees;
```

Более того, в запрос на слайде можно добавить столбцы, например, department_name, что не сделаешь в запросе с MINUS.

Связанная команда UPDATE

Использование связанного подзапроса для изменения строк таблицы на основе строк другой таблицы.

```
UPDATE таблица1 псевдоним1
SET    столбец = (SELECT выражение
                  FROM   таблица2 псевдоним2
                  WHERE  псевдоним1.столбец =
                         псевдоним2.столбец);
```


Связанная команда UPDATE

В команде UPDATE можно использовать *связанный подзапрос для изменения строк* (*correlated subquery to update rows*) таблицы на основе строк другой таблицы.

Использование связанной команды UPDATE

- **Сделаем таблицу EMPL6 ненормализованной, добавив столбец для хранения наименования отдела.**
- **Заполним таблицу, используя связанную команду изменения.**

```
ALTER TABLE empl6
ADD(department_name VARCHAR2(25));
```

```
UPDATE empl6 e
SET    department_name =
              (SELECT department_name
               FROM   departments d
               WHERE  e.department_id = d.department_id);
```

13 - 17 SQL

Связанная команда UPDATE (продолжение)

В примере на слайде таблица EMPL6 становится ненормализованной после добавления столбца для хранения наименования отдела. Затем в таблицу вносятся наименования отделов с помощью связанной команды изменения.

Приведем другой пример *связанной команды изменения* (*correlated update*).

Постановка задачи

Используйте связанный подзапрос для изменения строк таблицы EMPL6 на основе строк таблицы REWARDS:

```
UPDATE empl6
SET    salary = (SELECT empl6.salary + rewards.pay_raise
                 FROM   rewards
                 WHERE  employee_id  =
                        empl6.employee_id
                 AND  payraise_date =
                      (SELECT MAX(payraise_date)
                       FROM   rewards
                       WHERE  employee_id = empl6.employee_id))
WHERE empl6.employee_id
IN     (SELECT employee_id FROM rewards);
```

Связанная команда UPDATE (продолжение)

В этом примере используется таблица REWARDS, которая содержит столбцы EMPLOYEE_ID, PAY_RAISE и PAYRAISE_DATE. Каждый раз, когда сотрудник получает повышение, в таблицу REWARDS вставляется запись с номером сотрудника, размером повышения и датой получения повышения. В таблице REWARDS может быть более одной записи, относящейся к сотруднику. Столбец PAYRAISE _DATE используется для определения самого последнего по времени повышения, полученного сотрудником.

В примере столбец SALARY таблицы EMPL6 изменяется так, чтобы отразить самое последнее повышение, полученное сотрудником. Это выполняется путем добавления к текущему окладу сотрудника соответствующего повышения оклада из таблицы REWARDS.

Связанная команда DELETE

Использование связанного подзапроса для удаления строк таблицы на основе строк другой таблицы.

```
DELETE FROM таблица1 псевдоним1
WHERE  столбец оператор
             (SELECT выражение
                 FROM   таблица2 псевдоним2
                 WHERE  псевдоним1.столбец =
                        псевдоним2.столбец);
```


Связанная команда DELETE

В команде DELETE можно использовать *связанный подзапрос для удаления* (*correlated subquery to delete*) только таких строк таблицы, которые также существуют в другой таблице. Если принято решение, хранить в таблице JOB_HISTORY только последние четыре записи о занимаемых должностях сотрудника, тогда при появлении пятой самая старая строка вычеркивается из таблицы JOB_HISTORY. Для этого просматривается таблица JOB_HISTORY и находится MIN(START_DATE)для сотрудника. В следующем примере показывается, как такая операция может быть выполнена с использованием связанной команды DELETE:

```
DELETE FROM emp_history JH
WHERE employee_id =
      (SELECT employee_id
       FROM employees E
       WHERE JH.employee_id = E.employee_id
       AND START_DATE =
             (SELECT MIN(start_date)
              FROM job_history JH
              WHERE JH.employee_id = E.employee_id)
              AND 5 >  (SELECT COUNT(*)
                        FROM job_history JH
                        WHERE JH.employee_id = E.employee_id
                        GROUP BY EMPLOYEE_ID
                        HAVING COUNT(*) >= 4));
```

Использование связанной команды DELETE

Использование связанного подзапроса для удаления только таких строк таблицы EMPL6, которые также присутствуют в таблице EMP_HISTORY.

```
DELETE FROM empl6 E
WHERE employee_id =
           (SELECT employee_id
            FROM   emp_history
            WHERE  employee_id = E.employee_id);
```


Использование связанной команды DELETE

Пример

Две таблицы используются в этом примере:

- таблица EMPL6, в которой находятся подробные сведения о всех работающих в компании сотрудниках;
- таблица EMP_HISTORY, в которой находятся подробные сведения о всех ранее работавших в компании сотрудниках.

Таблица EMP_HISTORY содержит информацию, имеющее отношение к ранее работавшим сотрудникам. Поэтому будет неверно, если в обоих таблицах EMPL6 и EMP_HISTORY будут существовать записи об одних и тех же наемных работниках. Можно удалить такие ошибочные записи с помощью запроса, показанного на слайде.

Корреляция Inline Views

- Позволяют из Inline Views обращаться к внешнему запросу
- Требуется использовать ключевое слово LATERAL

```
SELECT * FROM employees e,
  LATERAL(SELECT * FROM departments d
       WHERE e.department_id = d.department_id) f;
```

13 - 21 SQL

Латеральные подзапросы

Могут использоваться для Top-N анализа, т.е. когда в Inline Vew ограничивается колличество строк Также могут реализовывать логику "for each" средствами SQL.

Итоги

- Связанный подзапрос полезен, когда подзапрос должен возвращать различный результат для каждой строки-кандидата.
- Оператор `EXISTS` –это логический оператор, который проверяет наличие значения.
- Связанные подзапросы могут использоваться в командах `SELECT`, `UPDATE` и `DELETE`.
- Предложение `WITH` позволяет использовать тот же самый блок запроса в нескольких местах команды `SELECT`.

13 - 22 SQL

Итоги

БД выполняет связанный подзапрос, когда подзапрос ссылается на столбец таблицы из родительской команды. Связанный подзапрос выполняется один раз для каждой строки, обрабатываемой родительской командой, которая может быть `SELECT`, `UPDATE` или `DELETE`. Используя предложение `WITH`, можно повторно применять тот же самый запрос, когда стоимость повторной обработки блока запроса высокая и он появляется более одного раза внутри сложного запроса.

Практическое задание 13

- Создание многостолбцовых подзапросов
- Написание связанных подзапросов
- Использование оператора EXISTS
- Использование скалярных подзапросов

13 - 23 SQL

Практическое задание 13

В ходе выполнения заданий практического занятия вы будете создавать многостолбцовые, связанные и скалярные подзапросы.

14

Создание объектов

Представления

Таблица `EMPLOYEES`

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALA
100	Steven	King	SKING	515.123.4567	17-JUN-87	AD_PRES	240
101	Neena	Kochhar	NKOCHHAR	515.123.4568	21-SEP-89	AD_VP	170
102	Lex	De Haan	LDEHAAN	515.123.4569	13-JAN-93	AD_VP	170
103	Alexander	Hunold	AHUNOLD	590.423.4567	03-JAN-90	IT_PROG	90
104	Bruce	Ernst	BERNST	590.423.4568	21-MAY-91	IT_PROG	60
107	Diana	Lorentz	DLORENTZ	590.423.5567	07-FEB-99	IT_PROG	42
124	Kevin	Mourgos	KMOURGOS	650.123.5234	16-NOV-99	ST_MAN	58
141	Trenna	Rajs	TRAJS	650.121.8009	17-OCT-95	ST_CLERK	35
142	Curtis	Davies	CDAVIES	650.121.2994	29-JAN-97	ST_CLERK	31
143	Randall	Matos	RMATOS	650.121.2874	15-MAR-98	ST_CLERK	26
					JUL-98	ST_CLERK	25
					JAN-00	SA_MAN	105
					MAY-96	SA_REP	110
					MAR-98	SA_REP	86
178	Kimberely	Grant	KGRANT	011.44.1644.429263	24-MAY-99	SA_REP	70
200	Jennifer	Whalen	JWHALEN	515.123.4444	17-SEP-87	AD_ASST	44
201	Michael	Hartstein	MHARTSTE	515.123.5555	17-FEB-96	MK_MAN	130
202	Pat	Fay	PFAY	603.123.6666	17-AUG-97	MK_REP	60
205	Shelley	Higgins	SHIGGINS	515.123.8080	07-JUN-94	AC_MGR	120
206	William	Gietz	WGIETZ	515.123.8181	07-JUN-94	AC_ACCOUNT	83

EMPLOYEE_ID	LAST_NAME	SALARY
149	Zlotkey	10500
174	Abel	11000
176	Taylor	8600

20 rows selected.

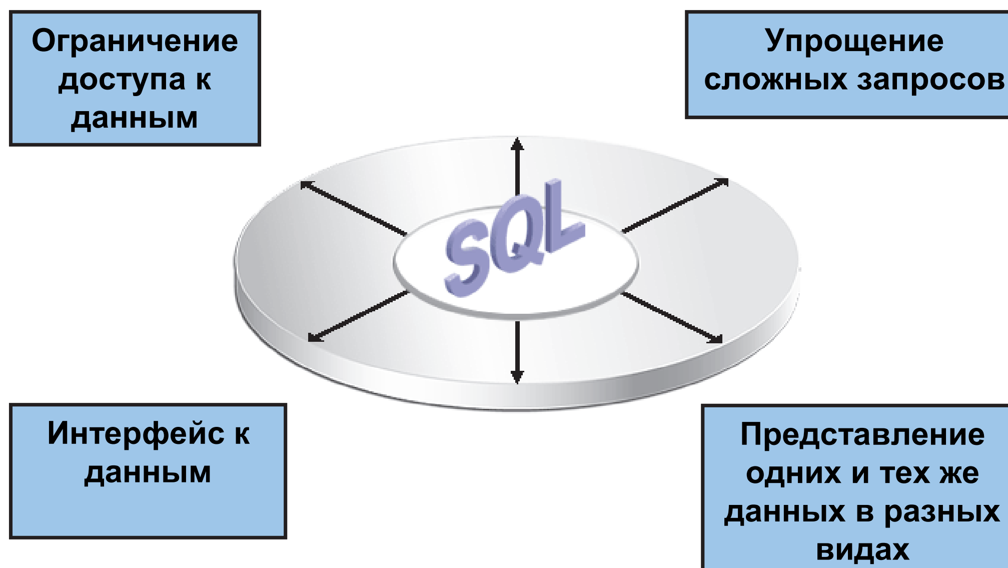
Представления

Представление (view) – это логическая таблица, основанная на реальной таблице (таблицах) и/или другом представлении (представлениях). Представление не содержит собственных данных, но выполняет роль “окна”, через которое можно просматривать и изменять таблицы. Таблицы, на которых основано представление, называются базовыми.

Представления позволяют скрыть внутреннее устройство таблиц, которые могут меняться по мере развития приложения за интерфейсами.

Также представление может скрыть часть строк базовых таблиц. Таким образом, применяться для разграничения доступа к строкам.

Представления можно использовать практически везде, где можно использовать обычные таблицы. Представления могут создаваться на базе других представлений.



Преимущества представлений

- Ограничивают доступ к базе данных, т.к. представление может показывать пользователю только часть столбцов и/или строк таблиц.
- Позволяют пользователям с помощью простых запросов получать результаты сложных запросов. Например, пользователи могут запрашивать информацию сразу из нескольких таблицах, не зная условия соединения таблиц.
- Могут служить интерфейсами - позволяют скрыть внутреннее устройство таблиц, которые могут меняться по мере развития приложения.
- Можно давать привилегии на представление и не давать привилегии на базовые таблицы (команда `GRANT SELECT, INSERT, UPDATE, DELETE ON viewname TO username;`)

Создание представления

- Создание представления

```
CREATE OR REPLACE VIEW empvu80
 AS SELECT  employee_id eid, last_name, salary sal
    FROM    employees
    WHERE   department_id = 80
WITH CHECK OPTION;
View created.
```

- Удаление представления

```
DROP VIEW empvu80;
```

14 - 4 SQL

Создание представления

В примере на слайде создается представление, включающее номер служащего, фамилию и должность для всех служащих отдела 80.

Именами столбцов можно управлять путем включения псевдонимов столбцов в запрос.

Запрос, определяющий представление, может содержать сложную команду `SELECT`, включающую соединения, объединения и т.п.

`OR REPLACE` позволяет пересоздать представление, если оно есть с таким именем.

`WITH CHECK OPTION` управляет поведением автоматически изменяемых представлений. Если оно присутствует, при выполнении операций INSERT и UPDATE с этим представлением будет проверяться, удовлетворяют ли новые строки условию, определяющему представление (то есть, проверяется, будут ли новые строки видны через это представление). Если они не удовлетворяют условию, операция не будет выполнена. Если указание CHECK OPTION отсутствует, команды INSERT и UPDATE смогут создавать в этом представлении строки, которые не будут видны в нём.

Удаление представлений не влияет на базовые таблицы.

Правила обновления строк таблиц через представления

- Операции INSERT UPDATE DELETE MERGE можн выполнять если представление не содержит:
  - агрегатные и аналитические функции
  - GROUP BY, HAVING, WITH, DISTINCT, LIMIT, OFFSET
  - UNION, INTERSECT, EXCEPT

14 - 5 SQL

Правила обновления строк таблиц через представления

Операции DML могут выполняться над представлением если оно не содержит ничего из следующего:

•Список выборки в запросе не должен содержать агрегатные и аналитические функции

•Предложение GROUP BY, HAVING, WITH, DISTINCT, LIMIT, OFFSET на верхнем уровне запроса

•операции с множествами (UNION, INTERSECT и EXCEPT) на верхнем уровне запроса

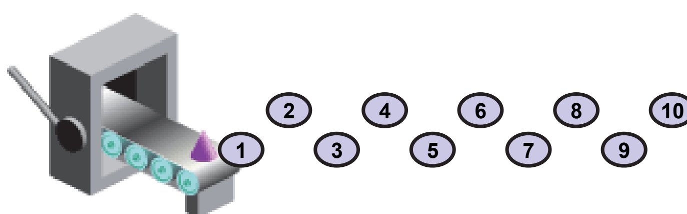
DML операции будут менять строки в базовых таблицах. Представление может содержать как изменяемые, так и не изменяемые столбцы. Столбец будет изменяемым, если это простая ссылка на изменяемый столбец нижележащего базовой таблицы. Если это выражение (например, SAL*12) или функция, то этот столбец будет доступен только для чтения.

Создать эффект изменяемого представления для представлений, не удовлетворяющие этим условиям, можно создав триггеры INSTEAD OF, которые будут выполнять DML операции.

Последовательности

Последовательность:

- **Автоматически генерирует уникальные номера.**
- **Является совместно используемым объектом.**
- **Обычно используется для генерации значений главного ключа.**
- **Заменяет код в прикладной программе.**



14 - 6 SQL

Последовательности

Последовательность (sequence) – это объект базы данных, который создается пользователем и может использоваться несколькими пользователями для генерации целых чисел.

Последовательность может быть создана для генерации неповторяющихся значений или для повторного использования в цикле тех же самых чисел.

Этот объект базы данных может сэкономить время, т.к. замещает код, который потребовался бы для обеспечения уникальности чисел в прикладной программе.

Числа последовательности хранятся и генерируются независимо от таблиц. Последовательности могут быть связаны с таблицами в целях автоматического удаления при удалении таблицы.

Создание последовательности

```
CREATE SEQUENCE имя
       [INCREMENT BY n]
       [MAXVALUE n]
       [MINVALUE n]
       [START WITH n]
       [CYCLE]
       [CACHE n];
```

В postgres:

```
SELECT seq.CURRVAL FROM dual;
INSERT INTO tab VALUES (seq.NEXTVAL);
```

```
SELECT CURRVAL('seq');
SELECT NEXTVAL ('seq');
```


Создание последовательности

`INCREMENT BY n` интервал между двумя последовательными числами, где *n* – целое число. По умолчанию 1.

`START WITH n` первое генерируемое число в последовательности. По умолчанию 1.

`MAXVALUE n` максимальное значение.

`MINVALUE n` минимальное значение

`CYCLE` генерация чисел после достижения максимального или минимального значения идёт по кругу

`CACHE n` количество чисел, которые генерируются и хранятся в памяти для ускорения доступа

Пропуски в последовательностях

Хотя генераторы чисел генерируют числа без пропусков, это происходит независимо от фиксации и отката транзакций. Следовательно, в случае отката команды, содержащей ссылку на последовательность, данное число теряется.

Еще одна возможная причина пропуска в последовательности – это системные сбои. Если используется CACHE, в случае системного сбоя они теряются.

Команда удаления последовательности DROP SEQUENE имя;

Перезапуск последовательности **в БД postgres**: ALTER SEQUENCE имя RESTART WITH n;

В БД Oracle нет аналогичной команды, но можно поменять INCREMENT BY так, чтобы следующее значение стало желаемым, выполнить NEXTVAL и вернуть INCREMENT BY обратно. При этом надо предотвратить использование последовательности другими сессиями. Удаление и пересоздание последовательности не является хорошим решением, так как от последовательности могут зависеть другие объекты.

Автоматическая генерация значений столбца

- Используются с ключевыми столбцами и столбцами хранящими дату создания строки
- Значения генерируются и присваиваются автоматически
- Явное присваивание значения может привести к ошибкам
- Может потребоваться вернуть автоматически сгенерированное значение приложению
- Нужно идентифицировать такие столбцы

14 - 8 SQL

Автоматическая генерация значений столбца

Модель данных может предусматривать для некоторых таблиц автоматическую генерацию значений столбцов. Вставка явных значений командой INSERT возможна, но это может привести к ошибкам в будущем. Важно идентифицировать такие столбцы и не вставлять в них значения явно.

Столбцы с автоматической генерацией значений обычно используют:

Тип данных или свойство столбца, которое при вставке строки генерирует инкрементальные значения. Это используется, обычно, для столбцов, являющихся первичным ключом. В команде INSERT значение для такого столбца либо не устанавливается, либо вставляется NULL. Приложению, выполняющему INSERT часто требуется получить значение, которое было автоматически сгенерировано. Команда INSERT должна вернуть значение, использовать SELECT для выборки значения после INSERT нельзя, так как значение может обновиться другой транзакцией, либо нужно выполнять команды в рамках одной транзакции.

Свойство столбца DEFAULT, указывающее функцию для вычисления значения столбца. Обычно используется для столбцов, сохраняющих время создания строки. Функция при этом выдаёт текущие дату и время.

Триггера (процедуры, выполняющиеся по событию) на INSERT, которые устанавливают значение в полях вставляемой строки. Триггера на порядок замедляют вставку, поэтому, лучше их не использовать.

Объект БД последовательность (sequence), которая должна использоваться для генерации значения конкретного столбца таблицы. Ограничений на необходимость использования последовательности со стороны БД обычно нет и разработчик должен явно использовать последовательность в INSERT. Команде INSERT обычно указывают обращение к последовательности в качестве значения для столбца.

Автоинкрементальные столбцы в БД Oracle

- Начиная с версии 12c

```
create table emp (n INTEGER
 GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY);
INSERT INTO emp VALUES (DEFAULT);
```

- Можно задать поведение при вставке NULL или явного задания значения

Автоинкрементальные столбцы в БД Oracle

В БД Oracle используется синтаксис:

```
GENERATED [ ALWAYS | BY DEFAULT [ ON NULL ] ] AS IDENTITY [
(START WITH 1 INCREMENT BY 1 и другие опции ) ]
```

Создаётся служебная последовательность ISEQ$$_номер . Список таких последовательностей имеется в представлении sys.idnseq$.

Начиная с версии 12c в опции DEFAULT можно использовать последовательности явно:

```
CREATE SEQUENCE t1_seq;
CREATE TABLE t1 (id INTEGER DEFAULT t1_seq.NEXTVAL);
```

Практическое задание 14

- Создание и использование представлений
- Создание последовательностей

14 - 10 SQL

Практическое задание 14

15

Получение информации об объектах

Рассматриваемые вопросы

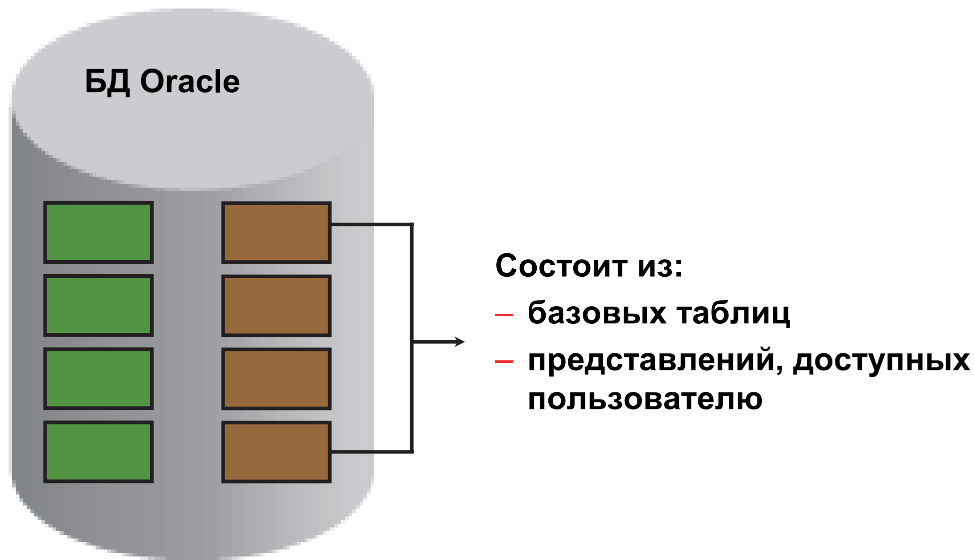
- **Получение информации об объектах, хранящихся в БД**
- **Словарь БД Oracle**

15 - 2 SQL

Цели урока

В этом уроке вы познакомитесь с методами получения информации об объектах БД - метаданным. Эта информация нужна в процессе разработки, чтобы получать данные о существующих объектах. В коде приложений нечасто возникает потребность обращаться к метаданным в процессе работы. Поэтому можно создавать приложения, не привязанные к БД конкретных производителей.

Структура словаря данных



15 - 3 SQL

Структура словаря данных

В базовых таблицах хранится информация о БД. На них созданы представления словаря БД.

С помощью представлений обобщается и выводится информация, хранимая в базовых таблицах словаря данных. В этих представлениях данные базовых таблиц декодируются в полезную информация (например, в имена пользователей и наименования таблиц). При этом используются соединения и предложения `WHERE`, в результате чего упрощается выдаваемая информация. Большинству пользователей предоставляется доступ к представлениям, а не к базовым таблицам.

Структура словаря данных

Соглашение по наименованиям представлений:

Префикс представления	Назначение
`USER`	Пользовательское представление (что находится в вашей схеме; что вам как пользователю принадлежит)
`ALL`	Расширенное пользовательское представление (к чему у вас есть доступ)
`DBA и CDB`	Представление администратора базы данных (что находится в каждой схеме)
`V$`	Служебные данные

Структура словаря данных

Словарь данных состоит из множеств представлений. Во многих случаях каждое отдельное множество содержит три представления, позволяющих вывести похожую информацию и отличающихся префиксом друг от друга. Например, представления `USER_OBJECTS`, `ALL_OBJECTS` и `DBA_OBJECTS`.

В этих трех представлениях содержится схожая информация об объектах базы данных, отличающаяся областью действия. `USER_OBJECTS` предоставляет информацию об объектах, которые были созданы вами или принадлежат вам как пользователю-владельцу. `ALL_OBJECTS` отображает все объекты, к которым у вас есть доступ. `DBA_OBJECTS` содержит информацию о всех объектах, принадлежащих всем пользователям. Обычно в представлениях с префиксом `ALL` и `DBA` имеется столбец `OWNER`, показывающий, кому принадлежит объект.

Кроме того, существует множество представлений с префиксом `v$` в названии. Это табличный интерфейс для доступа к данные из служебных структур памяти и служебных файлов.

Как использовать представления словаря

Начните с представления DICTIONARY. В нем содержатся наименования и описания таблиц и представлений словаря.

```
DESCRIBE DICTIONARY
```

Name	Null?	Type
TABLE_NAME		VARCHAR2(30)
COMMENTS		VARCHAR2(4000)

```
SELECT *
FROM   dictionary
WHERE  table_name = 'USER_OBJECTS';
```

TABLE_NAME	COMMENTS
USER_OBJECTS	Objects owned by the user

Как использовать представления словаря

Представление DICTIONARY знакомит со словарем данных. Оно содержит наименования и краткие описания каждого представления, доступного данному пользователю.

В запросе можно задать поиск информации по имени определенного представления. Можно также использовать слово или фразу для поиска по столбцу COMMENTS, который описывает, что содержит представление словаря БД.

Второе полезное представление DICT_COLUMNS:

```
TABLE_NAME       VARCHAR2(128)
COLUMN_NAME      VARCHAR2(128)
COMMENTS         VARCHAR2(4000)
```

Из него можно узнать, что хранится в столбцах представлений словаря БД.

Представления USER_OBJECTS и ALL_OBJECTS

USER_OBJECTS:

- **Запрос к представлению USER_OBJECTS используется для просмотра сведений об объектах, принадлежащих пользователю**
- **Можно получить список имен и типов всех объектов схемы пользователя, а также следующие сведения:**
  - **Дата создания**
  - **Дата последнего изменения**
  - **Статус (valid или invalid)**

ALL_OBJECTS:

- **Запрос к представлению ALL_OBJECTS используется для просмотра сведений о всех объектах, к которым имеет доступ пользователь.**

15 - 6 SQL

Представление USER_OBJECTS

Запрос к представлению USER_OBJECTS используется для просмотра имен и типов объектов схемы пользователя. Ниже приведены некоторые из столбцов этого представления:

- **OBJECT_NAME:** наименование объекта;
- **OBJECT_ID:** номер объекта в словаре;
- **OBJECT_TYPE:** тип объекта (например, TABLE, VIEW, INDEX, SEQUENCE);
- **CREATED:** временная метка создания объекта;
- **LAST_DDL_TIME:** временная метка последнего изменения объекта с помощью команды DDL;
- **STATUS:** статус объекта (VALID, INVALID или N/A)
- **GENERATED:** было ли наименование объекта сгенерировано системой? (Y|N)

Кроме того, можно запрашивать представление ALL_OBJECTS для просмотра перечня всех объектов, к которым пользователь имеет право доступа.

Представление USER_OBJECTS

```
SELECT object_name, object_type, created, status
FROM   user_objects
ORDER BY object_type;
```

OBJECT_NAME	OBJECT_TYPE	CREATED	STATUS
REG_ID_PK	INDEX	10-DEC-03	VALID
...			
DEPARTMENTS_SEQ	SEQUENCE	10-DEC-03	VALID
REGIONS	TABLE	10-DEC-03	VALID
LOCATIONS	TABLE	10-DEC-03	VALID
DEPARTMENTS	TABLE	10-DEC-03	VALID
JOB_HISTORY	TABLE	10-DEC-03	VALID
JOB_GRADES	TABLE	10-DEC-03	VALID
EMPLOYEES	TABLE	10-DEC-03	VALID
JOBS	TABLE	10-DEC-03	VALID
COUNTRIES	TABLE	10-DEC-03	VALID
EMP_DETAILS_VIEW	VIEW	10-DEC-03	VALID

Представление USER_OBJECTS

В примере на слайде выводятся наименование, тип, дата создания и статус для всех объектов, принадлежащих текущему пользователю.

В столбце OBJECT_TYPE может находиться одно их следующих значений: TABLE, VIEW, SEQUENCE, INDEX, PROCEDURE, FUNCTION, PACKAGE или TRIGGER.

Столбец STATUS содержит VALID, INVALID или N/A. В то время, как таблицы всегда в правильном состоянии (VALID), представления, процедуры, функции, пакеты и триггеры могут быть неверными (INVALID).

Представление CAT

Представление CAT упрощает получение сведений об объектах. Оно содержит только два столбца: TABLE_NAME и TABLE_TYPE. С его помощью для текущего пользователя выводятся наименования всех объектов следующего типа: INDEX, TABLE, CLUSTER, VIEW, SYNONYM, SEQUENCE или UNDEFINED.

Информация о таблицах

USER_TABLES:

```
DESCRIBE user_tables
```

Name	Null?	Type
TABLE_NAME	NOT NULL	VARCHAR2(30)
TABLESPACE_NAME		VARCHAR2(30)
CLUSTER_NAME		VARCHAR2(30)
IOT_NAME		VARCHAR2(30)

```
SELECT table_name
FROM   user_tables;
```

TABLE_NAME
JOB_GRADES
REGIONS
COUNTRIES
LOCATIONS
DEPARTMENTS

...

Представление USER_TABLES

Представление USER_TABLES используется для вывода наименований всех таблиц пользователя. Кроме того, в представлении содержится детальная информация о хранении таблиц.

Можно запрашивать представление ALL_TABLES для просмотра перечня всех таблиц, к которым пользователь имеет право доступа.

Информация о столбцах

```
SELECT column_name, data_type, data_length,
       data_precision, data_scale, nullable
FROM   user_tab_columns
WHERE  table_name = 'EMPLOYEES';
```

COLUMN_NAME	DATA_TYPE	DATA_LENGTH	DATA_PRECISION	DATA_SCALE	NUL
EMPLOYEE_ID	NUMBER	22	6	0	N
FIRST_NAME	VARCHAR2	20			Y
LAST_NAME	VARCHAR2	25			N
EMAIL	VARCHAR2	25			N
PHONE_NUMBER	VARCHAR2	20			Y
HIRE_DATE	DATE	7			N
JOB_ID	VARCHAR2	10			N
SALARY	NUMBER	22	8	2	Y
COMMISSION_PCT	NUMBER	22	2	2	Y
MANAGER_ID	NUMBER	22	6	0	Y
DEPARTMENT_ID	NUMBER	22	4	0	Y

Информация о столбцах (продолжение)

Данные о таблицах пользователя и их хранении содержатся в представлении USER_TABLES, а подробные сведения о столбцах таблиц – в USER_TAB_COLUMNS.

Представление USER_TAB_COLUMNS содержит, например, следующие столбцы:

- имя столбца;
- тип данных столбца;
- длина данных определенного типа в байтах;
- точность и масштаб для столбца с типом NUMBER;
- разрешено ли неопределенное значение (определено ли для столбца ограничение NOT NULL?);
- значение по умолчанию.

Используя запрос к представлению USER_TAB_COLUMNS, можно выяснить такие детальные сведения о столбцах: наименование, тип данных, длина данных, разрешено ли неопределенное значение, а также значение по умолчанию.

В примере на слайде выводятся данные о столбцах таблицы EMPLOYEES.

Информация об ограничениях целостности

```
SELECT constraint_name, constraint_type,
       search_condition, r_constraint_name,
       delete_rule, status
FROM   user_constraints
WHERE  table_name = 'EMPLOYEES';
```

CONSTRAINT_NAME	CON	SEARCH_CONDITION	R_CONSTRAINT_NAME	DELETE_RULE	STATUS
EMP_LAST_NAME_NN	C	"LAST_NAME" IS NOT NULL			ENABLED
EMP_EMAIL_NN	C	"EMAIL" IS NOT NULL			ENABLED
EMP_HIRE_DATE_NN	C	"HIRE_DATE" IS NOT NULL			ENABLED
EMP_JOB_NN	C	"JOB_ID" IS NOT NULL			ENABLED
EMP_SALARY_MIN	C	salary > 0			ENABLED
EMP_EMAIL_UK	U				ENABLED
EMP_EMP_ID_PK	P				ENABLED
EMP_DEPT_FK	R		DEPT_ID_PK	NO ACTION	ENABLED
EMP_JOB_FK	R		JOB_ID_PK	NO ACTION	ENABLED
EMP_MANAGER_FK	R		EMP_EMP_ID_PK	NO ACTION	ENABLED

USER_CONSTRAINTS: пример

В приведенном примере выполняется запрос к представлению USER_CONSTRAINTS. В результате для таблицы EMPLOYEES выводятся наименования ограничений и их типы, условия ограничений check, наименования ограничений главного ключа, на которые ссылаются внешние ключи, правила удаления строк с внешним ключом, а также статус ограничений.

Столбец CONSTRAINT_TYPE может иметь следующие значения:

- C (ограничение check для таблицы);
- P (главный ключ);
- U (уникальный ключ);
- R (ограничение ссылочной целостности);
- V (ограничение для представления с фразой with check option);
- O (ограничение для представления с фразой with read only).

Значение столбца DELETE_RULE:

- CASCADE; при удалении строки главной таблицы также удаляются зависимые строки подчиненной таблицы;
- NO ACTION; строка главной таблицы удаляется при условии, что нет зависимых от нее строк в подчиненной таблице.

Значение столбца STATUS:

- ENABLED; ограничение проверяется;
- DISABLED; ограничение не проверяется.

Информация об ограничениях целостности

```
DESCRIBE user_cons_columns
```

Name	Null?	Type
OWNER	NOT NULL	VARCHAR2(30)
CONSTRAINT_NAME	NOT NULL	VARCHAR2(30)
TABLE_NAME	NOT NULL	VARCHAR2(30)
COLUMN_NAME		VARCHAR2(4000)
POSITION		NUMBER

```
SELECT constraint_name, column_name
FROM   user_cons_columns
WHERE  table_name = 'EMPLOYEES';
```

CONSTRAINT_NAME	COLUMN_NAME
EMP_EMAIL_UK	EMAIL
EMP_SALARY_MIN	SALARY
EMP_JOB_NN	JOB_ID
EMP_HIRE_DATE_NN	HIRE_DATE

...

15 - 11 SQL

Запрос к представлению USER_CONS_COLUMNS

Чтобы выявить столбцы, к которым применяется ограничение, необходимо запросить представление словаря USER_CONS_COLUMNS. С помощью представления можно узнать имя владельца ограничения, его наименование, таблицу, к которой применяется ограничение, а также позицию столбца или атрибута в определении ограничения.

Примечание: ограничение может применяться к нескольким столбцам.

Для получения требуемой информации можно также соединить в запросе представления USER_CONSTRAINTS и USER_CONS_COLUMNS.

Информация о представлениях

1

```
DESCRIBE user_views
```

Name	Null?	Type
VIEW_NAME	NOT NULL	VARCHAR2(30)
TEXT_LENGTH		NUMBER
TEXT		LONG

2

```
SELECT DISTINCT view_name FROM user_views;
```

VIEW_NAME
EMP_DETAILS_VIEW

3

```
SELECT text FROM user_views
WHERE view_name = 'EMP_DETAILS_VIEW';
```

TEXT
SELECT e.employee_id, e.job_id, e.manager_id, e.department_id, d.locat ion_id, l.country_id, e.first_name, e.last_name, e.salary, e.commissio n_pct, d.department_name, j.job_title, l.city, l.state_province, c.cou ntry_name, r.region_name FROM employees e, departments d, jobs j, loca tions l, countries c, regions r WHERE e.department_id = d.department_id AN D d.location_id = l.location_id AND l.country_id = c.country_id AND c.region _id = r.region_id AND j.job_id = e.job_id WITH READ ONLY

Сведения о представлениях, хранимые в словаре данных

После того, как представление было создано, его имя и определение можно запросить из представления словаря данных USER_VIEWS. Текст команды SELECT, определяющей представление, хранится в столбце TEXT типа LONG.

В примерах на слайде:

1. Выводятся столбцы представления USER_VIEWS. На слайде не полностью отражены результаты команды DESCRIBE user_views.
2. Выводятся наименования представлений.
3. По команде SELECT выводится текст запроса, который определяет представление EMP_DETAILS_VIEW.

Доступ к данным с использованием представлений

Когда вы обращаетесь к данным через представление, БД выполняет следующие операции:

- Выбирает определение представления из USER_VIEWS.
- Проверяет привилегии доступа к базовой таблице, на которой основано представление.
- Преобразует запрос представления в эквивалентную операцию с базовой таблицей или таблицами. Иными словами, выборка и обновление данных производятся в базовых таблицах.

Информация о последовательностях

```
DESCRIBE user_sequences
```

Name	Null?	Type
SEQUENCE_NAME	NOT NULL	VARCHAR2(30)
MIN_VALUE		NUMBER
MAX_VALUE		NUMBER
INCREMENT_BY	NOT NULL	NUMBER
CYCLE_FLAG		VARCHAR2(1)
ORDER_FLAG		VARCHAR2(1)
CACHE_SIZE	NOT NULL	NUMBER
LAST_NUMBER	NOT NULL	NUMBER

15 - 13 SQL

Представление USER_SEQUENCES

В представлении USER_SEQUENCES хранятся описания всех последовательностей, принадлежащих пользователю. Атрибуты последовательностей задаются пользователем при их создании и доступны с помощью представления USER_SEQUENCES. В нем находятся следующие столбцы:

- SEQUENCE_NAME; наименование последовательности.
- MIN_VALUE; минимальное значение последовательности.
- MAX_VALUE; максимальное значение последовательности.
- INCREMENT_BY; значение, задающее шаг последовательности.
- CYCLE_FLAG; продолжать ли генерацию значений последовательности сначала после достижения максимума (минимума)?
- ORDER_FLAG; генерируются ли числа последовательности по порядку?
- CACHE_SIZE; количество чисел последовательности в кэш-памяти.
- LAST_NUMBER; последний номер последовательности, записанный на диск. Если для последовательности используется кэш, тогда число, записанное на диск, – это последнее число, размещенное в кэше последовательности. Такое число, скорее всего, будет больше используемого последнего значения.

Информация о последовательностях

- **Проверить значения последовательности можно в представлении `USER_SEQUENCES` словаря данных.**

```
SELECT   sequence_name, min_value, max_value,
         increment_by, last_number
FROM     user_sequences;
```

SEQUENCE_NAME	MIN_VALUE	MAX_VALUE	INCREMENT_BY	LAST_NUMBER
LOCATIONS_SEQ	1	9900	100	3300
DEPARTMENTS_SEQ	1	9990	10	280
EMPLOYEES_SEQ	1	1.0000E+27	1	207

- **Столбец `LAST_NUMBER` содержит следующее свободное число, если установлен параметр `NOCACHE`.**

15 - 14 SQL

Проверка параметров последовательности

С момента создания последовательности она документируется в словаре данных. Т.к. последовательность является объектом базы данных, информация о ней имеется в представлении словаря данных `USER_OBJECTS`.

Проверить установки для последовательности можно путем выборки информации из представления словаря данных `USER_SEQUENCES`.

Просмотр следующего доступного значения последовательности

Если последовательность была создана с опцией `NOCACHE`, тогда можно просмотреть с помощью представления `USER_SEQUENCES` следующее доступное значение, не выбирая его из последовательности.

Практическое задание 15

- Запрос информации о таблицах и столбцах
- Запрос информации о представлениях
- Запрос информации о последовательностях
- Добавление комментария к таблице

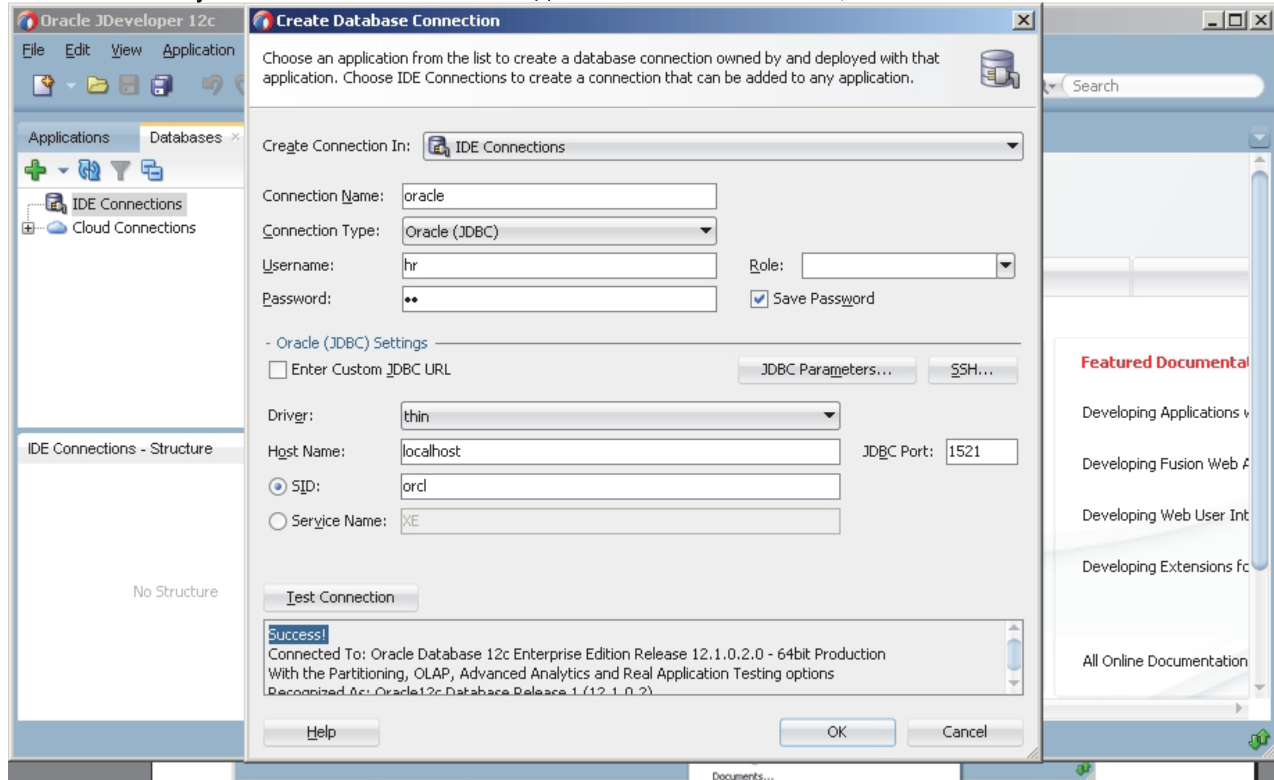
Практическое задание 15

На этом практическом занятии вы будете выполнять запросы для того, чтобы получить сведения об объектах БД.

Упражнения

Практическое задание 1
1. Запустите JDeveloper

3. Кликните на IDE Connections, нажмите правую кнопку мыши и выберите **New Connection**.
Введите в поле Connection Name: **hr**
В поле Username: hr
В поле Password: hr
галочку в Save Password
В поле Connection type: TNS
Network Alias; SIEBELDB
Нажмите кнопку Test Connection и если под ней появится Success!, то нажмите OK.

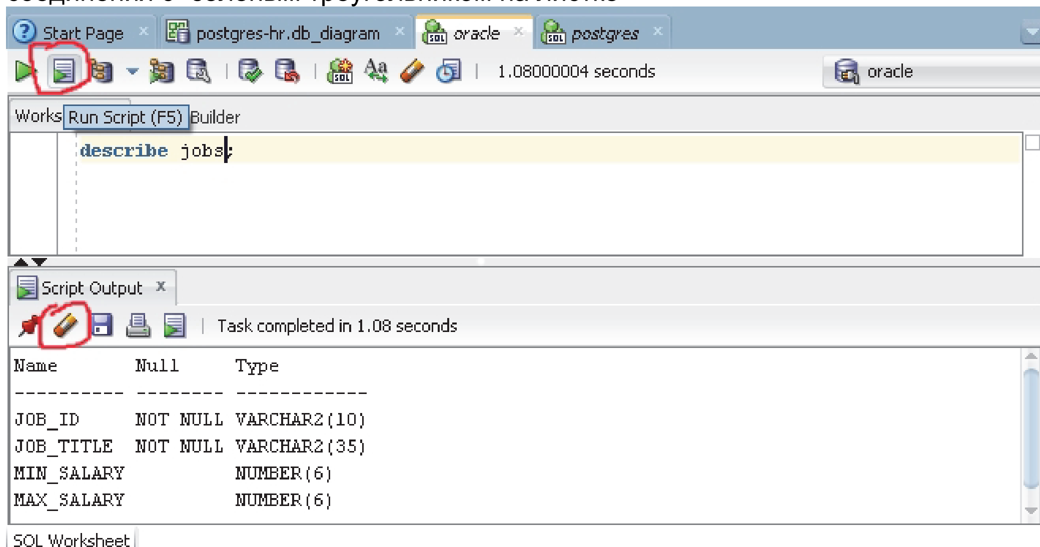


Вы создали соединение с БД Oracle.

7. Посмотрите текст комментария к таблице.
Комментарии используются для разработчиков, приложения с ними не работают.
Комментарии были созданы стандартными командами SQL:

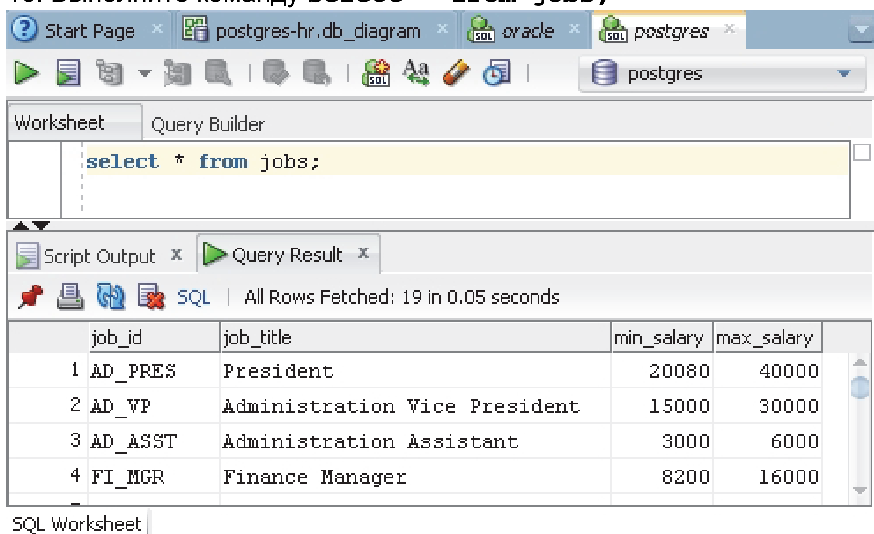
```
COMMENT ON TABLE COUNTRIES  IS 'country table. Contains 25 rows. References with
locations table.';
COMMENT ON COLUMN COUNTRIES.COUNTRY_ID IS 'Primary key of countries table.';
```

13. Выберите окно oracle и наберите в нём текст: "`describe employees;`".
Для выполнения набранной команды нажмите клавишу на клавиатуре **F5** или кнопку в тулбаре окна соединения с "зелёным треугольником на листке"



В окне Script Output появится список столбцов таблицы jobs. На тулбаре этого окна нажмите кнопку с изображением ластика (вторая слева) или Ctrl+Shift+D. Окно с результатом очистится.

16. Выполните команду **`select * from jobs;`**



	job_id	job_title	min_salary	max_salary
1	AD_PRES	President	20080	40000
2	AD_VP	Administration Vice President	15000	30000
3	AD_ASST	Administration Assistant	3000	6000
4	FI_MGR	Finance Manager	8200	16000

24. Наберите "**`select user from dual;`**" Появится имя пользователя, под которым вы подсоединены к БД, а также время, за которое команда была выполнена.

```
SQL> select user from dual;
USER
------------------------------
HR

Elapsed: 00:00:00.036
SQL>
```

Практическое задание 2

Решения и ответы приведены в конце практического задания. При выполнении задания используйте соединеиея, которые вы создали в JDevelper.

1. Будет ли успешно выполнена эта команда `SELECT`:

```
SELECT last_name, salary AS Sal
FROM   employees;
```

2. Будет ли успешно выполнена эта команда `SELECT`:

```
SELECT * FROM employees;
```

3. Команда `SELECT` содержит 4 ошибки. Укажите их

```
SELECT    employee_id, last_name
sal x 12  ANNUAL SALARY
FROM      employees;
```

4. Выясните структуру таблицы `DEPARTMENTS`.

а) в `sqlci`

б) в `psql`

в) JDeveloper

5. Выберите данные обо всех должностях из таблицы `EMPLOYEES`. В выводимых результатах идентификаторы должностей не должны повторяться.

```
JOB_ID
----------
IT_PROG
AC_MGR
AC_ACCOUNT
...
 19 rows selected
```

6. Выберите данные, включающие номер и дату найма сотрудника из таблицы employees. Номер должен выводиться первым. Определите псевдоним `STARTDATE` для столбца `HIRE_DATE`.

```
EMPLOYEE_ID STARTDATE
----------- -------------------------------
        100 17-JUN-03
        101 21-SEP-05
        102 13-JAN-01
...
 107 rows selected
```

7. Назначьте в нём псевдонимы столбцов **Empno** и **Hire Date**:

```
     Empno Hire Date
---------- -------------------------------
       100 17-JUN-03
       101 21-SEP-05
       102 13-JAN-01
...
 107 rows selected
```

Обратите внимаение, что по умолчанию заголовки столбцов выдаются несколько раз. В окне JDeveloper добавьте перед командой SELECT команду **set pagesize 50000** и повторите команду. Это команда программ sqlcl и JDeveloper. Строки выведутся без лишних заголовков.

8. В JDeveloper выберите данные обо всех сотрудниках и идентификаторах их должностей. Выведите на экран имя, соединенное с идентификатором должности через запятую и пробел. Назовите выводимый столбец `ET`.

```
ET
-------------------------------------
King, AD_PRES
Kochhar, AD_VP
De Haan, AD_VP
...
Gietz, AC_ACCOUNT
```

```
107 rows selected
```

9.Создайте запрос для вывода имени и фамилии сотрудников из таблицы EMPLOYEES. Разделите значения столбцов запятыми и выведите их в одном столбце с наименованием WITH.
Почему следующий запрос не выполняется?

```
SELECT first_name || ',' || last_name WITH FROM employees;
```

Исправьте этот запрос так, чтобы он выполнялся в соединениях oracle и postgres
10. Замените слово WITH на OUT и выполните в соединениях oracle и postgres:

```
SELECT first_name || ',' || last_name OUT FROM employees;
```

В соединении oracle запрос выполнился, в соединении postgres выдал ошибку.
Почему?
11. Добавьте AS перед OUT и повторите запрос в обоих соединениях. Запрос выполнится успешно:

```
SELECT first_name || ',' || last_name AS OUT FROM employees;
```

Что лучше использовать AS или обрамлять псевдоним столбца двойными кавычками?

Практическое задание 2, решения

1. Будет ли успешно выполнена эта команда SELECT:

```
SELECT last_name, salary AS Sal
FROM   employees;
```

Да

2. Будет ли успешно выполнена эта команда SELECT:

```
SELECT * FROM employees;
```

Да

3. Команда SELECT содержит 4 ошибки. Укажите их

```
SELECT     employee_id, last_name
sal x 12   ANNUAL SALARY
FROM       employees;
```

1) Таблица EMPLOYEES не содержит столбец sal. Столбец называется SALARY.
2) Оператором умножения является *, а не *x*, как в строке 2.
3) Псевдоним не может включать пробелы. Псевдоним ANNUAL SALARY должен быть написан как ANNUAL_SALARY или заключен в двойные кавычки.
4) Пропущена запятая после столбца LAST_NAME.

4. Выясните структуру таблицы DEPARTMENTS.
sqlci: DESCRIBE departments;
psql: \d departments;
JDevelper: структура видна визуально

5. Выберите данные обо всех должностях из таблицы EMPLOYEES. В выводимых результатах идентификаторы должностей не должны повторяться.

```
SELECT DISTINCT job_id FROM employees;
```

6. Выберите данные, включающие номер и дату найма сотрудника из таблицы employees. Номер должен выводиться первым. Определите псевдоним STARTDATE для столбца HIRE_DATE.

```
SELECT employee_id, hire_date STARTDATE FROM employees;
```

7. Назначьте в нём псевдонимы столбцов **Empno** и **Hire Date**:

```
SELECT employee_id "Empno", hire_date "Hire Date" FROM employees;
```

В окне JDeveloper добавьте перед командой SELECT команду **set pagesize 50000** и повторите команду. Это команда программ sqlcl и JDeveloper:

```
set pagesize 50000
SELECT employee_id "Empno", hire_date "Hire Date" FROM employees;
```

Выполните вышеприведенную команду SELECT в psql:

```
SELECT employee_id "Empno", hire_date "Hire Date" FROM employees;
```

8. Выберите данные обо всех сотрудниках и идентификаторах их должностей. Выведите на экран имя, соединенное с идентификатором должности через запятую и пробел. Назовите выводимый столбец ET.

```
SELECT last_name||', '||job_id "ET" FROM employees;
```

9.Создайте запрос для вывода имени и фамилии сотрудников из таблицы EMPLOYEES. Разделите значения столбцов запятыми и выведите их в одном столбце с наименованием OUT.
Исправьте нижеприведенный запрос:

```
SELECT first_name || ',' || last_name WITH FROM employees;
```

WITH зарезервированное слово в обеих БД, его нужно взять в двойные кавычки:

```
SELECT last_name "WITH" FROM employees;
```

В postgres также можно указать слово AS перед WITH и команда выполнится:

```
SELECT last_name AS WITH FROM employees;
```

Для БД Oracle слова AS недостаточно.

10. В БД Oracle слово OUT не считается зарезервированным.
БД postgres считает зарезервированными все слова, упоминаемые в стандартах SQL, даже если они не используются в настоящее время.

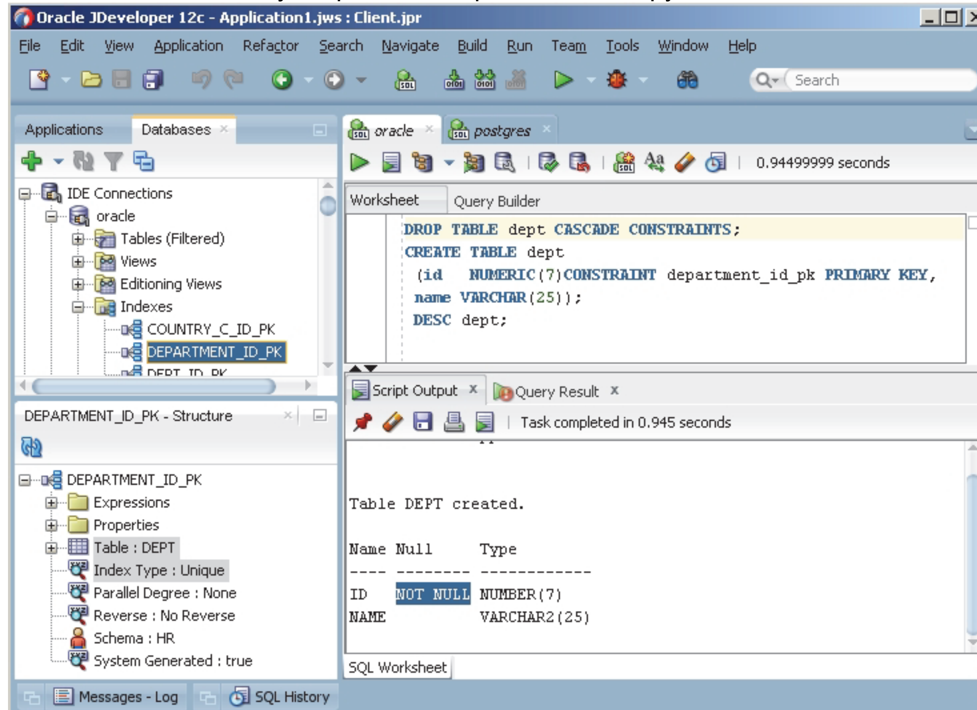
11. Чтобы избегать ошибок с зарезервированными словами стоит брать алиасы в двойные кавычки.

Практическое задание 3
Ответы приведены в конце этого задания.

3. Создайте таблицу `DEPT` в соединении oracle:

```
CREATE TABLE dept
 (id   NUMBER(7) CONSTRAINT department_id_pk PRIMARY KEY,
  name VARCHAR2(25));
```

4. При создании ограничения целостности с типами первичный (PK) или уникальный ключ (UK) автоматически создаётся уникальный индекс. Раскройте в JDeveloper в окне Databases слева вверху IDE Connections-> oracle -> Indexes. Встав на Indexes нажмите клавишу F5 (или по правой кнопке мыши Refresh или иконку с синими стрелками вверху окна). Выберите мышью индекс с названием DEPARTNENT_ID_PK. В окне "DEPARTNENT_ID_PK - Structure" слева внизу вы увидите описание индекса. Индексу было автоматически назначено имя ограничения целостности, что позволяет идентифицировать индекс по его названию. Для удобства рекомендуется давать имена ограичениям целостности, однако, нужно следить, чтобы эти имена не дублировались при создании других таблиц.



Обратите внимание, что Index Type: Unique, что означает, что индекс был создан со свойством уникальности и не позволит вставлять строки в DEPT, если значения в столбце ID дублируются.

6. В таблице DEPT нет данных. Если загружать в таблицу большое количество строк не используя специальные техники, то при вставках будут обновляться записи и в индексе. Это приведёт к замедлению вставок и неоптимальности структуры индекса. Лучше действовать в следующем порядке: создать таблицу без первичных и уникальных ключей, загрузить данные, создать индексы, добавить ограничения целостности.
Удалите таблицу DEPT в соединении postgres:

```
DROP TABLE dept;
```

Создайте таблицу без первичного ключа:

```
CREATE TABLE dept
 (id   NUMERIC(7),
 name VARCHAR(25));
```

7. Заполните таблицу `DEPT` данными из таблицы `DEPARTMENTS` в соединениях oracle и postgres. Данные возьмите из столбцов `department_id`, `department_name`.

БД Oracle позволяет создавать индексы на тот же набор столбцов, но они должны отличаться по типу или другим характеристикам. Также в БД Oracle можно перестраивать индексы с опцией ONLINE, которая блокирует индекс на очень короткое время.

15. Создайте таблицу `EMP` в JDeveloper в соединении oracle:

```
CREATE TABLE  emp
```

```
  (id           NUMBER(7),
   last_name    VARCHAR2(25),
   first_name   VARCHAR2(25),
   dept_id      NUMBER(7)
     CONSTRAINT emp_dept_id_FK REFERENCES dept (id));
```

В этой команде на уровне столбца определяется внешний ключ.
Перепишите команду так, чтобы она работала в postgres и выполните её в соединении postgres.

16. Удалите таблицу emp и создайте её с ограничением целостности со свойствами каскадного удаления и возможности отложенной проверки. Используйте синтаксис определения ограничения целостности не на уровне столбца, а на уровне таблицы, так как этот синтаксис более универсален - он позволяет указывать несколько столбцов в ограничении.

17. **После того, как вы успешно выполнили предыдущий пункт задания, попробуйте** удалить таблицу `DEPT`:

```
DROP TABLE dept;
```

Вы получите ошибку:

```
ОШИБКА: удалить объект таблица dept нельзя, так как от него зависят другие объекты
  Detail: ограничение emp_dept_id_fk в отношении таблица emp зависит от объекта таблица
dept Hint: Для удаления зависимых объектов используйте DROP ... CASCADE.
```

18. Создайте таблицу `EMPLOYEES2` на основе структуры таблицы EMPLOYEES в postgres и oracle.

```
CREATE TABLE employees2 AS
  SELECT  *
  FROM    employees;
```

19. В соединении oracle дайте команду **`DESC employees2;`** и подсчитайте колличество строк.

```
select count(*) from employees2;
  COUNT(*)
----------
       108
DESC employees2;
```

Вы увидите ограничения целостности NOT NULL:

```
Name           Null     Type
-------------- -------- ------------
EMPLOYEE_ID             NUMBER(6)
FIRST_NAME              VARCHAR2(20)
LAST_NAME      NOT NULL VARCHAR2(25)
EMAIL          NOT NULL VARCHAR2(25)
PHONE_NUMBER            VARCHAR2(20)
HIRE_DATE      NOT NULL DATE
JOB_ID         NOT NULL VARCHAR2(10)
SALARY                  NUMBER(8,2)
COMMISSION_PCT          NUMBER(2,2)
MANAGER_ID              NUMBER(6)
DEPARTMENT_ID           NUMBER(4)
```

21. Создайте и заполните временную таблицу в соединениях oracle и postgres в JDeveloper

```
DROP TABLE emp_temp;
CREATE GLOBAL TEMPORARY TABLE emp_temp ON COMMIT PRESERVE ROWS AS select * from
employees;
insert into emp_temp select * from emp_temp;
insert into emp_temp select * from emp_temp;
select count(*) from emp_temp;
```

Вы получите 432 строки.

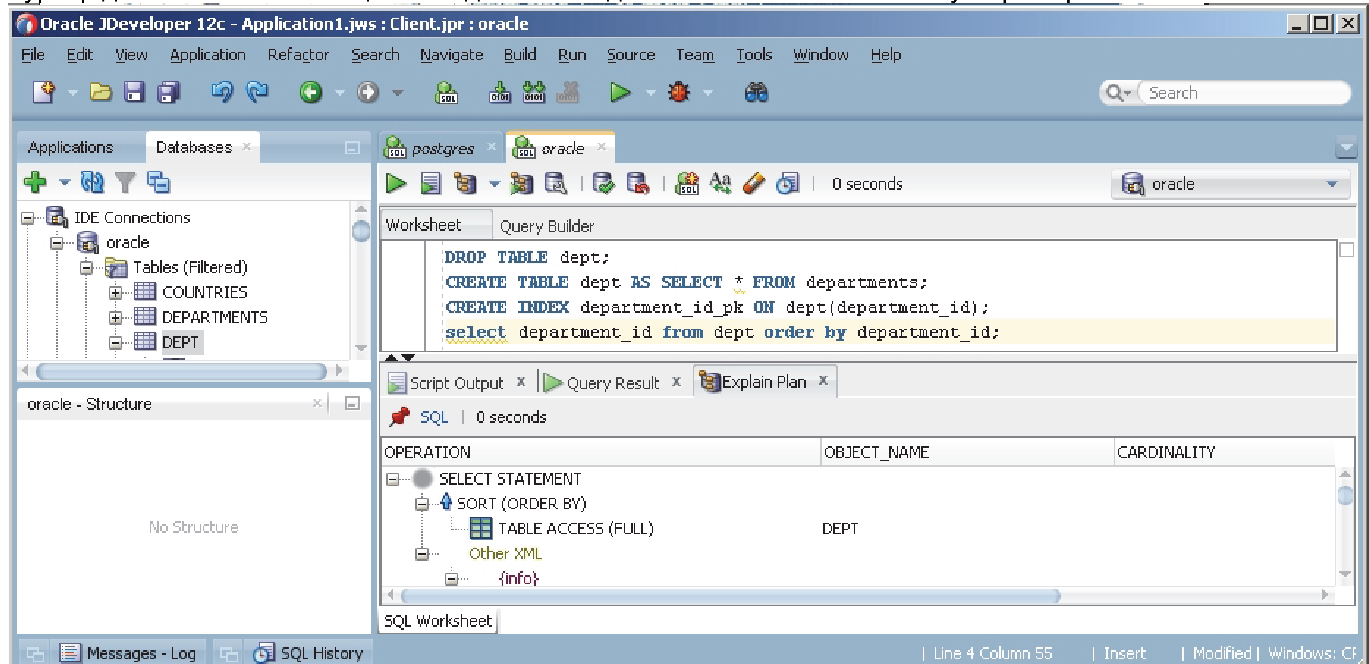
22. В соединении oracle нажмите F5. Вы можете получить большее колличество строк. Почему?
Посмотрите сообщения в окне Script Output. Вы увидите, что таблица не была удалена из-за ошибки
`SQL Error: ORA-14452: attempt to create, alter or drop an index on temporary table already in use.`
Чтобы устранить ошибку, вы можете закрыть соединение в окне Databases->IDE Connections->oracle по правой кнопке мыши выбрать Disconnect. На предложение Save Changes ответить No. Окно соединения oracle закроется. Чтобы его заново открыть нажмите кнопку на основном тулбаре JDeveloper с текстом

"SQL" и ыберите соединение oracle. В открывшемся окне вы сможете удалить таблицу командой `DROP TABLE emp_temp;`

23. В JDeveloper в соединении oracle наберите команды:

```
DROP TABLE dept;
CREATE TABLE dept AS SELECT * FROM departments;
CREATE INDEX department_id_pk ON dept(department_id);
select department_id from dept order by department_id;
```

Курсор должен стоять в конце последней команды. Нажмите F10 или иконку Explain plan

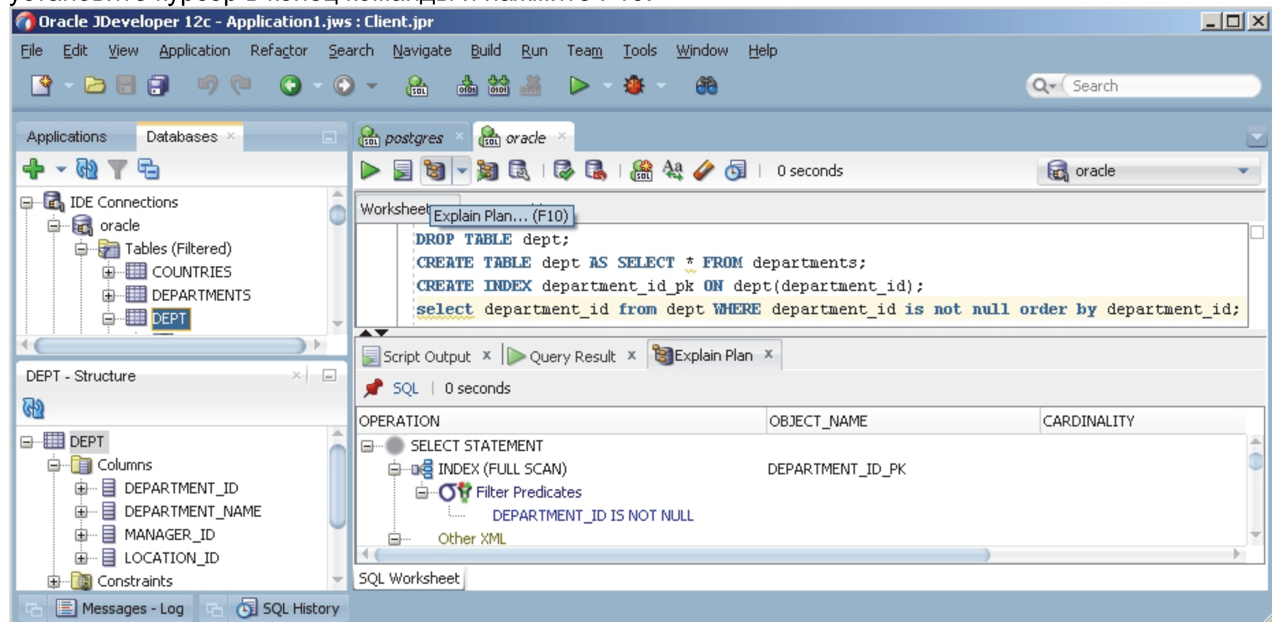


В плане выполнения команды вы увидите TABLE ACCESS (FULL) означающее полное сканирование блоков таблицы, без использования индекса.

24. Измените последнюю команду на

`select department_id from dept` **`WHERE department_id is not null`** `order by department_id;`

установите курсор в конец команды и нажмите F10.

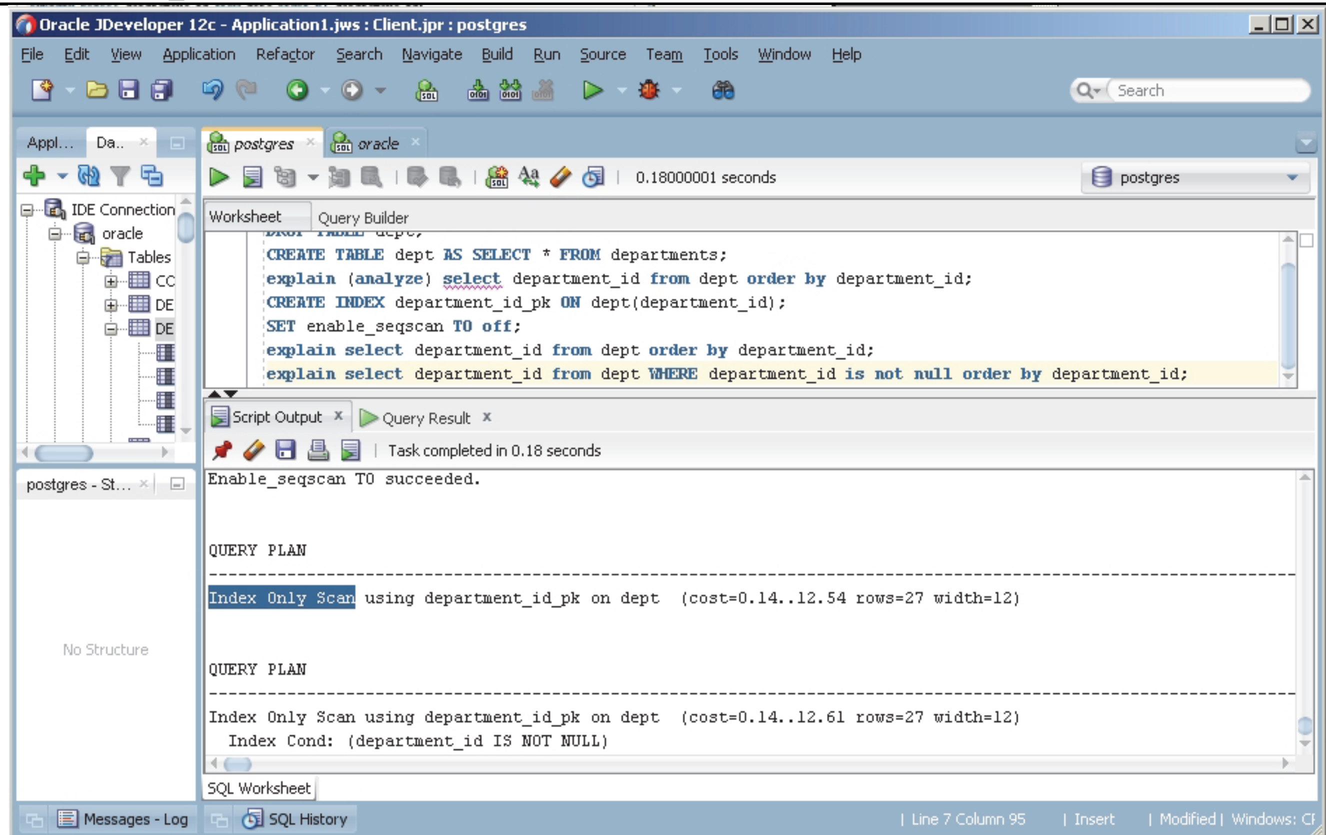


Вы увидите INDEX (FULL SCAN), что означает что идёт сканирование индекса, а не блоков талицы. Это проиошло благодаря добавлению условия IS NOT NULL. Индексы в БД Oracle не хранят пустые значения NULL и если они могут выдаваться в запросе (нет ограничения целостности NOT NULL или в самом запросе нет условия) идёт полное сканирование блоков таблицы.

25. В окне соединения postgres наберите команды и нажмите F5.

```
DROP TABLE dept;
```

```
CREATE TABLE dept AS SELECT * FROM departments;
explain (analyze) select department_id from dept order by department_id;
CREATE INDEX department_id_pk ON dept(department_id);
SET enable_seqscan TO off;
explain select department_id from dept order by department_id;
explain select department_id from dept WHERE department_id is not null order by
department_id;
SET enable_seqscan TO on;
```



Первое значение cost=0,14 затраты на получение первой строки. Второе ..12.54 затраты на получение всех строк.
Seq Scan - читается вся таблица. Index Scan - используется индекс для условий WHERE при этом читает блоки таблицы при выборке строк. Index Only Scan - читается только индекс.
При небольшом колличестве строк использование Seq Scan даёт меньшую стоимость, поэтому и использовалась команда SET enable_seqscan TO off;

26. В соединении postgres и oracle дайте команды:

```
drop table emp;
create table emp (n char(1));
insert into emp values('');
select * from emp where n='';
select * from emp where n=' ';
select * from emp where n is null;
```

Сравните результаты. В чём различия?

28. В соединении oracle дайте команды:

```
CREATE TABLE empi (v VARCHAR(4000), w VARCHAR(4000));
CREATE INDEX empi_idx ON empi(v,w);
DROP TABLE empi;
```

Вы получите ошибку при создании индекса:

```
SQL Error: ORA-01450: maximum key length (6398) exceeded
01450. 00000 -  "maximum key length (%s) exceeded"
```

Почему?

Практическое задание 3 решения

1. Создайте таблицу DUAL в JDeveloper в соединении postgres, чтобы можно было использовать её во фразе FROM для литералов и однострочных функций также, как в БД Oracle:

```
CREATE TABLE DUAL (DUMMY VARCHAR(1));
```

```
INSERT INTO DUAL VALUES ('X');
```

2. Перепишите команду так, чтобы она выполнялась в обеих БД:

```
SELECT 'A'||
'B' FROM DUAL;
```

3. Перепишите команду так, чтобы она работала в postgres и выполните её:

```
CREATE TABLE dept
 (id   NUMERIC(7) CONSTRAINT department_id_pk PRIMARY KEY,
 name VARCHAR(25));
```

7. Заполните таблицу `DEPT` данными из таблицы DEPARTMENTS в соединениях oracle и postgres:

```
INSERT INTO dept
  SELECT  department_id, department_name
  FROM    departments;
```

8. Создайте **НЕ**уникальный индекс в соединении postgres:

```
CREATE INDEX department_id_pk ON dept(id);
```

10. Удалите индекс и создайте уникальный:

```
DROP INDEX department_id_pk;
CREATE UNIQUE INDEX department_id_pk ON dept(id);
```

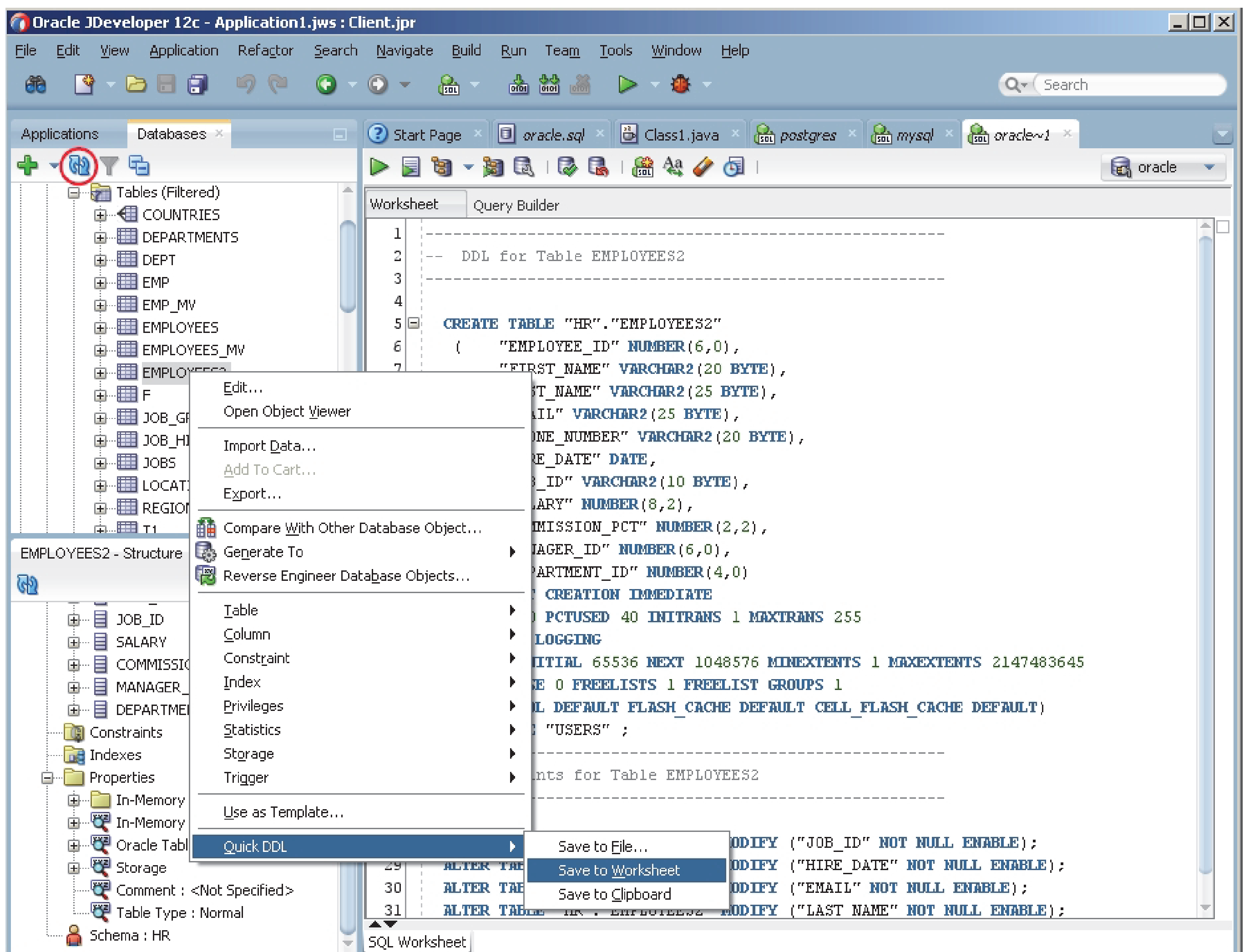
15. Перепишите команду так, чтобы она работала в postgres и выполните её:

```
CREATE TABLE  emp
  (id           NUMERIC(7),
   last_name    VARCHAR(25),
   first_name   VARCHAR(25),
   dept_id      NUMERIC(7)
     CONSTRAINT emp_dept_id_FK REFERENCES dept(id));
```

16. Удалите таблицу emp и создайте её с ограничением целостности со свойствами каскадного удаления и возможности отложенной проверки. Используйте синтаксис определения ограничения целостности не на уровне столбца, а на уровне таблицы, так как этот синтаксис более универсален - он позволяет указывать несколько столбцов в ограничении:

```
DROP TABLE emp;
CREATE TABLE emp (
id  NUMERIC(7),
last_name     VARCHAR(25),
first_name    VARCHAR(25),
dept_id       NUMERIC(7),
CONSTRAINT emp_dept_id_FK FOREIGN KEY (dept_id) REFERENCES dept (id) ON DELETE CASCADE
DEFERRABLE INITIALLY IMMEDIATE);
```

20. Проверьте, скопировалось ли что-либо кроме NOT NULL в БД Oracle при создании таблицы employees2. *Подсказка:* выберите в списке таблиц навигатора соединений с БД таблицу EMPLOYEES2 и по правой кнопке мыши выберите Quick DDL->Save to Worksheet. Вы получите команды DDL создания таблицы employees2 и зависимых от нее объектов.



26. Результаты различны. В postgres insert вставил поле с пробелом, в oracle с NULL, поэтому результаты выборок различны.

27. Результаты также различны, хотя теперь в oracle было вставлено поле с пробелом.
Не стоит использовать тип данных CHAR в приложениях.

28. Длина индексируемого значения не должна превышать примерно 70% размера блока табличного пространства БД Oracle, в котором создаётся индекс. Размер блока БД Oracle по умолчанию 8192 байт.

Практическое задание 4

Решения приведены в конце задания.

1. Нужен отчет, в котором показываются фамилии и оклады служащих, получающих более 14000.

LAST_NAME	SALARY
King	24000
Kochhar	17000
De Haan	17000

2.Создайте запрос для вывода фамилии и номера отдела служащего под номером 130

LAST_NAME	DEPARTMENT_ID
Atkinson	50

3. Выведите фамилии и оклады всех служащих, чей оклад **не** входит в диапазон от 2000 до 16000

LAST_NAME	SALARY
King	24000
Kochhar	17000
De Haan	17000

4.Выведите фамилии и даты начала работы всех служащих, с фамилиями Matos и Taylor. Отсортируйте данные в порядке возрастания даты найма.

LAST_NAME	HIRE_DATE
Taylor	24-JAN-06
Matos	15-MAR-06
Taylor	24-MAR-06

5. Выведите фамилию и номер отдела всех служащих из отделов 20 и 70. Отсортируйте данные по фамилиям в убывающем порядке так, чтобы сотрудники 70 отдела выводились первыми.

LAST_NAME	DEPARTMENT_ID
Baer	70
Hartstein	20
Fay	20

6. Выведите фамилию и зарплату сотрудников отделов 20 и 50, у каоторых зарплата от 6000 до 8000 **включительно**. Назовите столбцы Emp и Sal

Emp	Sal
Weiss	8000
Kaufling	7900
Vollman	6500
Fay	6000

Обратите внимание, что границы диапазона включены в выборку

7. Создайте отчет, содержащий фамилии всех служащих, не имеющих менеджера.

LAST_NAME
King

8. Создайте отчет для вывода фамилий и комиссионных всех служащих, зарабатывающих комиссионные. Отсортируйте данные в порядке убывания окладов, а потом комиссионных.

LAST_NAME	COMMISSION_PCT
Russell	0.4
Partners	0.3
Errazuriz	0.3
Ozer	0.25
Cambrault	0.3
Abel	0.3
Vishney	0.25

...

9. Выведите все фамилии служащих, в которых третья буква **"a"**

LAST_NAME
Grant
Grant
Whalen

10. Выведите все фамилии служащих, в которых есть буквы “r” и “z”

LAST_NAME
Lorentz
Errazuriz
Ozer

11.Выберите фамилии, должности и оклады всех служащих, не работающих торговыми представителями (`SA_REP`) и рекламными представилеями (`PR_REP`) и у которых оклад 6000 или 9000 или 12000.

LAST_NAME	JOB_ID	SALARY
Hunold	IT_PROG	9000
Ernst	IT_PROG	6000
Faviet	FI_ACCOUNT	9000
Errazuriz	SA_MAN	12000
Fay	MK_REP	6000

12. Нужно получить фамилии и комиссионные всех служащих, у которых комиссионные (commission_pct) составляют 35% (0.35)

LAST_NAME	COMMISSION_PCT
King	0.35
Sully	0.35
McEwen	0.35

13. Выберите идентификаторы должностей (job_id), в которых первый символ "A", а третий символ подчеркивание.

JOB_ID
AC_MGR
AC_ACCOUNT
AD_ASST
AD_VP
AD_PRES

14. Выполните команду:

```
SELECT * FROM employees
WHERE last_name='King'
```

```
ORDER BY manager_id;
```

Строка с пустым значением в столбце manager_id выводится последней. Сделайте так, чтобы при сортировке она вводилась первой.

Следующие пункты этого задания (15-26) опциональны, можно их не выполнять, а просмотреть.

15. Выполните в соединении **postgres** команды (можно выделить и склеить текст):

```
DROP TABLE TEST;
CREATE TABLE TEST (N VARCHAR(2));
INSERT INTO TEST VALUES('E');
INSERT INTO TEST VALUES('ж');
INSERT INTO TEST VALUES('ё');
INSERT INTO TEST VALUES('e');
INSERT INTO TEST VALUES('a');
INSERT INTO TEST VALUES('Ж');
INSERT INTO TEST VALUES('Ё');
CREATE INDEX TEST1 ON TEST(N);
SET enable_seqscan TO off;
SELECT * FROM TEST WHERE N>'A'  ORDER BY N;
EXPLAIN SELECT * FROM TEST WHERE N>'A' ORDER BY N;
SET enable_seqscan TO on;
DROP TABLE TEST;
```

Вы получите результат:

```
QUERY PLAN
---------------------
Index Only Scan using test1 on test  (cost=0.13..8.17 rows=2 width=12)
  Index Cond: (n > 'A'::text)
```

Запрос выдал данные используя только индекс test1.

Выдались данные:

```
e
E
ё
Ё
ж
Ж

 6 rows selected
```

Буква "a" не выдалась. Это значит, что оператор > считает что "A">"a".

Буквы верхнего регистра идут после букв нижнего регистра.

16. Выполните в соединении **oracle** команды (можно выделить и склеить текст):

```
DROP TABLE TEST;
ALTER SESSION SET NLS_SORT='BINARY';
ALTER SESSION SET NLS_COMP='BINARY';
CREATE TABLE TEST (N VARCHAR(2));
INSERT INTO TEST VALUES('E');
INSERT INTO TEST VALUES('ж');
INSERT INTO TEST VALUES('ё');
INSERT INTO TEST VALUES('e');
INSERT INTO TEST VALUES('a');
INSERT INTO TEST VALUES('Ж');
INSERT INTO TEST VALUES('Ё');
CREATE INDEX TEST1 ON TEST(N);
ANALYZE TABLE TEST COMPUTE STATISTICS;
SELECT * FROM TEST WHERE N>'A'  ORDER BY N;
```

Вы получите:

```
E
Ж
a
e
ж
ё

 6 rows selected
```

Сначала идут буквы верхнего регистра.

Проблемы: **Буква "ё" идёт в конце. Буква "Ё" не выдалась, она идёт раньше всех букв.**

17. Выделите последнюю команду SELECT * FROM TEST WHERE N>'A' ORDER BY N; и нажмите F10. Вы увидите план выполнения команды

OPERATION	OBJECT_NAME	CARDINALITY	COST
SELECT STATEMENT		7	1
INDEX (RANGE SCAN)	TEST1	7	1
Access Predicates			
N>'A'			

В плане видно, что используется индекс TEST1. Оцениваемая кардинальность это число возможных значений - 7 строк.

18. Выполните команду:

```
ALTER SESSION SET NLS_SORT='RUSSIAN';
```

а за ней **выполните**

```
SELECT * FROM TEST WHERE N>'A'  ORDER BY N;
```

Примечание: вы можете выделять команды в окне Worksheet и нажимая F5 будут выполняться только эти команды. Тем самым вы можете не стирать историю команд, а выполяднть только те команды, которые хотите.

Вы получите результат:

```
a
Е
е
ё
Ж
ж

 6 rows selected
```

Буква "Ё" отсутствует.

19. Посмотрите план выполнения (выделить команду SELECT...; и нажать F10)

OPERATION	OBJECT_NAME	CARDINALITY	COST
SELECT STATEMENT		7	2
SORT (ORDER BY)		7	2
INDEX (RANGE SCAN)	TEST1	7	1
Access Predicates			

В плане видно, что индекс TEST1 продолжает использоваться. Оцениваемая "стоимость" выполнения возросла.

20. Выполните команду:

```
ALTER SESSION SET NLS_COMP='LINGUISTIC';
```

Выполните SELECT * FROM TEST WHERE N>'A' ORDER BY N;

```
a
Е
е
Ё
ё
Ж
ж

 7 rows selected
```

Все буквы выдались. При выборке буквы верхнего регистра идут до букв нижнего регистра, что соответствует правилам русского языка.

21. Посмотрите план выполнения (выделить команду SELECT и нажать F10)

OPERATION	OBJECT_NAME	CARDINALITY	COST
SELECT STATEMENT		1	4
SORT (ORDER BY)		1	4
TABLE ACCESS (FULL)	TEST	1	3
Filter Predicates			

Индекс перестал использоваться, таблица просматривается полностью. Если бы в таблице было много строк, то это снизило производительность. Оцениваемая стоимость выполнения также возросла. В простом случае выборки из одной таблицы при полном сканировании кардинальность роли не играет.

22. Создайте индекс, основанный на функции:

```
CREATE INDEX TEST2 ON TEST(NLSSORT(N,'NLS_SORT=RUSSIAN'));
ANALYZE TABLE TEST COMPUTE STATISTICS;
```

Вторая команда обновляет статистику по таблице, которая влияет на оценки кардинальности и стоимости выполнения, которые отображаются в плане выпонения как CARDINAITY и COST.

23. Посмотрите план выполнения

OPERATION	OBJECT_NAME	CARDINALITY	COST
SELECT STATEMENT		7	2
TABLE ACCESS (BY INDEX ROWID)	TEST	7	2
INDEX (RANGE SCAN)	TEST2	7	1
Access Predicates			
NLSSORT(N,'nls_sort="RUSSIAN"')>HEXTORAW('8C000100')			

Стал использоваться индекс и оценка стоимости выполнения как у обычного индекса.
Вместо задания своств сессии можно использовать оператор COLLATE

```
SELECT * FROM TEST WHERE N>'A' COLLATE RUSSIAN ORDER BY N COLLATE RUSSIAN;
```

результат и будет тот же, индекс TEST2 будет использоваться.
Первый COLLATE заменяет NLS_COMP, второй NLS_SORT.
План отразит использование COLLATE для вычисления предиката:

OPERATION	OB...	CARDI...	C...
SELECT STATEMENT		7	2
TABLE ACCESS (BY INDEX ROWID)	TEST	7	2
INDEX (RANGE SCAN)	TEST2	7	1
Access Predicates			
NLSSORT(N,'nls_sort="RUSSIAN"')>NLSSORT('A' COLLATE RUSSIAN,'nls_sort="RUSSIAN"			

Поскольку команды изменения свойств сессии специфичны для БД и выдаются отдельными командами, то в приложениях удобнее включать стандартный оператор COLLATE.
Не нужно забывать, что его нужно ставить после всех операций, на которые может оказать влияние сортировка и сравнение. Например, если не указать COLLATE после сравнения, то выдастся не 7, а 6 строк:

```
SELECT * FROM TEST WHERE N>'A'    ORDER BY N COLLATE RUSSIAN;
а
Е
е
ё
Ж
ж

 6 rows selected
```

что будет эквивалентно установке NLS_SORT=RUSSIAN NLS_COMP=BINARY, как в пункте 18 этого задания. План выполнения будет аналогичным geyrne 19 задания - будет использоваться индекс TEST1.

24. Дайте команду:

```
ALTER SESSION SET NLS_SORT='GENERIC_M';
```

и выполните запрос снова (выделить SELECT и нажать F5). Вы получите результат:

```
е
Е
ё
Ё
ж
Ж

 6 rows selected
```

Выдался такой же результат, как и в БД postgres.

25. Посмотрите план выполнения

OPERATION	OBJECT_NAME	CARDINALITY	COST
SELECT STATEMENT		1	4
SORT (ORDER BY)		1	4
TABLE ACCESS (FULL)	TEST	1	3
Filter Predicates			
NLSSORT(N,'nls_sort=''GENERIC_M''')>HEXTORAW('03130000020003')			

Индексы не используются. Для того, чтобы индексы использовались нужно создать новый индекс основанный на функции CREATE INDEX TEST2 ON TEST(NLSSORT(N,'NLS_SORT=RUSSIAN'));

Примечание: для сравнений без учёта регистра, можно использовать суффикс _CI. Например: ALTER SESSION SET NLS_SORT='GENERIC_M_**CI**'; или BINARY_CI. В любом случае, обычные индексы использоваться не будут. Даже для BINARY_CI нужен индекс, основанный на функции.

26. Удалите таблицу TEST и верните настроки сессии к значениям по умолчанию:

```
DROP TABLE TEST;
ALTER SESSION SET NLS_SORT='BINARY';
ALTER SESSION SET NLS_COMP='BINARY';
```

Практическое задание 4 решения

1. Нужен отчет, в котором показываются фамилии и оклады служащих, получающих более 14000.

Ответ:

```
SELECT  last_name, salary FROM employees
WHERE   salary > 14000;
```

2.Создайте запрос для вывода фамилии и номера отдела служащего под номером 130

Ответ:

```
SELECT  last_name, department_id FROM employees
WHERE   employee_id = 130;
```

3. Выведите фамилии и оклады всех служащих, чей оклад **не** входит в диапазон от 2000 до 16000

Ответ:

```
SELECT  last_name, salary FROM employees
WHERE   salary NOT BETWEEN 2000 AND 16000;
```

4.Выведите фамилии и даты начала работы всех служащих, с фамилиями Matos и Taylor. Отсортируйте данные в порядке возрастания даты найма.

Ответ:

```
SELECT   last_name, hire_date
FROM     employees
WHERE    last_name IN ('Matos', 'Taylor')
ORDER BY hire_date;
```

5. Выведите фамилию и номер отдела всех служащих из отделов 20 и 70. Отсортируйте данные по фамилиям в убывающем порядке так, чтобы сотрудники 70 отдела выводились первыми.

Ответ:

```
SELECT   last_name, department_id
FROM     employees
WHERE    department_id IN (20, 70)
ORDER BY department_id DESC, last_name DESC;
```

6. Выведите фамилию и зарплату сотрудников отделов 20 и 50, у каоторых зарплата от 6000 до 8000 **включительно**. Назовите столбцы Emp и Sal

Ответ:

```
SELECT   last_name "Emp", salary "Sal"
FROM     employees
WHERE    salary  BETWEEN 6000 AND 8000
AND      department_id IN (20, 50);
```

Обратите внимание, что границы диапазона включены в выборку

7. Создайте отчет, содержащий фамилии всех служащих, не имеющих менеджера.

Ответ:

```
SELECT   last_name
FROM     employees
WHERE    manager_id IS NULL;
```

8. Создайте отчет для вывода фамилий и комиссионных всех служащих, зарабатывающих комиссионные. Отсортируйте данные в порядке убывания окладов, а потом комиссионных.

Ответ:

```
SELECT   last_name, commission_pct
FROM     employees
WHERE    commission_pct IS NOT NULL
ORDER BY salary DESC, commission_pct DESC;
```

9. Выведите все фамилии служащих, в которых третья буква **"a"**

Ответ:

```
SELECT   last_name FROM employees
WHERE    last_name LIKE '__a%';
```

10. Выведите все фамилии служащих, в которых есть буквы “r” и “z”

Ответ:

```
SELECT   last_name FROM employees
```

```
WHERE    last_name LIKE '%r%' AND last_name LIKE '%z%';
```

11.Выберите фамилии, должности и оклады всех служащих, не работающих торговыми представителями (SA_REP) и рекламными представилеями (PR_REP) и у которых оклад 6000 или 9000 или 12000.
Ответ:

```
SELECT   last_name, job_id, salary
FROM     employees
WHERE    job_id NOT IN ('SA_REP', 'PR_REP') AND salary IN (6000, 9000, 12000);
```

12. Нужно получить фамилии и комиссионные всех служащих, у которых комиссионные (commission_pct) составляют 35% (0.35)
Ответ:

```
SELECT   last_name, commission_pct
FROM     employees
WHERE    commission_pct = .35;
```

13. Выберите идентификаторы должностей (job_id), в которых первый символ "A", а третий символ подчеркивание.
Ответ:

```
SELECT DISTINCT job_id
FROM employees WHERE job_id LIKE 'A_\_%' ESCAPE '\';
```

14. В поле manager_id пустое значение выводится последним. Сделайте так, чтобы при сортировке строка с пустым менеджером выводилась первой:

```
SELECT * FROM employees
WHERE last_name='King'
ORDER BY manager_id NULLS FIRST;
```

.

Практическое задание 5

1. Следующая команда выводит текущую дату в базе данных Oracle

```
SELECT sysdate FROM dual;
```

Выведите текущую дату в postgres

```
SELECT now()::date;
```

Функции sysdate и now() унаследованные и нестандартны. Суффикс ::date это унаследованный и нестандартный синтаксис приведения к типу date в БД postgres.

2. Замените эти функции на стандартную функцию и выполните запросы повторно в обоих соединениях. Нужны ли () после названия функции? Можно ли их добавлять или убирать?

3. Требуется отчет, в котором приводится фамилия служащего, его оклад и новый оклад, повышенный на 15% и округленный до целого. Столбец в отчете, содержащий новый оклад, должен иметь имя New.

LAST_NAME	SALARY	New
King	24000	27600
Kochhar	17000	19550
De Haan	17000	19550
Hunold	9000	10350
Ernst	6000	6900
Austin	4800	5520
Pataballa	4800	5520
Lorentz	4200	4830

...

4. Добавьте еще один столбец, который будет содержать результат вычитания старого оклада из нового. Назовите столбец `Increase`. Выполните запрос.

LAST_NAME	SALARY	New	Increase
King	24000	27600	3600
Kochhar	17000	19550	2550
De Haan	17000	19550	2550
Hunold	9000	10350	1350
Ernst	6000	6900	900
Austin	4800	5520	720
Pataballa	4800	5520	720
Lorentz	4200	4830	630
Greenberg	12008	13809	1801

...

5. Выведите фамилии служащих (первая буква каждой фамилии должна быть заглавной, а остальные - строчными) и длину каждой фамилии для тех служащих, фамилия которых начинается с символа *J*, *A* или *M*. Присвойте соответствующие заголовки столбцам. Отсортируйте результат по фамилиям служащих.

Name	Length
Abel	4
Ande	4
Atkinson	8
Austin	6
Johnson	7
Jones	5
Mallin	6
Markle	6

...

6. Нужны сведения о продолжительности работы служащих. Для каждого служащего выведите фамилию и вычислите количество полных месяцев со дня найма до настоящего времени. Назовите столбец `MONTHS_WORKED`. Результаты отсортируйте по количеству отработанных месяцев.
Колличество дней в месяцах различна и нужно использовать специальные функции, которые отсутствуютв стандарте SQL.

В **БД Oracle** можно использовать функцию MONTHS_BETWEEN.

LAST_NAME	MONTHS_WORKED
Banda	107
Kumar	107
Ande	108
Markle	109
Lee	109
Philtanker	110
Zlotkey	110
Marvins	110
Geoni	110

...
В **postgres** есть функция AGE https://postgrespro.ru/docs/postgrespro/9.6/functions-datetime
Нужно из её результата функцией EXTRACT получить количество месяцев.

last_name	months_worked
Kumar	107.0
Banda	107.0
Ande	108.0
Lee	109.0
Markle	109.0
Marvins	110.0
Geoni	110.0
Philtanker	110.0
Zlotkey	110.0

...

7. Получите по каждому служащему отчет в следующем виде:
"фамилия" зарабатывает "оклад" , а хочет   "утроенный оклад". Назовите столбец `Dream`.

Dream
King зарабатывает $24,000.00 , а хочет $72,000.00
Kochhar зарабатывает $17,000.00 , а хочет $51,000.00
De Haan зарабатывает $17,000.00 , а хочет $51,000.00
Hunold зарабатывает $9,000.00 , а хочет $27,000.00
Ernst зарабатывает $6,000.00 , а хочет $18,000.00
Austin зарабатывает $4,800.00 , а хочет $14,400.00
Pataballa зарабатывает $4,800.00 , а хочет $14,400.00
Lorentz зарабатывает $4,200.00 , а хочет $12,600.00

...

8. Следующий пример работает в **БД Oracle**, но не работает в **postgres**, так как LPAD рабтает со строками, а salary число.

```
SELECT last_name,
       LPAD(salary, 15, '$') SALARY
FROM   employees;
```

Измените запрос так, чтобы запрос работал в БД postgres

```
SELECT last_name,
       LPAD(CAST(salary as varchar), 15, '$') SALARY
FROM   employees;
```

Измените вышеприведённый запрос так, чтобы запрос работал и в **БД Oracle** и в **БД postgres**

9. В БД Oracle по каждому служащему выводится фамилия, дата найма и день недели, когда он был нанят на работу. Результатат сортруется по дням недели, начиная с понедельника.

```
SELECT last_name, hire_date,
       TO_CHAR(hire_date, 'DAY') DAY
FROM employees
ORDER BY TO_CHAR(hire_date - 1, 'd'), last_name;
```

а) Поправьте вышеприведённый запрос так, чтобы он работал в postgres

LAST_NAME	HIRE_DATE	DAY
Ande	24-MAR-08	MONDAY
Banda	21-APR-08	MONDAY
Cambrault	15-OCT-07	MONDAY
Ernst	21-MAY-07	MONDAY
Greene	19-MAR-07	MONDAY
Kumar	21-APR-08	MONDAY
Ladwig	14-JUL-03	MONDAY
Mallin	14-JUN-04	MONDAY
Vollman	10-OCT-05	MONDAY
Walsh	24-APR-06	MONDAY
Abel	11-MAY-04	TUESDAY

...

Следующий запрос выдаёт корректно данные в postgres, но не в БД Oracle

```
SELECT last_name, hire_date,
       TO_CHAR(hire_date, 'DAY') "DAY"
FROM employees
ORDER BY TO_CHAR(hire_date, 'ID'), last_name;
```

Параметры форматирования https://postgrespro.ru/docs/postgrespro/9.6/functions-formatting
'ID' номер дня недели, считая с понедельника (1) до воскресенья (7)
'D' номер дня недели, считая с воскресенья (1) до субботы (7)

б) Нужно вывести фамилию и в отдельном столбце год, название месяца и дату приема сотрудника в таком формате '`Friday    , August    , 2002`'. Отсортируйте результаты по возрастанию даты для каждого года.
Используйте функцию TO_CHAR для форматирования hire_date.
Описание функции https://postgrespro.ru/docs/postgrespro/9.6/functions-formatting

LAST_NAME	HD
Faviet	Friday , August , 2002
Colmenares	Friday , August , 2007
Popp	Friday , December , 2007
Smith	Friday , February , 2007
King	Friday , January , 2004
Johnson	Friday , January , 2008
Baer	Friday , June , 2002
Higgins	Friday , June , 2002
Gietz	Friday , June , 2002

...

в) В результате видны пробелы после дня недели и месяца. **Уберите проелы, используя приставки "fm".**

LAST_NAME	HD
Faviet	Friday, August, 2002
Colmenares	Friday, August, 2007
Popp	Friday, December, 2007
Smith	Friday, February, 2007
King	Friday, January, 2004
Johnson	Friday, January, 2008
Baer	Friday, June, 2002
Higgins	Friday, June, 2002
Gietz	Friday, June, 2002

...

Поскольку поведение fm в БД Oracle и БД postgres различно, придётся использовать различные строки форматирования
https://postgrespro.ru/docs/postgrespro/9.6/functions-formatting
"fm" подавляет дополняющие пробелы и нули справа, которые добавляются по умолчанию. В БД postgres модификатор "fm" действует только на метасимволы, перед которым стоит "fm", тогда как в Oracle "fm" по умолчанию действует на все метасимволы до конца, но можно ещё раз вставить "fm" и он в БД Oracle переключит убирание пробелов.

10. В БД Oracle для вывода фамилии и суммы комиссионных каждого служащего используется следующий запрос. Если служащий не зарабатывает комиссионных, выдается в столбце “No Commission.”

```
SELECT last_name,
 NVL(TO_CHAR(commission_pct), 'No Commission') COMM
FROM employees;
```

а) Препишите запрос так, чтобы он работал в postgres
б) уберите пробелы и нули, используя fm

11. Выполните запрос в соединениях postgres и oracle

```
SELECT last_name, TRIM(LEADING 'H' FROM last_name) AS trimmed
FROM employees WHERE last_name LIKE 'H%';
```

Команды выдали одинаковые результаты в обеих БД
12. В БД Oracle следующий запрос отображает первые восьмь букв фамилий сотрудников и их заработной платы в виде гистограммы, состоящей из звездочек. Каждая звездочка означает 1000. Строки отсортированы по заработной плате в убывающем порядке. Результат выведен одним столбцом, озаглавленным как `GRADE`.

```
SELECT rpad(last_name, 8)||' '||
       rpad(' ', salary/1000+1, '*')
       GRADE
FROM  employees
ORDER BY salary DESC, last_name ASC;
```

	GRADE
4	Russell ***************
5	Partners **************
6	Hartstei **************
7	Greenber *************
8	Higgins *************
9	Errazuri *************
10	Ozer ************
11	Abel ************
12	Cambraul ************
13	Raphaely ************
14	Vishney **********

Запрос переписан так, чтобы он выполнялся в БД postgres:

```
SELECT rpad(last_name, 8,' ')||' '||
       rpad(' ', cast(salary/1000+1 as INTEGER), '*')
       GRADE
FROM  employees
ORDER BY salary DESC, last_name ASC;
```

Однако, запросы выдаёт другое количество звездочек, чем исходный. Это значит, что запрос переписан некорректно и при реальной миграции кода это привело бы к ошибке.

	grade		
4	Russell	**************	
5	Partners	**************	<
6	Hartstei	*************	
7	Greenber	************	
8	Higgins	************	
9	Errazuri	************	
10	Ozer	************	<
11	Abel	***********	
12	Cambraul	***********	
13	Raphaely	***********	
14	Vishney	***********	<

Выясните, в чём причина ошибки и поправьте последний запрос так, чтобы он выдавал данные как исходный.

13. Покажите сотрудников, которые были приняты на работу в первой половине месяца (до 16 числа месяца).
Пример для БД Oracle:

```
SELECT last_name, hire_date
FROM   employees
WHERE  TO_CHAR(hire_date, 'DD') < 16;
```

Перепишите запрос, чтобы он выполнялся в БД postgres.

14. Создайте отчет, в котором покажите фамилии, оклады сотрудников и сколько тысяч долларов они зарабатывают.
Пример для БД Oracle:

```
SELECT last_name, salary, TRUNC(salary, -3)/1000  Thousands
FROM   employees;
```

Перепишите запрос, чтобы он выполнялся в БД postgres.

15. В БД Oracle есть функция `DECODE`. С ее помощью можно отобразить должности сотрудника и уровни должности (grade). Разряд каждого типа должности `JOB_ID` приведен в таблице:

Должность	***Разряд***
AD_PRES	A
ST_MAN	B
IT_PROG	C
SA_REP	D
ST_CLERK	E
Другая	0

Выполните команду в соединении oracle:

```
SELECT job_id, decode (job_id,
                       'ST_CLERK',   'E',
                       'SA_REP',     'D',
                       'IT_PROG',    'C',
                       'ST_MAN',     'B',
                       'AD_PRES',    'A',
                       '0') GRADE
FROM employees;
```

Перепишите команду используя стандартное выражение `CASE` и выполните в соединении postgres.

16. Выполните команду в соединении oracle:

```
SELECT CAST(1 AS CHAR) + CAST(1 AS CHAR) plus FROM DUAL;
```

Команда выдала 2.
Выполните команду в соединении postgres

```
SELECT CAST(1 AS CHAR) + CAST(1 AS CHAR);
```

Команда выполнилась с ошибкой - строки не складываются и автоматического приведения сроки к числу нет.

17. Создайте в соединении postgres приведение типов, которое может использоваться неявно:

```
CREATE CAST (CHAR AS NUMERIC) WITH INOUT AS IMPLICIT;
```

Выполните предыдущую команду. Теперь она выполнится и выдаст 2.

18. Удалите объект CAST:

```
DROP CAST (CHAR AS NUMERIC);
```

19. Выполните следующую команду в соединениях oracle и postgres:

```
select COUNT(hire_date) from employees where hire_date < '01-JAN-06';
```

убедитесь, что результат одинаков и выдалось 53 строки.

20. Выполните следующую команду в соединениях oracle и postgres:

```
select COUNT(hire_date) from employees where hire_date < '2006-01-01';
```

В соединении postgres команда вернула 53 строки, в соединении oracle ошибку. БД Oracle по умолчанию использует формат дат 'DD-MON-RR'.

21. Перепишите команду с использованием явного преобразования типов TO_DATE, чтобы команда одинаково выполнялась в обеих БД.

Практическое задание 5 решения

2. Замените эти функции на стандартную функцию `current_date` и выполните запросы повторно. Она выдаёт одинаковый результат в типе date.

```
SELECT current_date FROM dual;
```

Обратите внимание, что current_date() выдаст ошибку.

```
SELECT current_date();
```

Название функции по стандарту SQL current_date без круглых скобок.

Выполните в соединении postgres:

```
SELECT version;
ОШИБКА: столбец "version" не существует
```

Выполните

```
SELECT version();
```

эта команда выполнилась - функция вернула значение. Эта функция без аргументов, она не описана в стандартах SQL и присутствует в БД postgres, в БД Oracle отсутствует.

Функции вызываются так, как описано в стандартах SQL.

3. Требуется отчет, в котором приводится фамилия служащего, его оклад и новый оклад, повышенный на 15% и округленный до целого. Столбец в отчете, содержащий новый оклад, должен иметь имя New.

```
SELECT last_name, salary,
        ROUND(salary * 1.15, 0) "New"
FROM employees;
```

4. Добавьте еще один столбец, который будет содержать результат вычитания старого оклада из нового. Назовите столбец `Increase`. Выполните запрос.

```
SELECT last_name, salary,
        ROUND(salary * 1.15, 0) "New",
        ROUND(salary * 1.15, 0) - salary "Increase"
FROM employees;
```

5. Выведите фамилии служащих (первая буква каждой фамилии должна быть заглавной, а остальные - строчными) и длину каждой фамилии для тех служащих, фамилия которых начинается с символа *J*, *A* или *M*. Присвойте соответствующие заголовки столбцам. Отсортируйте результат по фамилиям служащих.

```
SELECT  INITCAP(last_name) "Name",
        LENGTH(last_name) "Length"
FROM employees
WHERE   last_name LIKE 'J%'
OR      last_name LIKE 'M%'
OR      last_name LIKE 'A%'
ORDER BY last_name;
```

6. Нужны сведения о продолжительности работы служащих. Для каждого служащего выведите фамилию и вычислите количество полных месяцев со дня найма до настоящего времени. Назовите столбец `MONTHS_WORKED`. Результаты отсортируйте по количеству отработанных месяцев.
Колличество дней в месяцах различна и нужно использовать специальные функции, которые отсутствуютв стандарте SQL.
В **БД Oracle** можно использовать функцию MONTHS_BETWEEN:

```
SELECT last_name, TRUNC(MONTHS_BETWEEN(CURRENT_DATE, hire_date)) MONTHS_WORKED
FROM   employees
ORDER BY months_worked;
```

В **postgres** есть функция AGE https://postgrespro.ru/docs/postgrespro/9.6/functions-datetime

```
SELECT last_name, EXTRACT(YEAR FROM AGE(CURRENT_DATE, HIRE_DATE))*12 +
        EXTRACT(MONTH FROM AGE(CURRENT_DATE, HIRE_DATE)) MONTHS_WORKED
FROM   employees
ORDER BY months_worked;
```

7. Получите по каждому служащему отчет в следующем виде:
"фамилия" зарабатывает "оклад" , а хочет "утроенный оклад". Назовите столбец `Dream`.

```
SELECT  last_name || ' зарабатывает '
        || TO_CHAR(salary, 'fm$99,999.00')
        || ' , а хочет '
```

```
        || TO_CHAR(salary * 3, 'fm$99,999.00') "Dream"
FROM    employees;
```

8. Измените запрос так, чтобы запрос работал в БД postgres

```
SELECT last_name,
       LPAD(CAST(salary as varchar), 15, '$') SALARY
FROM   employees;
```

Измените запрос так, чтобы запрос работал и в **БД Oracle** и в **БД postgres**

```
SELECT last_name,
       LPAD(CAST(salary as varchar(15)), 15, '$') SALARY
FROM   employees;
```

9.

а) Поправьте запрос так, чтобы он работал в postgres

```
SELECT last_name, hire_date,
       TO_CHAR(hire_date, 'DAY') "DAY"
FROM     employees
ORDER BY TO_CHAR(hire_date - 1, 'd'), last_name;
```

б) Нужно вывести год, название месяца и дату приема сотрудника в таком формате '`2005 December  15`'
Отсортируйте результаты по возрастанию даты для каждого года.

```
SELECT last_name, TO_CHAR(hire_date, 'YYYY Month DD') HD
FROM   employees
ORDER BY HD;
```

в) В результате видны пробелы между годом и месяцем. Уберите их, используя приставку "fm".

```
SELECT last_name, TO_CHAR(hire_date, 'fmDay, Month, YYYY') HD
FROM   employees
ORDER BY HD;
```

Однако, такой запрос убирает пробелы только в БД Oracle.
Для БД postgres:

```
SELECT last_name, TO_CHAR(hire_date, 'fmDay, fmMonth, fmYYYY') HD
FROM   employees
ORDER BY HD;
```

Причина описана в https://postgrespro.ru/docs/postgrespro/9.6/functions-formatting
"fm" подавляет дополняющие пробелы и нули справа, которые добавляются по умолчанию. В БД postgres модификатор "fm" действует только на метасимволы, перед которым стоит "fm", тогда как в Oracle "fm" по умолчанию действует на все метасимволы до конца, но можно ещё раз вставить "fm" и он в БД Oracle переключит убирание пробелов.
Пример:
'**fm**Day, Month, YYYY' в postgres действует только на Day, в Oracle на всё.
'**fm**Day, **fm**Month, YYYY' в postgres действуют на Day Month, в Oracle только на Day
'**fm**Day, **fm**Month, **fm**YYYY' в postgres действуют на Day Month YYYY; в Oracle действуют на Day и YYYY.

10. Препишите запрос так, чтобы он работал в postgres

```
SELECT last_name,
 COALESCE(TO_CHAR(commission_pct,'999999.99'), 'No Commission') COMM
FROM employees;
```

б) Нужно убрать пробелы и нули:

```
SELECT last_name,
 COALESCE(TO_CHAR(commission_pct,'fm999999.99'), 'No Commission') COMM
FROM employees;
```

12. Правильный запрос:

```
SELECT rpad(last_name, 8,' ')||' '||
       rpad(' ', cast(trunc(salary/1000)+1 AS INTEGER), '*')
       GRADE
FROM  employees
ORDER BY salary DESC, last_name ASC;
```

13. Перепишите запрос, чтобы он выполнялся в БД postgres.

```
SELECT last_name, hire_date
FROM   employees
```

```
WHERE  CAST(TO_CHAR(hire_date, 'DD') AS INTEGER) < 16;
```

14. Перепишите запрос, чтобы он выполнялся в БД postgres.

```
SELECT last_name, salary, ROUND(TRUNC(salary, -3)/1000)  Thousands
FROM   employees;
```

15. Перепишите команду используя стандартное выражение `CASE` и выполните в соединении postgres:

```
SELECT job_id, CASE job_id
               WHEN 'ST_CLERK' THEN 'E'
               WHEN 'SA_REP'   THEN 'D'
               WHEN 'IT_PROG'  THEN 'C'
               WHEN 'ST_MAN'   THEN 'B'
               WHEN 'AD_PRES'  THEN 'A'
               ELSE '0'  END  GRADE
FROM employees;
```

21. Перепишите команду с использованием явного преобразования типов TO_DATE, чтобы команда одинаково выполнялась в обеих БД.

```
SELECT COUNT(hire_date)
 FROM employees
 WHERE hire_date < TO_DATE('2006-01-01','YYYY-MM-DD');
```

Практическое задание 6

1. Выведите город и страну, в которой находится город. Используйте таблицы `LOCATIONS` и `COUNTRIES`. Для получения результатов используйте натуральное соединение.

CITY	COUNTRY_ID
Sydney	AU
Sao Paulo	BR
Toronto	CA
Whitehorse	CA
Geneva	CH
Bern	CH
Beijing	CN
Munich	DE

...

2. Выведите название отдела, город и страну где находится отдел. Используйте в дополнение к соединению двух таблиц соединение с третьей DEPARTMENTS. Вы должны соединить три таблицы.

DEPARTMENT_NAME	CITY	COUNTRY_ID
Marketing	Toronto	CA
Public Relations	Munich	DE
Human Resources	London	UK
Sales	Oxford	UK
IT	Southlake	US
Shipping	South San Francisco	US
Administration	Seattle	US
Purchasing	Seattle	US

...

3. Напишите запрос для вывода фамилии и названия отдела для всех служащих. Укажите явно соединение по столбцу department_id

LAST_NAME	DEPARTMENT_NAME
King	Executive
Kochhar	Executive
De Haan	Executive
Hunold	IT
Ernst	IT
Austin	IT
Pataballa	IT
Lorentz	IT

...

4. Напишите запрос для вывода фамилии и названия отдела всех служащих, работающих в городе Toronto. Вам нужно соединить три таблицы EMPLOYEES, DEPARTMENTS, LOCATIONS

LAST_NAME	DEPARTMENT_NAME
Hartstein	Marketing
Fay	Marketing

...

5. Выведите фамилии сотрудников и номера их менеджеров.

LAST_NAME	LAST_NAME_1
Hartstein	King
Zlotkey	King
Cambrault	King
Errazuriz	King
Partners	King
Russell	King
Mourgos	King
Vollman	King
Kaufling	King

10**6** строк
...

6. Однако, в результате отсутствует сотрудник King, у которого нет менеджера. Измените запрос так, чтобы получить фамилии всех служащих, включая King, который не имеет менеджера. Упорядочьте результат по возрастанию номера служащего.

LAST_NAME	LAST_NAME_1
King	(null)
Kochhar	King
De Haan	King
Hunold	De Haan
Ernst	Hunold
Austin	Hunold
Pataballa	Hunold
Lorentz	Hunold

10**7** строк
...

7. Напишите SELECT, в котором показываются номер отдела, фамилия служащего и фамилии всех служащих, работающих в одном отделе с данным служащим. Дайте столбцам имена dept, employee, colleague.

DEPT	EMPLOYEE	COLLEAGUE
90	De Haan	King
90	Kochhar	King
90	De Haan	Kochhar
90	King	Kochhar
90	Kochhar	De Haan
90	King	De Haan
60	Lorentz	Hunold
60	Pataballa	Hunold

...

8. Добавьте в конец запроса фразу `ORDER BY dept, 2 , colleague;`
Проверьте, что команда выполнилась успешно.

DEPT	EMPLOYEE	COLLEAGUE
20	Fay	Hartstein
20	Hartstein	Fay
30	Baida	Colmenares
30	Baida	Himuro
30	Baida	Khoo
30	Baida	Raphaely
30	Baida	Tobias
30	Colmenares	Baida

...

9. Нужен отчет о категориях должностей и окладах. Сначала посмотрите структуру таблицы `JOB_GRADES`. В БД Oracle **DESC JOB_GRADES;** В psql **`\d job_grades;`** Затем создайте запрос для вывода фамилии, должности, названия отдела, оклада и категории (`GRADE_LEVEL`) всех служащих.

LAST_NAME	JOB_ID	DEPARTMENT_NAME	SALARY	GRADE_LEVEL
King	AD_PRES	Executive	24000	E
Kochhar	AD_VP	Executive	17000	E
De Haan	AD_VP	Executive	17000	E
Russell	SA_MAN	Sales	14000	D
Partners	SA_MAN	Sales	13500	D
Hartstein	MK_MAN	Marketing	13000	D
Greenberg	FI_MGR	Finance	12008	D
Higgins	AC_MGR	Accounting	12008	D
Errazuriz	SA_MAN	Sales	12000	D

...

10. Нужны сведения о том, какие сотрудники были наняты после приема на работу после Davies. Создайте запрос для вывода фамилий и дат найма всех таких сотрудников.

LAST_NAME	HIRE_DATE
Kochhar	21-SEP-05
Hunold	03-JAN-06
Ernst	21-MAY-07
Austin	25-JUN-05
Pataballa	05-FEB-06
Lorentz	07-FEB-07
Chen	28-SEP-05
Sciarra	30-SEP-05
Urman	07-MAR-06

...

11. Нужны сведения о всех сотрудникахх, нанятых раньше своих менеджеров. Выведите фамилии и даты найма таких сотрудников, а также фамилии и даты найма их менеджеров.

LAST_NAME	HIRE_DATE	LAST_NAME_1	HIRE_DATE_1
Kaufling	01-MAY-03	King	17-JUN-03
Raphaely	07-DEC-02	King	17-JUN-03
De Haan	13-JAN-01	King	17-JUN-03
Higgins	07-JUN-02	Kochhar	21-SEP-05
Baer	07-JUN-02	Kochhar	21-SEP-05
Mavris	07-JUN-02	Kochhar	21-SEP-05
Whalen	17-SEP-03	Kochhar	21-SEP-05
Greenberg	17-AUG-02	Kochhar	21-SEP-05
Austin	25-JUN-05	Hunold	03-JAN-06

...

Практическое задание 6 решения

1. Выведите город и страну, в которой находится город. Используйте таблицы `LOCATIONS` и `COUNTRIES`. Для получения результатов используйте натуральное соединение.

```
SELECT city, country_id
FROM   locations
NATURAL JOIN countries;
```

2. Выведите название отдела, город и страну где находится отдел.

```
SELECT department_name, city, country_id
FROM   locations
NATURAL JOIN countries NATURAL JOIN departments;
```

3. Напишите запрос для вывода фамилии и названия отдела для всех служащих. Укажите явно соединение по столбцу department_id

```
SELECT last_name, department_name
FROM   employees
JOIN   departments
USING (department_id);
```

4. Напишите запрос для вывода фамилии и названия отдела всех служащих, работающих в городе Toronto.

```
SELECT e.last_name, d.department_name
FROM   employees e JOIN departments d
ON     (e.department_id = d.department_id)
JOIN   locations l
ON     (d.location_id = l.location_id)
WHERE l.city = 'Toronto';
```

5. Выведите фамилии сотрудников и номера их менеджеров.

```
SELECT w.last_name , m.last_name
FROM   employees w join employees m
ON     (w.manager_id = m.employee_id);
```

6. Однако, в результате отсутствует King. Измените запрос так, чтобы получить фамилии всех служащих, включая King, который не имеет менеджера. Упорядочьте результат по возрастанию номера служащего.

```
SELECT w.last_name , m.last_name
FROM   employees w LEFT OUTER
JOIN employees m
ON     (w.manager_id = m.employee_id)
ORDER BY w.employee_id;
```

7. Подготовьте отчет, в котором показываются номер отдела, фамилия служащего и фамилии всех служащих, работающих в одном отделе с данным служащим. Дайте столбцам имена dept, employee, colleague.

```
SELECT e.department_id dept, e.last_name employee,
       c.last_name colleague
FROM   employees e JOIN employees c
ON     (e.department_id = c.department_id)
WHERE  e.employee_id <> c.employee_id;
```

8. Добавьте в конец запроса фразу `ORDER BY dept, 2 , colleague;`
Проверьте, что команда выполнилась успешно.

```
SELECT e.department_id dept, e.last_name employee,
       c.last_name colleague
FROM   employees e JOIN employees c
ON     (e.department_id = c.department_id)
WHERE  e.employee_id <> c.employee_id
ORDER BY dept, 2 , colleague;
```

9. Нужен отчет о категориях должностей и окладах. Сначала посмотрите структуру таблицы `JOB_GRADES`. В БД Oracle **DESC JOB_GRADES;** В psql **`\d job_grades;`** Затем создайте запрос для вывода фамилии, должности, названия отдела, оклада и категории (`GRADE_LEVEL`) всех служащих.

```
SELECT e.last_name, e.job_id, d.department_name,
```

```
       e.salary, j.grade_level
FROM   employees e JOIN departments d
ON     (e.department_id = d.department_id)
JOIN   job_grades j
ON    (e.salary BETWEEN j.lowest_sal AND j.highest_sal);
```

10. Нужны сведения о том, какие сотрудники были наняты после приема на работу после Davies. Создайте запрос для вывода фамилий и дат найма всех таких служащих.

```
SELECT e.last_name, e.hire_date
FROM   employees e JOIN employees davies
ON     (davies.last_name = 'Davies')
WHERE  davies.hire_date < e.hire_date;
```

11. Нужны сведения о всех служащих, нанятых раньше своих менеджеров. Выведите фамилии и даты найма таких служащих, а также фамилии и даты найма их менеджеров.

```
SELECT w.last_name, w.hire_date, m.last_name, m.hire_date
FROM   employees w JOIN employees m
ON    (w.manager_id = m.employee_id)
WHERE    w.hire_date <  m.hire_date;
```

Практическое задание 7

Определите правильность следующих трех утверждений. Обведите ответ “Да“ или “Нет“.

1. Групповые функции применяются к большому количеству строк для получения одного результата на группу.

2. При вычислении групповых функций учитываются неопределенные значения.

3. Предложение `WHERE` ограничивает количество выбираемых строк до того, как они обрабатываются групповыми функциями.

4. Напишите запрос для вывода самого высокого, самого низкого, суммы всех окладов и средний оклада по всем служащим.
БД Oracle

MAX(SALARY)	MIN(SALARY)	SUM(SALARY)	AVG(SALARY)
24000	2100	691416	6461.831775700934579439252336448598130841

БД postgres

max	min	sum	avg
24000.00	2100.00	691416.00	6461.8317757009345794

В БД Oracle размерность типа NUMBER, используемого внутренне для хранения всех численных типов и для вычислений 39 или 40 разрядов.

5. Измените запрос так, чтобы получить то же самое отдельно по каждой должности (JOB_ID). Добавьте идентификатор должности первым столбцом в выборке.

JOB_ID	MAX(SALARY)	MIN(SALARY)	SUM(SALARY)	AVG(SALARY)
IT_PROG	9000	4200	28800	5760
AC_MGR	12008	12008	12008	12008
AC_ACCOUNT	8300	8300	8300	8300
ST_MAN	8200	5800	36400	7280
PU_MAN	11000	11000	11000	11000
AD_ASST	4400	4400	4400	4400
AD_VP	17000	17000	34000	17000
SH_CLERK	4200	2500	64300	3215

...
Почему в столбце со средним окладом целые числа?

6. Напишите запрос для вывода должности и количества служащих, занимающих каждую должность.

JOB_ID	COUNT(EMPLOYEE_ID)
IT_PROG	5
AC_MGR	1
AC_ACCOUNT	1
ST_MAN	5
PU_MAN	1
AD_ASST	1
AD_VP	2
SH_CLERK	20

...

7. Получите количество служащих, имеющих подчиненных, без их перечисления.
Подсказка: используйте столбец `MANAGER_ID` для определения числа менеджеров.

count
18

8. Напишите запрос для вывода разности между самым высоким и самым низким окладами.

?column?
21900.00

9. Напишите запрос для вывода номера каждого менеджера и заработной платы самого низкооплачиваемого из его подчиненных. Исключите всех, для которых неизвестны их менеджеры. Исключите все группы, где минимальный оклад составляет менее 6000. Отсортируйте выходные строки в порядке убывания оклада.

MANAGER_ID	MIN(SALARY)
102	9000
205	8300
145	7000
146	7000
108	6900
147	6200
149	6200
148	6100

10. Следующий запрос в БД Oracle выводит общее количество служащих и количества служащих, нанятых в 2002, 2003 годах.

```
SELECT  COUNT(*) total,
        SUM(DECODE(TO_CHAR(hire_date, 'YYYY'),2002,1,0)) "2002",
        SUM(DECODE(TO_CHAR(hire_date, 'YYYY'),2003,1,0)) "2003"
FROM    employees;
```

DECODE нестандартная функция. Вместо нее лучше использовать стандартное выражение CASE.
Синтаксис функции:

```
DECODE(expr, search, result [, search, result...] [, default])
```

Синтаксис выражения:

```
CASE WHEN expr=search THEN result [...] ELSE default END
```

Нужно переписать запрос на использование стандартного синтаксиса.
Сначала проверьте, что TO_CHAR(hire_date, 'YYYY'),2002,1,0) работает корректно в БД postgres:

```
SELECT TO_CHAR(hire_date, 'YYYY') FROM employees;
```

Добавьте CASE и выполните в соединении postgres:

```
SELECT CASE WHEN TO_CHAR(hire_date, 'YYYY')=2002 THEN 2002 END FROM employees;
```

а) Почему в postgres команда не выполняется выдаёт ошибку:

```
ОШИБКА: оператор не существует: text = integer
  Hint: Оператор с данными именем и типами аргументов не найден. Возможно, вам следует
добавить явные приведения типов.
```

Устраните ошибку. Убедитесь, что команда одинаково работает в БД postgres и БД Oracle

11. Ниже представлен запрос для вывода всех должностей и суммы заработной платы служащих, работающих в этой должности в отделах 20, 50. Последний столбец должен содержать сумму заработной платы служащих *этих* отделов, занимающих каждую конкретную должность.

```
SELECT   job_id "Job",
         SUM(DECODE(department_id , 20, salary)) "Dept 20",
         SUM(DECODE(department_id , 50, salary)) "Dept 50"
         SUM(salary) "Total"
FROM     employees
GROUP BY job_id;
```

Перепишите код, так чтобы он работал в БД postgres и БД Oracle.

```
SELECT   job_id "Job",
         SUM(CASE WHEN department_id=20 THEN salary END) "Dept 20",
         SUM(CASE WHEN department_id=50 THEN salary END) "Dept 50",
         SUM(salary) "Total"
FROM     employees
GROUP BY job_id;
```

При выполнении в JDeveloper в соединении postgres вместо F5 используйте Ctrl+Enter, чтобы данные были выведены компактно. Для успешного выполнения нужно встать на строку команды, а не на пустую строку. Команда в процессе выполнения должна подсветиться синим цветом.
Большая ширина результата SUM() обусловлена особенностями форматирования результата JDeveloper. При выполнении команды из кода приложения роли не играет.

sum() возвращает bigint для аргументов smallint или int, numeric для аргументов bigint, и тип аргумента в остальных случаях. Столбец SALARY определен как десятичное число numeric(8,2).
Поэтому использовать дополнительные функции вокруг SUM() для форматирования не требуется.
Пример, если нужно округлить зарплату до целых числел:

```
SELECT   job_id "Job",
         CAST(SUM(CASE WHEN department_id=20 THEN salary END) AS INTEGER) "Dept 20",
         CAST(SUM(CASE WHEN department_id=50 THEN salary END) AS INTEGER) "Dept 50",
         CAST(SUM(salary) AS INTEGER) "Total"
FROM     employees
GROUP BY job_id;
```

12. Нужно вывести список отделов из таблицы employees, исключив дублирование. Используется запрос:

```
SELECT distinct department_id FROM employees;
```

Перепишите запрос на аналогичный, но с использованием группировки
Убедитесь, что выдалось 12 строк в обоих случаях.

13. Нужно подсчитать количество уникальных номеров отделов из таблицы employees. Используется запрос:

```
SELECT COUNT(DISTINCT department_id) FROM employees;
```

Выдалось 11 строк. В одной из строк номер отдела пуст:

```
SELECT * FROM employees WHERE department_id IS NULL;
SELECT COUNT(department_id) from employees where department_id IS NULL;
```

Обратите внимание COUNT(department_id) не подсчитывает пустые значения - выдан 0.
Запрос переписали так:

```
SELECT COUNT(*) FROM (SELECT DISTINCT department_id from employees) A;
```

Выдается 12 строк.
Перепишите запрос на использование GROUP BY.
Убедитесь, что выдалось 12 строк.

Практическое задание 7 решения

Определите правильность следующих трех утверждений. Обведите ответ “Да“ или “Нет“.

1. Групповые функции применяются к большому количеству строк для получения одного результата на группу.
Да

2. При вычислении групповых функций учитываются неопределенные значения.
Нет

3. Предложение `WHERE` ограничивает количество выбираемых строк до того, как они обрабатываются групповыми функциями.
Да

4. Напишите запрос для вывода самого высокого, самого низкого, суммы всех окладов и средний оклада по всем служащим.

```
SELECT MAX(salary),
       MIN(salary),
       SUM(salary),
       AVG(salary)
FROM   employees;
```

5. Измените запрос так, чтобы получить то же самое отдельно по каждой должности (JOB_ID). Добавьте идентификатор должности первым столбцом в выборке.

```
SELECT job_id, MAX(salary),
       MIN(salary),
       SUM(salary),
       AVG(salary)
FROM   employees
GROUP BY job_id;
```

Почему в столбце со средним окладом целые числа?
Сгруппируем по DEPARTMENT_ID вместо JOB_ID:

```
SELECT department_id,  MAX(salary),
       MIN(salary),
       SUM(salary),
       AVG(salary)
FROM   employees
GROUP BY department_id;
```

Выдаются десятичные числа, округлений нет. При группировке по JOB_ID получились целые числа.

DEPARTMENT_ID	MAX(SALARY)	MIN(SALARY)	SUM(SALARY)	AVG(SALARY)
100	12008	6900	51608	8601.3333333333333333333333333333333333
30	11000	2500	24900	4150
(null)	7000	7000	7000	7000
90	24000	17000	58000	19333.333333333333333333333333333333333
20	13000	6000	19000	9500
70	10000	10000	10000	10000
110	12008	8300	20308	10154
50	8200	2100	156400	3475.5555555555555555555555555555555556

...

6. Напишите запрос для вывода должности и количества служащих, занимающих каждую должность.

```
SELECT job_id, COUNT(*)
FROM employees
GROUP BY job_id;
```

Более оптимально использовать не *, а столбец-первичный ключ, так как в этой таблице он имеется.

```
SELECT job_id, COUNT(employee_id)
FROM employees
GROUP BY job_id;
```

7. Получите количество служащих, имеющих подчиненных, без их перечисления.

Подсказка: используйте столбец MANAGER_ID для определения числа менеджеров.

```
SELECT COUNT(DISTINCT manager_id)
FROM   employees;
```

8. Напишите запрос для вывода разности между самым высоким и самым низким окладами.

```
SELECT   MAX(salary) - MIN(salary)
FROM     employees;
```

9. Напишите запрос для вывода номера каждого менеджера и заработной платы самого низкооплачиваемого из его подчиненных. Исключите всех, для которых неизвестны их менеджеры. Исключите все группы, где минимальный оклад составляет менее 6000. Отсортируйте выходные строки в порядке убывания оклада.

```
SELECT   manager_id, MIN(salary)
FROM     employees
WHERE    manager_id IS NOT NULL
GROUP BY manager_id
HAVING   MIN(salary) > 6000
ORDER BY MIN(salary) DESC;
```

10. Следующий запрос в БД Oracle выводит общее количество служащих и количества служащих, нанятых в 2002, 2003 годах.

```
SELECT  COUNT(*) total,
        SUM(DECODE(TO_CHAR(hire_date, 'YYYY'),2002,1,0)) "2002",
        SUM(DECODE(TO_CHAR(hire_date, 'YYYY'),2003,1,0)) "2003"
FROM    employees;
```

Перепишите запрос на использование стандартного синтаксиса.
Сначала проверьте, что TO_CHAR(hire_date, 'YYYY'),2002,1,0) работает корректно в БД postgres:

```
SELECT TO_CHAR(hire_date, 'YYYY') FROM employees;
```

Поскольку SUM() работает с числами, то результат TO_CHAR нужно привести к числу

```
SELECT CAST(TO_CHAR(hire_date, 'YYYY') AS NUMERIC) FROM employees;
```

Добавьте CASE:

```
SELECT CASE WHEN CAST(TO_CHAR(hire_date, 'YYYY') AS NUMERIC)=2002 THEN 2002 END FROM
employees;
```

Теперь можно переписать команду. Убедитесь, что команда одинаково работает в БД postgres и БД Oracle

```
SELECT  COUNT(*) total,
SUM(CASE WHEN CAST(TO_CHAR(hire_date, 'YYYY') AS NUMERIC)=2002 THEN 2002 END) "2002",
SUM(CASE WHEN CAST(TO_CHAR(hire_date, 'YYYY') AS NUMERIC)=2003 THEN 2003 END) "2003"
FROM    employees;
```

11. Ниже представлен запрос для вывода всех должностей и суммы заработной платы служащих, работающих в этой должности в отделах 20, 50. Последний столбец должен содержать сумму заработной платы служащих *этих* отделов, занимающих каждую конкретную должность.

```
SELECT   job_id "Job",
         SUM(DECODE(department_id , 20, salary)) "Dept 20",
         SUM(DECODE(department_id , 50, salary)) "Dept 50"
         SUM(salary) "Total"
FROM     employees
GROUP BY job_id;
```

Перепишите код, так чтобы он работал в БД postgres и БД Oracle.

```
SELECT   job_id "Job",
         SUM(CASE WHEN department_id=20 THEN salary END) "Dept 20",
         SUM(CASE WHEN department_id=50 THEN salary END) "Dept 50",
         SUM(salary) "Total"
FROM     employees
GROUP BY job_id;
```

При выполнении в JDeveloper в соединении postgres вместо F5 используйте Ctrl+Enter, чтобы данные были выведены компактно. Для успешного выполнения нужно встать на строку команды, а не на пустую строку. Команда в процессе выполнения должна подсветиться синим цветом.
Большая ширина результата SUM() обусловлена особенностями форматирования результата JDeveloper. При выполнении команды из кода приложения роли не играет.

sum() возвращает bigint для аргументов smallint или int, numeric для аргументов bigint, и тип аргумента в остальных случаях. Столбец SALARY определен как десятичное число numeric(8,2).
Поэтому использовать дополнительные функции вокруг SUM() для форматирования не требуется.
Пример, если нужно округлить зарплату до целых числел:

```
SELECT   job_id "Job",
         CAST(SUM(CASE WHEN department_id=20 THEN salary END) AS INTEGER) "Dept 20",
         CAST(SUM(CASE WHEN department_id=50 THEN salary END) AS INTEGER) "Dept 50",
         CAST(SUM(salary) AS INTEGER) "Total"
FROM     employees
GROUP BY job_id;
```

12. Нужно вывести список отделов из таблицы employees исключив дублирование. Используется запрос:

```
SELECT distinct department_id FROM employees;
```

Перепишите запрос на аналогичный с использованием группировки

```
SELECT department_id FROM employees GROUP BY department_id;
```

Выдалось 12 строк в обоих случаях.

13. Нужно подсчитать количество уникальных номеров отделов из таблицы employees. Используется запрос:

```
SELECT COUNT(DISTINCT department_id) FROM employees;
```

Выдалось 11 строк. В одной из строк номер отдела пуст:

```
SELECT * FROM employees WHERE department_id IS NULL;
SELECT COUNT(department_id) from employees where department_id IS NULL;
```

Обратите внимание COUNT(выражение) не подсчитывает пустые значения - выдан 0.
Запрос переписали так:

```
SELECT COUNT(*) FROM (SELECT DISTINCT department_id from employees) A;
```

Выдается 12 строк. Перепишите запрос на использование GROUP BY:

```
SELECT COUNT(*) FROM (SELECT department_id FROM employees GROUP BY department_id) A;
```

Выдается 12 строк.
На маленьком количестве строк два последних запроса выполняются в postgres по одинаковому плану выполнения.

```
explain SELECT COUNT(*) FROM (SELECT department_id FROM employees GROUP BY
department_id) A;
QUERY PLAN
-----------------------------------------------------------------
Aggregate  (cost=3.60..3.61 rows=1 width=0)
  ->  HashAggregate  (cost=3.35..3.46 rows=11 width=5)
        Group Key: employees.department_id
        ->  Seq Scan on employees  (cost=0.00..3.08 rows=108 width=5)
```

Практическое задание 8

1. Нужен запрос, при выполнении которого пользователь вводит фамилию служащего, после чего выводятся фамилии и даты найма сотрудников, работающих в одном отделе с этим служащим (сам этот служащий не выводится). Например, если вводится Zlotkey, тогда по запросу выводятся другие служащие, работающие с ним в отделе (за исключением служащего Zlotkey).

```
SELECT last_name, hire_date
FROM   employees
WHERE  department_id = (SELECT department_id
                        FROM   employees
                        WHERE  last_name = 'Zlotkey')
AND    last_name <> 'Zlotkey';
```

2. Создайте запрос для вывода фамилий всех служащих, оклад которых выше среднего. Отсортируйте выходные данные в порядке увеличения окладов.

```
SELECT last_name, salary
FROM   employees
WHERE  salary > (SELECT AVG(salary)
                 FROM   employees)
ORDER BY salary;
```

3. Создайте запрос для вывода фамилий всех служащих, работающих в одном отделе с любым служащим, фамилия которого содержит букву "*u*".

```
SELECT last_name
FROM   employees
WHERE  department_id IN (SELECT department_id
                         FROM   employees
                         WHERE  last_name like '%u%');
```

4. Нужен отчет, в котором выводятся фамилии, идентификатор местоположения отдела (location_id) которых равен 1700.

```
SELECT last_name
FROM   employees
WHERE  department_id IN (SELECT department_id
                         FROM   departments
                         WHERE  location_id = 1700);
```

5. Нужен отчет, в котором выводится список фамилий всех служащих, подчиненных King.

```
SELECT last_name
FROM   employees
WHERE  manager_id = (SELECT employee_id
                     FROM   employees
                     WHERE  last_name = 'King');
```

6. Нужен отчет, в котором выводится фамилия каждого служащего, работающего в отделе с названием `Executive`.

```
SELECT last_name
FROM   employees
WHERE  department_id IN (SELECT department_id
                         FROM   departments
                         WHERE  department_name = 'Executive');
```

7. Измените запрос и выведите фамилии и оклады всех служащих при условии, что оклады превышают средний и служащие работают в одном отделе с любым сотрудником с буквой "u" в фамилии (last_name).

```
SELECT last_name, salary
FROM   employees
WHERE  department_id IN (SELECT department_id
                         FROM   employees
                         WHERE  last_name like '%u%')
AND salary > (SELECT AVG(salary)
              FROM employees);
```

8. Создайте отчет, в котором показываются номер отдела с наивысшей средней заработной платой и наименьший оклад работающего в нем сотрудника. Приведен пример, который работает в БД Oracle, однако недопустим с точки зрения стандарта SQL (глава 7 слайд 16).

```
SELECT department_id, MIN(salary)
FROM   employees
GROUP BY department_id
HAVING AVG(salary) = (SELECT MAX(AVG(salary))
                      FROM   employees
                      GROUP BY department_id);
```

Перепишите запрос так, чтобы он соответствовал стандарту.

```
SELECT department_id, MIN(salary)
FROM   employees
GROUP BY department_id
HAVING AVG(salary) = (SELECT MAX(avgsal)
                      FROM (SELECT AVG(salary) avgsal
                            FROM employees GROUP BY department_id) a);
```

9. Создайте отчет, в котором выводятся номера, наименования и местоположения отделов, в которых не работают торговые представители (`job_id != 'SA_REP'`).

```
SELECT *
FROM   departments
WHERE  department_id NOT IN(SELECT department_id
                            FROM employees
                            WHERE job_id = 'SA_REP'
                            AND department_id IS NOT NULL);
```

10. Выведите номера отделов и количество работающих в них сотрудников, где количество сотрудников меньше 3

```
SELECT d.department_id, COUNT(*)
FROM   departments d JOIN employees e
ON     d.department_id = e.department_id
GROUP BY d.department_id
HAVING COUNT(*) < 3;
```

Добавьте к выборке название отдела

```
SELECT d.department_id, d.department_name, COUNT(*)
FROM   departments d JOIN employees e
ON     d.department_id = e.department_id
GROUP BY d.department_id, d.department_name
HAVING COUNT(*) < 3;
```

11. Нужно вывести отчет, в котором выводятся фамилии и средние оклады в отделе сотрудника:

```
SELECT e.last_name, AVG(s.salary)
FROM   employees e JOIN employees s
ON     e.department_id = s.department_id
GROUP BY e.department_id, e.last_name;
```

12. Выдаст ли следующий запрос те же данные?

```
SELECT e.last_name, AVG(s.salary)
FROM   employees e JOIN employees s
ON     e.department_id = s.department_id
GROUP BY s.department_id, e.last_name;
```

Да, выдаст. Значения равны благодаря равенству в ON.

13. Выдаст ли следующий запрос те же данные?

```
SELECT e.last_name, AVG(s.salary)
FROM   employees e JOIN employees s
ON     e.department_id = s.department_id
GROUP BY e.last_name;
```

Для проверки добавьте в конец обоих запросов `HAVING e.last_name IN ('King', 'Taylor');`
Есть два сотрудника с фамилиями King и Taylor в разных отделах. Средние оклады будут вычислены неверно.

Практическое задание 9

1. В последнем пункте предыдущего задания вы использовали HAVING для проверки. Используйте MINUS/EXCEPT для проверки.

```
SELECT e.last_name, AVG(s.salary)
FROM   employees e JOIN employees s
ON     e.department_id = s.department_id
GROUP BY e.department_id, e.last_name
MINUS/EXCEPT
SELECT e.last_name, AVG(s.salary)
FROM   employees e JOIN employees s
ON     e.department_id = s.department_id
GROUP BY e.last_name;
```

Запрос выдал 4 строки

2. Поменяйте местами запросы и проверьте, изменится ли результат

```
SELECT e.last_name, AVG(s.salary)
FROM   employees e JOIN employees s
ON     e.department_id = s.department_id
GROUP BY e.last_name
MINUS/EXCEPT
SELECT e.last_name, AVG(s.salary)
FROM   employees e JOIN employees s
ON     e.department_id = s.department_id
GROUP BY e.department_id, e.last_name;
```

Результат поменялся - выдано 2 строки. Почему?
Если вы не можете сразу ответить, выполните отдельно запросы, добавив `HAVING e.last_name IN ('King', 'Taylor');`
В результате вы получие, что оба запроса выдают 6 разных строк. При выполнении MINUS/EXCEPT строки второго запроса, которые отсутствуют в первом в результат не попадают.

3. Выполните следующие команды:

```
SELECT  commission_pct
FROM    employees
UNION
SELECT  commission_pct
FROM    employee;
```

обратите внимание, что NULL выдался в одной строке - дубли отфильтрованы

```
SELECT  commission_pct
FROM    employees
INTERSECT
SELECT  commission_pct
FROM    employee;
```

обратите внимание, что NULL выдался в одной строке - дубли отфильтрованы

```
SELECT  commission_pct
FROM    employees
MINUS/EXCEPT
SELECT  commission_pct
FROM    employee;
```

Строк не выдано - все дубли, в том числе NULL отфильтрованы.

4. Нужен список номеров отделом, в которых нет сотрудников с идентификатором должности `ST_CLERK`. Используйте операторы над множествами для создания отчета.

```
SELECT department_id
FROM   departments
MINUS/EXCEPT
SELECT department_id
FROM   employees
WHERE  job_id = 'ST_CLERK';
```

5. Нужен перечень стран, в которых не располагаются подразделения компании. Выведите идентификаторы и названия таких стран. Для создания отчета используйте операторы над множествами.

```
SELECT country_id, country_name
FROM   countries
```

```
MINUS/EXCEPT
SELECT l.country_id,c.country_name
FROM   locations l JOIN  countries c
ON     (l.country_id = c.country_id);
```

6. Используя операторы над множествами, выведите список должностей отделов 10, 50 и 20 в таком же порядке отделов. Включите в отчет столбцы `job_id` и `department_id`.

```
SELECT  job_id, department_id, 'x' dummy
FROM    employees
WHERE   department_id = 10
UNION
SELECT  job_id, department_id, 'y' dummy
FROM    employees
WHERE   department_id = 50
UNION
SELECT  job_id, department_id, 'z' dummy
FROM    employees
WHERE   department_id = 20
ORDER BY  dummy;
```

7. Сделайте так, чтобы сначала выполнялось объединение 2 и 3 SELECT. Используйте круглые скобки.

```
SELECT  job_id, department_id, 'x' dummy
FROM    employees
WHERE   department_id = 10
UNION
(SELECT  job_id, department_id, 'y' dummy
FROM    employees
WHERE   department_id = 50
UNION
SELECT  job_id, department_id, 'z' dummy
FROM    employees
WHERE   department_id = 20)
ORDER BY  dummy;
```

8. Выведите номер сотрудника и идентификатор его должности, если его текущая должность совпадает с той, которую он уже занимал, работая в компании

```
SELECT    employee_id,job_id
FROM      employees
INTERSECT
SELECT   employee_id,job_id
FROM     job_history;
```

9. Необходим отчет, в котором выводится следующее:

- фамилии и отделы всех сотрудников из таблицы `EMPLOYEES`, независимо от того, относятся ли они к какому-то отделу или нет;
- номера и наименования всех отделов из таблицы `DEPARTMENTS`, независимо от того, есть ли в них сотрудники или нет.

Для получения отчета напишите составной запрос.

```
SELECT last_name,department_id, NULL
FROM   employees
UNION
SELECT NULL,department_id,department_name
FROM  departments;
```

10. Выполните команду

```
SELECT manager_id, job_id, sum(salary)
FROM employees
WHERE manager_id < 110
GROUP BY ROLLUP(manager_id,job_id);
```

11. Чтобы идентифицировать результаты группировок и отличить их от значений NULL добавьте столбцы с функцией GROUPPING. Используйте Ctrl+Enter вместо F5 для выполнения запроса, чтобы было удобнее просматривать результат

```
SELECT manager_id MGR , job_id JOB,
```

```
sum(salary), GROUPING(manager_id), GROUPING(job_id)
FROM employees
WHERE manager_id < 110
GROUP BY ROLLUP(manager_id, job_id);
```

12. Замените ROLLUP на GROUPING SETS и убедитесь, что результат одинаков. Для написания запроса используйте слайд 21 главы 9.

```
SELECT manager_id MGR , job_id JOB,
sum(salary), GROUPING(manager_id), GROUPING(job_id)
FROM employees
WHERE manager_id < 110
GROUP BY GROUPING SETS ((manager_id, job_id),manager_id,());
```

13. Замените ROLLUP на CUBE

```
SELECT manager_id MGR , job_id JOB,
sum(salary), GROUPING(manager_id), GROUPING(job_id)
FROM employees
WHERE manager_id < 110
GROUP BY CUBE(manager_id, job_id);
```

14. Замените CUBE на GROUPING SETS

```
SELECT manager_id MGR , job_id JOB,
sum(salary), GROUPING(manager_id), GROUPING(job_id)
FROM employees
WHERE manager_id < 110
GROUP BY GROUPING SETS ((manager_id, job_id),manager_id,job_id,());
```

Практическое задание 10

1. Выберите зарплату каждого сотрудника и среднюю зарплату сотрудников в его отделе, используя оконную функцию. Отфильтруйте результат выведя только сотрудников 60 отдела.

```
DEPARTMENT_ID LAST_NAME                     SALARY AVG(salary)
------------- ------------------------- ---------- ----------
           60 Lorentz                         4200       5760
           60 Austin                          4800       5760
           60 Pataballa                       4800       5760
           60 Ernst                           6000       5760
           60 Hunold                          9000       5760
```

2. Выведите колличество строк в окнах, чтобы понять, по скольким строкам в окне бралось усреднение

```
DEPARTMENT_ID LAST_NAME                     SALARY COUNT(salary) AVG(salary)
------------- ------------------------- ---------- ---------- ----------
           60 Lorentz                         4200          5       5760
           60 Austin                          4800          5       5760
           60 Pataballa                       4800          5       5760
           60 Ernst                           6000          5       5760
           60 Hunold                          9000          5       5760
```

3. Добавьте упорядочивание ORDER BY salary DESC, сделав окна скользящими и сравните результат

department_id	last_name	salary	count	avg
60	Lorentz	4200	5	5760.0000000000000000
60	Austin	4800	4	6150.0000000000000000
60	Pataballa	4800	4	6150.0000000000000000
60	Ernst	6000	2	7500.0000000000000000
60	Hunold	9000	1	9000.0000000000000000

4. Измените порядок в окне с DESC на ASC и сравните результат

department_id	last_name	salary	count	avg
60	Lorentz	4200	1	4200.0000000000000000
60	Austin	4800	3	4600.0000000000000000
60	Pataballa	4800	3	4600.0000000000000000
60	Ernst	6000	4	4950.0000000000000000
60	Hunold	9000	5	5760.0000000000000000

5. Добавьте ROW_NUMBER() чтобы вывести номер текущей строки в нескользящем окне (уберите везде ORDER BY salary ASC). Используйте Ctrl+Enter вместо F5. Для успешного выполнения нужно встать на строку команды, а не на пустую строку. Команда в процессе выполнения должна подсветиться синим цветом.

department_id	last_name	salary	row_number	count	avg
60	Lorentz	4200	5	5	5760.0000000000000000
60	Austin	4800	3	5	5760.0000000000000000
60	Pataballa	4800	4	5	5760.0000000000000000
60	Ernst	6000	2	5	5760.0000000000000000
60	Hunold	9000	1	5	5760.0000000000000000

6. Повторите для скользящего окна **ORDER BY salary DESC**

department_id	last_name	salary	row_number	count	avg
60	Lorentz	4200	5	5	5760.0000000000000000
60	Austin	4800	3	4	6150.0000000000000000
60	Pataballa	4800	4	4	6150.0000000000000000
60	Ernst	6000	2	2	7500.0000000000000000
60	Hunold	9000	1	1	9000.0000000000000000

7. Добавьте RANK() и DENSE_RANK() обратите внимание на отличия в результатах этих функций

8. Добавьте CUME_DIST() и PERCENT_RANK()

department_id	last_name	salary	row_number	rank	dense_rank	cume_dist	percent_rank	count	avg
60	Lorentz	4200	5	5	4	1.0	1.0	5	5760
60	Austin	4800	3	3	3	0.8	0.5	4	6150
60	Pataballa	4800	4	3	3	0.8	0.5	4	6150
60	Ernst	6000	2	2	2	0.4	0.25	2	7500
60	Hunold	9000	1	1	1	0.2	0.0	1	9000

9. Выполните команду и посмотрите её описание на слайде 13 главы 10:

```
SELECT last_name, salary, NTILE(4) OVER (ORDER BY salary DESC) AS quartile
FROM employees
WHERE department_id = 100
ORDER BY last_name, salary, quartile;
```

10. Выполните команду и идентифицируйте как сдвигаются строки. Обратите внимание на NULL в четырех полях

```
SELECT last_name, salary,
 LAG(salary,2) OVER (ORDER BY salary) AS LAG1,
 LEAD(salary,2) OVER (ORDER BY salary) AS LEAD1
FROM employees;
```

11. Посмотрите результат выполнения команд в Oracle и postgres.
Посмотрите описание функций на слайде 18 главы 10.
Oracle:

```
SELECT LISTAGG(last_name, '; ') WITHIN GROUP (ORDER BY hire_date, last_name),
 MIN(hire_date)
 FROM employees WHERE department_id=30;
```

postgres:

```
SELECT STRING_AGG(last_name, '; ' ORDER BY hire_date,last_name),
 MIN(hire_date)
 FROM employees WHERE department_id=30;
```

12. Посмотрите результат выполнения команды. Обратите внимание на результат LAST_VALUE на одном скользящем окне которое определено как OVER (ORDER BY salary)

```
SELECT department_id, last_name, salary,
 FIRST_VALUE(last_name) OVER (ORDER BY salary ) AS F,
 LAST_VALUE(last_name)  OVER (ORDER BY salary ) AS L
FROM employees;
```

department_id	last_name	salary	first_value	last_value
50	Olson	2100	Olson	Olson
50	Markle	2200	Olson	Philtanker
50	Philtanker	2200	Olson	Philtanker
50	Gee	2400	Olson	Landry
50	Landry	2400	Olson	Landry
50	Patel	2500	Olson	Perkins
30	Colmenares	2500	Olson	Perkins

Практическое задание 10 решения

1. Выберите зарплату каждого сотрудника и среднюю зарплату сотрудников в его отделе.используя оконную функцию

```
SELECT department_id, last_name, salary,
  AVG(salary)  OVER (PARTITION BY department_id) AS average
FROM employees WHERE department_id=60 ORDER BY salary;
```

2. Выведите колличество строк в окнах, чтобы понять, по скольким строкам в окне бралось усреднение

```
SELECT department_id, last_name, salary,
 COUNT(salary) OVER (PARTITION BY department_id) AS cn,
 AVG(salary)   OVER (PARTITION BY department_id) AS average
FROM employees WHERE department_id=60 ORDER BY salary;
```

3. Добавьте упорядочивание ORDER BY salary DESC, сделав окна скользящими и сравните результат

```
SELECT department_id, last_name, salary,
 COUNT(salary) OVER (PARTITION BY department_id ORDER BY salary DESC) AS cn,
 AVG(salary)   OVER (PARTITION BY department_id ORDER BY salary DESC) AS average
FROM employees WHERE department_id=60 ORDER BY salary;
```

4. Измените порядок в окне с DESC на ASC и сравните результат

```
SELECT department_id, last_name, salary,
 COUNT(salary) OVER (PARTITION BY department_id ORDER BY salary ASC) AS cn,
 AVG(salary)   OVER (PARTITION BY department_id ORDER BY salary ASC) AS average
FROM employees WHERE department_id=60 ORDER BY salary;
```

5. Добавьте ROW_NUMBER() чтобы вывести номер текущей строки в окне. Используйте Ctrl+Enter вместо F5. Для успешного выполнения нужно встать на строку команды, а не на пустую строку. Команда в процессе выполнения должна подсветиться синим цветом.

```
SELECT department_id, last_name, salary,
 ROW_NUMBER()  OVER (PARTITION BY department_id) AS rn,
 COUNT(salary) OVER (PARTITION BY department_id) AS cn,
 AVG(salary)   OVER (PARTITION BY department_id) AS average
FROM employees WHERE department_id=60 ORDER BY salary;
```

6. Повторите для скользящего окна

```
SELECT department_id, last_name, salary,
 ROW_NUMBER()  OVER (PARTITION BY department_id ORDER BY salary DESC) AS rn,
 COUNT(salary) OVER (PARTITION BY department_id ORDER BY salary DESC) AS cn,
 AVG(salary)   OVER (PARTITION BY department_id ORDER BY salary DESC) AS average
FROM employees WHERE department_id=60 ORDER BY salary;
```

7. Добавьте RANK() и DENSE_RANK() обратите внимание на отличия в результатах этих функций

```
SELECT department_id, last_name, salary,
 ROW_NUMBER()  OVER (PARTITION BY department_id ORDER BY salary DESC) AS rn,
 RANK()        OVER (PARTITION BY department_id ORDER BY salary DESC) AS rk,
 DENSE_RANK()  OVER (PARTITION BY department_id ORDER BY salary DESC) AS dr,
 COUNT(salary) OVER (PARTITION BY department_id ORDER BY salary DESC) AS cn,
 AVG(salary)   OVER (PARTITION BY department_id ORDER BY salary DESC) AS average
FROM employees WHERE department_id=60 ORDER BY salary;
```

8. Добавьте CUME_DIST() и PERCENT_RANK()

```
SELECT department_id, last_name, salary,
 ROW_NUMBER()  OVER (PARTITION BY department_id ORDER BY salary DESC) AS rn,
 RANK()        OVER (PARTITION BY department_id ORDER BY salary DESC) AS rk,
 DENSE_RANK()  OVER (PARTITION BY department_id ORDER BY salary DESC) AS cd,
 CUME_DIST()   OVER (PARTITION BY department_id ORDER BY salary DESC) AS pk,
PERCENT_RANK() OVER (PARTITION BY department_id ORDER BY salary DESC) AS dr,
 COUNT(salary) OVER (PARTITION BY department_id ORDER BY salary DESC) AS cn,
 AVG(salary)   OVER (PARTITION BY department_id ORDER BY salary DESC) AS average
FROM employees WHERE department_id=60 ORDER BY salary;
```

Практическое задание 11

1. Создайте таблицу в JDeveloper в соединении postgres ИЛИ oracle. Выберите одно из соединений, с которым будете работать.

```
DROP TABLE emp;
CREATE TABLE emp
  (id  NUMERIC(4) CONSTRAINT emp_id_nn NOT NULL,
   last_name VARCHAR(25),
   first_name VARCHAR(25),
   email  VARCHAR(20),
   salary  NUMERIC(8) DEFAULT 1000);
```

2. Убедитесь в том, что таблица существует и строк в ней нет.

```
no rows selected
```

3. Вставьте данные в таблицу emp.
а) Напишите команду INSERT для вставки в таблицу emp первой строки из ниже приведенных образцов.

ID	LAST_NAME	FIRST_NAME	EMAIL	SALARY
1	Patel	Ralph	rpatel	2020
2	Dancs	**Betty**	bdancs	1860

Не указывайте столбцы в предложении INSERT.
б) Добавьте после команды INSERT в окне JDeveloper команду SELECT * FROM emp; чтобы сразу просматривать результат
в) **выполните** обе команды

```
1 row inserted.
id    last_name                 first_name                email                salary
----- ------------------------- ------------------------- -------------------- ---------
1     Patel                     Ralph                     rpatel               2020
```

4. Вставьте в таблицу EMP данные только двух столбцов **ID** и **FIRST_NAME**, используя вторую строку из вышеуказанных образцов, отредактировав команду INSERT в JDeveloper в окне соединения.
На этот раз укажите столбцы явно в предложении INSERT.
Убедитесь в том, что данные вставлены в таблицу

```
1 row inserted.
id    last_name                 first_name                email                salary
----- ------------------------- ------------------------- -------------------- ---------
1     Patel                     Ralph                     rpatel               2020
2                               Betty                                          1000
```

Обратите внимание, что SALARY у Betty равна 1000, хотя вы ее не указывали.
Почему?

5. Смените фамилию служащей номер 2 на “Dancs”.

```
1 row updated.
id    last_name                 first_name                email                salary
----- ------------------------- ------------------------- -------------------- ---------
1     Patel                     Ralph                     rpatel               2020
2     Dancs                     Betty                                          1000
```

6. Сделайте оклад равным 1000 для всех служащих, имеющих оклад ниже 1000.

```
0 rows updated.
```

Обратите внимание, что БД вернула сообщение об обновлении нуля строк.
Приложение, работающее с БД, получает от БД информацию о том, сколько строк было изменено.

7. Сделайте оклад равным 1000 для всех служащих, имеющих оклад ниже 2000.

```
1 row updated.
```

Обратите внимание, что БД вернула сообщение об обновлении одной строки. Однако данные в строке не изменились, так как у Betty оклад был равен 1000. Физически изменения произошли, даже если данные поменялись на те же самые. БД по команде UPDATE не проверяла значение столбца SALARY, она просто выполнила команду.

8. Сделайте оклад равным 1500 для всех служащих, имеющих оклад ниже 2000.

```
1 row updated.
```

Теперь данные поменялись.

9. Выполните следующие команды и посмотрите результат:

```
INSERT INTO emp SELECT * FROM emp;
SELECT * FROM emp;
```

Строки в таблице задублированы и указать БД с какой строкой вы хотите работать сложно.

10. Уберем дубликаты с использованием обычной таблицы.

```
CREATE TABLE emp1 AS SELECT DISTINCT * FROM emp;
DROP TABLE emp;
ALTER TABLE emp1 RENAME TO emp;
SELECT * FROM emp;
```

11. Удалите из таблицы `EMP` строку для служащей Betty Dancs.

```
DELETE FROM emp WHERE last_name = 'Dancs';
SELECT * FROM emp;
```

Обратите внимание, что как и для команды UPDATE, БД вернула количество обработанных командой DELETE строк.
Можно удалить дубликаты без пересоздания таблицы, но придется использовать функционал специфичный для БД конкретного производителя. В БД Oracle для цели удаления можно использовать служебные столбцы ROWID, ROWNUM, В БД postgres ctid, oid (по умолчанию oid не создаётся). Эти служебные столбцы можно использовать, если явно указать их имя.

12. Удалите все строки в таблице emp командой TRUNCATE

```
Table EMP truncated.
```

13. Выполните команду:

```
INSERT INTO emp SELECT employee_id, last_name, first_name, email, salary
WHERE department_id=60;
```

Прочтите сообщение об ошибке и исправьте команду

14. Выполните исправленную команду

```
5 rows inserted.
id    last_name                 first_name                email                 salary
----- ------------------------- ------------------------- --------------------- ---------
103   Hunold                    Alexander                 AHUNOLD               9000
104   Ernst                     Bruce                     BERNST                6000
105   Austin                    David                     DAUSTIN               4800
106   Pataballa                 Valli                     VPATABAL              4800
107   Lorentz                   Diana                     DLORENTZ              4200
```

15. Используйте следующий подзапрос для обновления строки и посмотрите результат

```
UPDATE emp SET salary = (SELECT salary FROM employees WHERE employee_id=100)+1 WHERE
id=103;
SELECT * FROM emp;
```

в результате оклад сотрудника 102 изменился на 24001, что на 1 больше, чем у сотрудника 100 из таблицы employees.

```
103   Hunold                    Alexander                 AHUNOLD               24001
```

Практическое задание 11 решения

1. Создайте таблицу в JDeveloper в соединении postgres ИЛИ oracle. Выберите одно из соединений, с которым будете работать.

```
DROP TABLE emp;
CREATE TABLE emp
  (id  NUMERIC(4) CONSTRAINT emp_id_nn NOT NULL,
   last_name VARCHAR(25),
   first_name VARCHAR(25),
   email  VARCHAR(20),
   salary  NUMERIC(8) DEFAULT 1000);
```

2. Убедитесь в том, что таблица существует и строк в ней нет.

```
SELECT * FROM emp;
```

3. Вставьте данные в таблицу emp.
а) Напишите команду INSERT для вставки в таблицу emp первой строки из ниже приведенных образцов. Не указывайте столбцы в предложении INSERT.
б) Добавьте после команды INSERT в окне JDeveloper команду SELECT * FROM emp; чтобы сразу просматривать результат
в) выполните обе команды

ID	LAST_NAME	FIRST_NAME	EMAIL	SALARY
1	Patel	Ralph	rpatel	2020
2	Dancs	**Betty**	bdancs	1860

```
INSERT INTO emp VALUES (1, 'Patel', 'Ralph', 'rpatel', 2020);
```

4. Вставьте в таблицу EMP данные только двух столбцов **ID** и **FIRST_NAME**, используя вторую строку из вышеуказанных образцов, отредактировав команду INSERT в JDeveloper в окне соединения.
На этот раз укажите столбцы явно в предложении INSERT.

```
INSERT INTO emp (id, first_name) VALUES (2, 'Betty');
```

Убедитесь в том, что данные вставлены в таблицу.
Обратите внимание, что SALARY у Betty равна 1000, хотя вы ее не указывали.
Потому что при создании таблицы emp было указано значение по умолчанию для столбца.

5. Смените фамилию служащей номер 2 на “Dancs”.

```
UPDATE emp SET last_name = 'Dancs' WHERE id = 2;
SELECT * FROM emp;
```

6. Сделайте оклад равным 1000 для всех служащих, имеющих оклад ниже 1000.

```
UPDATE emp SET salary = 1000 WHERE salary < 1000;
0 rows updated.
SELECT * FROM emp;
```

Обратите внимание, что БД вернула сообщение об обновлении нуля строк.
Приложение, работающее с БД, получает от БД информацию о том, сколько строк было изменено.

7. Сделайте оклад равным 1000 для всех служащих, имеющих оклад ниже 2000.

```
UPDATE emp SET salary = 1000 WHERE salary < 2000;
1 row updated.
SELECT * FROM emp;
```

Обратите внимание, что БД вернула сообщение об обновлении одной строки. Однако данные в строке не изменились, так как у Betty оклад был равен 1000. Физически изменения произошли, даже если данные поменялись на те же самые. БД по команде UPDATE не проверяла значение столбца SALARY, она просто выполнила команду.

8. Сделайте оклад равным 1500 для всех служащих, имеющих оклад ниже 2000.

```
UPDATE emp SET salary = 1500 WHERE salary < 2000;
SELECT * FROM emp;
```

Теперь данные поменялись.

9. Выполните команду и посмотрите результат

```
INSERT INTO emp SELECT * FROM emp;
SELECT * FROM emp;
```

Строки в таблице задублированы и указать БД с какой строкой вы хотите работать сложно.

10. Уберем дубликаты с использованием обычной таблицы.

```
CREATE TABLE emp1 AS SELECT DISTINCT * FROM emp;
DROP TABLE emp;
ALTER TABLE emp1 RENAME TO emp;
SELECT * FROM emp;
```

11. Удалите из таблицы `EMP` строку для служащей Betty Dancs.

```
DELETE FROM emp WHERE last_name = 'Dancs';
SELECT * FROM emp;
```

Обратите внимание, что, как и для команды UPDATE, БД вернула количество обработанных командой DELETE строк.
Можно удалить дубликаты без пересоздания таблицы, но придется использовать функционал специфичный для БД конкретного производителя. В БД Oracle для цели удаления можно использовать служебные столбцы ROWID, ROWNUM, В БД postgres ctid, oid (по умолчанию oid не создаётся). Эти служебные столбцы можно использовать, если явно указать их имя.

12. Удалите все строки в таблице emp командой TRUNCATE

```
TRUNCATE TABLE emp;
```

13. Выполните команду:

```
INSERT INTO emp SELECT employee_id, last_name, first_name, email, salary
WHERE department_id=60;
```

Прочтите сообщение об ошибке и исправьте команду

14. Выполните исправленную команду

```
INSERT INTO emp SELECT employee_id, last_name, first_name, email, salary FROM employees
WHERE department_id=60;
5 rows inserted.
SELECT * FROM emp;
```

15. Используйте следующий подзапрос для обновления строки и посмотрите результат

```
UPDATE emp SET salary = (SELECT salary FROM employees WHERE employee_id=100)+1 WHERE
id=103;
SELECT * FROM emp;
```

в результате оклад сотрудника 102 изменился на 24001, что на 1 больше, чем у сотрудника 100 из таблицы employees.

Практическое задание 13

1. Создайте запрос для вывода фамилии, номера отдела и оклада всех служащих, чей номер отдела и оклад совпадают с номером отдела и окладом любого служащего, зарабатывающего комиссионные

```
last_name       department_id salary
--------------- ------------- ---------
Russell         80            14000
Partners        80            13500
Errazuriz       80            12000
Cambrault       80            11000
Zlotkey         80            10500
Tucker          80            10000
Bernstein       80            9500
...
Johnson         80            6200
 34 rows selected
```

2. Выведите фамилию, название отдела и оклад всех служащих, чей оклад и комиссионные совпадают с окладом и комиссионными любого служащего, работающего в отделе, местоположение которого (location_id) 1700.

```
last_name       department_name        salary
--------------- ---------------------- ---------
King            Executive              24000
Kochhar         Executive              17000
De Haan         Executive              17000
Hunold          IT                     9000
Greenberg       Finance                12008
Faviet          Finance                9000
...
Gietz           Accounting             8300
 36 rows selected
```

3. Создайте запрос для вывода фамилии, даты найма и оклада всех служащих, которые получают такой же оклад и такие же комиссионные, как Kochhar. Данные о сотруднике Kochhar исключите из выборки.

```
last_name       hire_date                 salary
--------------- ------------------------- ---------
De Haan         2001-01-13                17000
```

4. Выведите фамилию, должность и оклад всех служащих, оклад которых превышает оклад каждого клерка торгового менеджера (`JOB_ID = 'SA_MAN'`). Отсортируйте результаты по убыванию окладов.

```
last_name       job_id     salary
--------------- ---------- ---------
King            AD_PRES    24000
Kochhar         AD_VP      17000
De Haan         AD_VP      17000
```

5. Выведите номера, фамилии и отделы служащих, живущих в городах, названия которых начинаются с буквы T.

```
employee_id last_name       department_id
----------- --------------- -------------
201         Hartstein       20
202         Fay             20
```

6. Напишите запрос для нахождения всех сотрудников, которые зарабатывают больше среднего оклада по их отделу. Выведите фамилию, оклад, номер отдела и средний оклад по отделу. Отсортируйте результаты по средней зарплате.

```
last_name       salary    department_id avg
--------------- --------- ------------- ---------------------
Dilly           3600      50            3475.5555555555555556
Bell            4000      50            3475.5555555555555556
Kaufling        7900      50            3475.5555555555555556
Sarchand        4200      50            3475.5555555555555556
Everett         3900      50            3475.5555555555555556
...
King            24000     90            19333.333333333333
```

```
 38 rows selected
```

7. Найдите всех сотрудников, не являющихся руководителями.
(а) Первый раз выполните это с помощью оператора `NOT EXISTS`.

```
last_name
---------------
Ernst
Austin
Pataballa
Lorentz
...
Gietz
 89 rows selected
```

б) Замените `NOT EXISTS` на `NOT IN`
Вы получите:

```
no rows selected
```

Почему?

8. Выведите фамилии сотрудников, зарабатывающих меньше среднего оклада по их отделу

```
last_name
---------------
Kochhar
De Haan
Austin
Pataballa
...
Gietz
 65 rows selected
```

9. Выведите фамилии сотрудников, у которых есть коллеги по отделу, которые были приняты на работу позже, но имеют более высокий оклад.

```
last_name
---------------
De Haan
Austin
Pataballa
Lorentz
...
Grant
 70 rows selected
```

10. Выведите номера, фамилии и наименования отделов всех сотрудников.
Используйте скалярный подзапрос в команде `SELECT` для вывода наименований отделов

```
employee_id last_name       department
----------- --------------- ----------------------
205         Higgins         Accounting
206         Gietz           Accounting
200         Whalen          Administration
100         King            Executive
...
178         Grant
 107 rows selected
```

11. Напишите запрос для вывода наименований отделов и фондов заработной платы отделов, размер которых больше 1/8 от общего фонда заработной платы всей компании. Используйте предложение `WITH` для создания запроса. Назовите запрос `summary`.

```
department_name        dept_total
---------------------- ----------
Sales                  304500
Shipping               156400
```

Практическое задание 13 решения

1. Создайте запрос для вывода фамилии, номера отдела и оклада всех служащих, чей номер отдела и оклад совпадают с номером отдела и окладом любого служащего, зарабатывающего комиссионные

```
SELECT last_name, department_id, salary
      FROM employees
      WHERE (salary, department_id) IN
              (SELECT salary, department_id
               FROM employees
               WHERE commission_pct IS NOT NULL);
```

2. Выведите фамилию, название отдела и оклад всех служащих, чей оклад и комиссионные совпадают с окладом и комиссионными любого служащего, работающего в отделе, местоположение которого (location_ID) 1700.

```
SELECT e.last_name, d.department_name, e.salary
 FROM   employees e JOIN departments d USING (department_id)
 WHERE   (salary, COALESCE(commission_pct,0))  IN
  (SELECT salary, COALESCE(commission_pct,0)
   FROM employees e JOIN departments d USING (department_id)
   WHERE d.location_id = 1700);
```

3. Создайте запрос для вывода фамилии, даты найма и оклада всех служащих, которые получают такой же оклад и такие же комиссионные, как Kochhar. Не выводите данные о сотруднике Kochhar.

```
SELECT last_name, hire_date, salary
FROM   employees
WHERE  (salary, COALESCE(commission_pct,0)) IN
             (SELECT salary, COALESCE(commission_pct,0)
              FROM employees
              WHERE last_name = 'Kochhar')
AND last_name != 'Kochhar';
```

4. Выведите фамилию, должность и оклад всех служащих, оклад которых превышает оклад каждого клерка торгового менеджера (`JOB_ID = 'SA_MAN'`). Отсортируйте результаты по убыванию окладов.

```
SELECT last_name, job_id, salary
    FROM   employees
    WHERE  salary > ALL
                (SELECT salary
                 FROM employees
    WHERE  job_id = 'SA_MAN')
    ORDER BY salary DESC;
```

5. Выведите номера, фамилии и отделы служащих, живущих в городах, названия которых начинаются с буквы T.

```
SELECT employee_id, last_name, department_id
  FROM   employees
  WHERE  department_id IN (SELECT department_id
                           FROM departments
                           WHERE location_id IN
                                (SELECT  location_id
                                 FROM locations
                                 WHERE city LIKE 'T%'));
```

6. Напишите запрос для нахождения всех сотрудников, которые зарабатывают больше среднего оклада по их отделу. Выведите фамилию, оклад, номер отдела и средний оклад по отделу. Отсортируйте результаты по средней зарплате. Используйте псевдонимы для выбираемых столбцов, приведенные в приводимом ниже примере выходных результатах.

```
SELECT e.last_name ename, e.salary salary,
             e.department_id deptno, AVG(a.salary) dept_avg
  FROM   employees e, employees a
  WHERE  e.department_id = a.department_id
   AND   e.salary > (SELECT AVG(salary)
                     FROM   employees
                     WHERE  department_id = e.department_id )
  GROUP BY e.last_name, e.salary, e.department_id
  ORDER BY AVG(a.salary);
```

7. Найдите всех сотрудников, не являющихся руководителями.
(а) Первый раз выполните это с помощью оператора NOT EXISTS.

```
SELECT a.last_name
   FROM employees a
   WHERE NOT EXISTS (SELECT 'X'
                     FROM employees b
                     WHERE b.manager_id = a.employee_id);
```

б) Замените NOT EXISTS на NOT IN.

```
SELECT a.last_name
 FROM   employees a
 WHERE  a.employee_id
   NOT IN (SELECT b.manager_id
           FROM employees b);
```

Почему? Подзапрос возвращает неопределенное значение (NULL), поэтому весь запрос не возвращает строк. Причина в том, что сравнение производится со всеми значениями, в том числе с неопределенным, что всегда приводит к значению NULL для всей операции сравнения. Если возможно присутствие значение NULL в результирующем наборе, то никогда не используйте оператор NOT IN взамен NOT EXISTS.

8. Выведите фамилии сотрудников, зарабатывающих меньше среднего оклада по их отделу

```
SELECT  last_name
  FROM  employees a
  WHERE a.salary < (SELECT AVG(b.salary)
                    FROM employees b
                    WHERE b.department_id = a.department_id);
```

9. Выведите фамилии сотрудников, у которых есть коллеги по отделу, которые были приняты на работу позже, но имеют более высокий оклад.

```
SELECT  last_name
         FROM    employees a
         WHERE EXISTS (SELECT 'X'
                       FROM employees b
                       WHERE b.department_id =
                             a.department_id
                       AND b.hire_date > a.hire_date
                       AND b.salary > a.salary);
```

10. Выведите номера, фамилии и наименования отделов всех сотрудников.
Используйте скалярный подзапрос в команде SELECT для вывода наименований отделов

```
SELECT employee_id, last_name,
               (SELECT department_name
                  FROM departments d
                  WHERE   e.department_id = d.department_id ) department
 FROM employees e
 ORDER BY department;
```

11. Напишите запрос для вывода наименований отделов и фондов заработной платы отделов, размер которых больше 1/8 от общего фонда заработной платы всей компании. Используйте предложение WITH для создания запроса. Назовите запрос summary.

```
WITH
summary AS (
  SELECT d.department_name, SUM(e.salary) AS dept_total
  FROM employees e JOIN departments d USING (department_id)
  GROUP BY d.department_name )
SELECT department_name, dept_total
FROM summary
WHERE dept_total > (SELECT SUM(dept_total) * 1/8 FROM summary)
ORDER BY dept_total DESC;
```

Практическое задание 14

1. Создайте представление dept60 в postgres и oracle, содержащее номер служащего, фамилию служащего и номер отдела для всех служащих отдела 60. Назовите столбцы представления EMPNO, EMPLOYEE и DEPTNO. Для обеспечения безопасности запретите операцию перевода служащего в другой отдел через представление.

```
CREATE VIEW dept60 AS
   SELECT   employee_id empno, last_name employee,
            department_id deptno
   FROM     employees
   WHERE    department_id = 60
   WITH CHECK OPTION;
```

2. Проверьте представление. Попробуйте сменить номер отдела служащего по фамилии Hunold на 80.

```
UPDATE   dept60
SET      deptno = 80
WHERE    employee = 'Hunold';
```

Вы получите ошибку

```
SQL Error: ОШИБКА: новая строка нарушает ограничение-проверку для представления
"dept60"
```

3. Создайте представление e_v в любом соединении. Включите номер служащего, фамилию служащего и номер отдела из таблицы EMPLOYEES. Смените заголовок столбца с фамилией служащего на EMPLOYEE.

```
CREATE OR REPLACE VIEW e_v AS
    SELECT employee_id, last_name employee, department_id
    FROM employees;
```

4. Убедитесь в возможности использования представления e_v. Выведите его содержимое.

```
SELECT count(*) FROM e_v;
```

5. Выведите фамилии всех служащих и номера их отделов.

```
SELECT   employee, department_id
FROM     e_v;
```

12. Необходимо создать последовательность для столбца главного ключа таблицы DEPT. Последовательность должна начинаться с 300 и иметь максимальное значение 1000. Шаг приращения значений – 10. Назовите последовательность DEPT_ID_SEQ.
Создайте последовательность В JDeveloper в соединении oracle и соединении postgres

```
CREATE SEQUENCE dept_id_seq
  START WITH 300
  INCREMENT BY 10
  MAXVALUE 1000;
```

13. В JDeveloper в соединении oracle дайте команды

```
SELECT dept_id_seq.CURRVAL FROM DUAL;
INSERT INTO dept (department_id,department_name)
  VALUES (dept_id_seq.nextval, 'Education');
INSERT INTO dept (department_id,department_name)
  VALUES (dept_id_seq.nextval, 'Administration');
```

Обратите внимание на ошибку после первой команды

```
ORA-08002: sequence DEPT_ID_SEQ.CURRVAL is not yet defined in this session
```

14. В JDeveloper в соединении oracle проверьте, какие данные вставлены

```
SELECT * FROM dept WHERE department_id>=300;
SELECT dept_id_seq.CURRVAL FROM DUAL;
```

15. Синтаксис обращения к последовательностям в БД postgres отличается от синтаксиса БД Oracle
В JDeveloper в соединении postgres дайте команды

```
select currval('dept_id_seq');
INSERT INTO dept (department_id,department_name)
  VALUES (nextval('dept_id_seq'), 'Education');
INSERT INTO dept (department_id,department_name)
  VALUES (nextval('dept_id_seq'), 'Administration');
```

```
SELECT * FROM dept WHERE department_id>=300;
select currval('dept_id_seq');
```

Обратите внимание на ошибку после первой команды

```
SQL Error: ОШИБКА: текущее значение (currval) для последовательности "dept_id_seq"
ещё не определено в этом сеансе
```

16. Измените шаг последовательности на 1 в БД postgres

```
alter sequence dept_id_seq increment by 1;
select currval('dept_id_seq');
select nextval('dept_id_seq');
```

Обратите внимание, что последовательность продолжила счёт и не сбросилась.

17. Измените шаг последовательности на 1 в БД Oracle

```
alter sequence dept_id_seq increment by 1;
SELECT dept_id_seq.CURRVAL FROM DUAL;
SELECT dept_id_seq.NEXTVAL FROM DUAL;
```

Обратите внимание, что последовательность продолжила счёт и не сбросилась.