

Федеральное агентство связи

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Сибирский государственный университет телекоммуникаций и информатики»
(СибГУТИ)

В.И. Мейкшан

Основы языка SQL в примерах и задачах

Учебно-методическое пособие

Новосибирск – 2013

Мейкшан В.И. Основы языка SQL в примерах и задачах: Учебно-методическое пособие / Сибирский государственный университет телекоммуникаций и информатики; Кафедра телекоммуникационных сетей и вычислительных средств. – Новосибирск, 2013. – 25 с. – 40 с.

Аннотация. Учебно-методическое пособие предназначено для проведения лабораторных и практических занятий по дисциплине «Базы данных». Представленный материал посвящен изучению основ языка SQL на примере СУБД Oracle 10g. В краткой форме излагаются основные синтаксические правила построения наиболее распространенных типов запросов на выборку из БД, модификацию данных и создание представлений. Эти правила иллюстрируются большим количеством примеров, которые могут быть полезными на этапе начального освоения материала по использованию операторов языка SQL. С целью закрепления полученных знаний предлагаются задания различной степени сложности для самостоятельной работы.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	4
1. ВВЕДЕНИЕ В СУБД ORACLE 10G	5
Общие сведения	5
Инсталляция	5
2. КРАТКИЕ СВЕДЕНИЯ ОБ ИНТЕРФЕЙСЕ СУБД ORACLE 10G XE ..	6
Группа ADMINISTRATION	6
Группа OBJECT BROWSER	7
Группа SQL	7
Группа UTILITIES	8
3. ОПИСАНИЕ БАЗЫ ДАННЫХ HR	10
Структурная схема базы данных HR	10
Управление базой данных с помощью Object Browser	13
4. ИЗУЧЕНИЕ ЭЛЕМЕНТОВ ЯЗЫКА SQL	16
4.1. СРЕДСТВА ЯЗЫКА SQL ДЛЯ МАНИПУЛИРОВАНИЯ ДАННЫМИ ...	17
4.1.1. <i>ВЫБОРКА ДАННЫХ С ПОМОЩЬЮ КОМАНДЫ SELECT</i>	17
Отображение заданных столбцов из одиночной таблицы	18
Выборка записей по заданному условию	19
Сортировка результатов запроса	20
4.1.1.1. <i>ВСТРОЕННЫЕ ФУНКЦИИ ЯЗЫКА SQL</i>	21
4.1.1.2. <i>ГРУППОВЫЕ (АГРЕГАТНЫЕ) ФУНКЦИИ</i>	22
4.1.1.3. <i>ЗАПРОСЫ С ГРУППИРОВКОЙ</i>	23
4.1.1.4. <i>ВЛОЖЕННЫЕ ЗАПРОСЫ (ПОДЗАПРОСЫ)</i>	24
4.1.1.5. <i>МНОГОТАБЛИЧНЫЕ ЗАПРОСЫ С ОПЕРАЦИЕЙ JOIN</i>	26
4.1.2. <i>РАБОТА С РЕЗУЛЬТАТАМИ НЕСКОЛЬКИХ ЗАПРОСОВ</i>	30
4.1.3. <i>ДОБАВЛЕНИЕ (ВСТАВКА) НОВЫХ ЗАПИСЕЙ В ТАБЛИЦУ С ПОМОЩЬЮ КОМАНДЫ INSERT</i>	31
4.1.4. <i>ИЗМЕНЕНИЕ ДАННЫХ В ТАБЛИЦАХ С ПОМОЩЬЮ ОПЕРАТОРОВ UPDATE И DELETE</i>	34
4.1.5. <i>ИСПОЛЬЗОВАНИЕ ОПЕРАТОРА MERGE</i>	35
4.2. СРЕДСТВА ЯЗЫКА SQL ДЛЯ ОПРЕДЕЛЕНИЯ ДАННЫХ	38
4.2.1. <i>РАБОТА С ПРЕДСТАВЛЕНИЯМИ</i>	38
ВОПРОСЫ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ	39
СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ	41

ВВЕДЕНИЕ

Основными целями изучения дисциплины «Базы данных» являются:

- обзор средств СУБД в области обработки данных;
- работа с данными в различных средах СУБД;
- разработка баз данных и приложений в различных средах;
- разработка сопроводительной документации к программным продуктам в соответствии с требованиями нормативных документов.

Дисциплина «Базы данных» включает в себя также изучение теоретических основ проектирования баз данных, особенностей систем управления базами данных разных классов; новейших направлений в области проектирования и организации процессов обработки данных баз данных, в частности, проблем сжатия больших информационных массивов.

В этой методической разработке в качестве основы технологии баз данных используется СУБД ORACLE. Рассматривается демонстрационная база данных HR (Human Resources), которая иллюстрирует реализацию учета данных о сотрудниках в рамках крупной компании. Применительно к этой БД подготовлены готовые примеры запросов к БД и составлены задания для самостоятельной работы студентов с целью практического изучения элементов языка SQL.

Процесс изучения основных директив языка SQL с помощью методических указаний построен таким образом, что вначале по отдельной директиве даются краткие теоретические сведения и примеры ее использования, а затем – практические задания для закрепления полученных знаний.

Перед тем, как непосредственно заняться изучением языка SQL в версии СУБД ORACLE, студент должен освоить средства пользовательского интерфейса, которые предоставляют доступ к основному инструментарию этой СУБД.

1. ВВЕДЕНИЕ В СУБД ORACLE 10G

Общие сведения

Oracle 10g XE (Express Edition) — это свободно распространяемая (бесплатная) версия СУБД в составе целой линейки программных продуктов, выпускаемых корпорацией Oracle для поддержки технологий баз данных. Она прекрасно подходит для использования в учебных заведениях с целью обучения студентов современным методам работы с реляционными БД.

Версия Oracle XE существует в нескольких вариантах для платформ Windows и Linux (Debian, Mandriva, Novell, Red Hat и Ubuntu). Программный пакет Oracle 10g XE можно скачать с официального Web-сайта корпорации и легко установить на свой ПК.

Версия Oracle Database 10g XE создана на основе исходного программного кода СУБД Oracle Database 10g Release 2 и полностью совместима с семейством программных продуктов Oracle Database. Предоставляются те же средства SQL и PL/SQL, что и во всех остальных версиях Oracle Database 10g, а также широкий спектр программных интерфейсов. Например, имеется полная поддержка создания и развертывания приложений, работающих на платформах Java, .NET, PHP и Windows.

Для версии XE существует ряд ограничений:

- поддерживается база данных размером не более 4 Гбайт;
- на одном компьютере может быть запущен только один экземпляр СУБД;
- при наличии на сервере нескольких процессоров СУБД Oracle XE использует только один из них;
- независимо от объема доступной оперативной памяти, СУБД Oracle XE будет использовать не более 1 Гбайт этих ресурсов.

Несмотря на эти ограничения, на основе СУБД Oracle XE можно создавать полноценные приложения для решения широкого круга реальных задач. Если по мере роста базы данных и увеличения количества пользователей СУБД Oracle XE перестает справляться с решаемыми задачами, либо достигается предельный для Oracle XE размер БД, то не составит особого труда перевести задачи (без изменения кода приложений) на одну из платных версий Oracle с более широкими возможностями.

Инсталляция

В случае операционной системы типа Windows инсталлятором является дистрибутив Oracle 10G XE в виде единственного файла OracleXE.exe. Процесс установки максимально упрощен и управление этим процессом практически невозможно. База данных и программный код устанавливаются в одну папку (которую можно задать), затем требуется ввести пароли для пользователей SYS и SYSTEM.

Собственно, на этом все настройки заканчиваются, и начинается инсталляция, которая длится несколько минут. Перед завершением инсталляции рекомендуется выбрать режим автоматического запуска браузера. В результате будет предоставлена возможность войти в систему под именем SYSTEM (или

SYS) с правами менеджера управления базой данных.

2. КРАТКИЕ СВЕДЕНИЯ ОБ ИНТЕРФЕЙСЕ СУБД ORACLE 10g XE

Процесс управления базой данных предельно прост и осуществляется через Web-интерфейс. Подобный подход облегчает администрирование, особенно в случае дистанционного управления через Интернет.

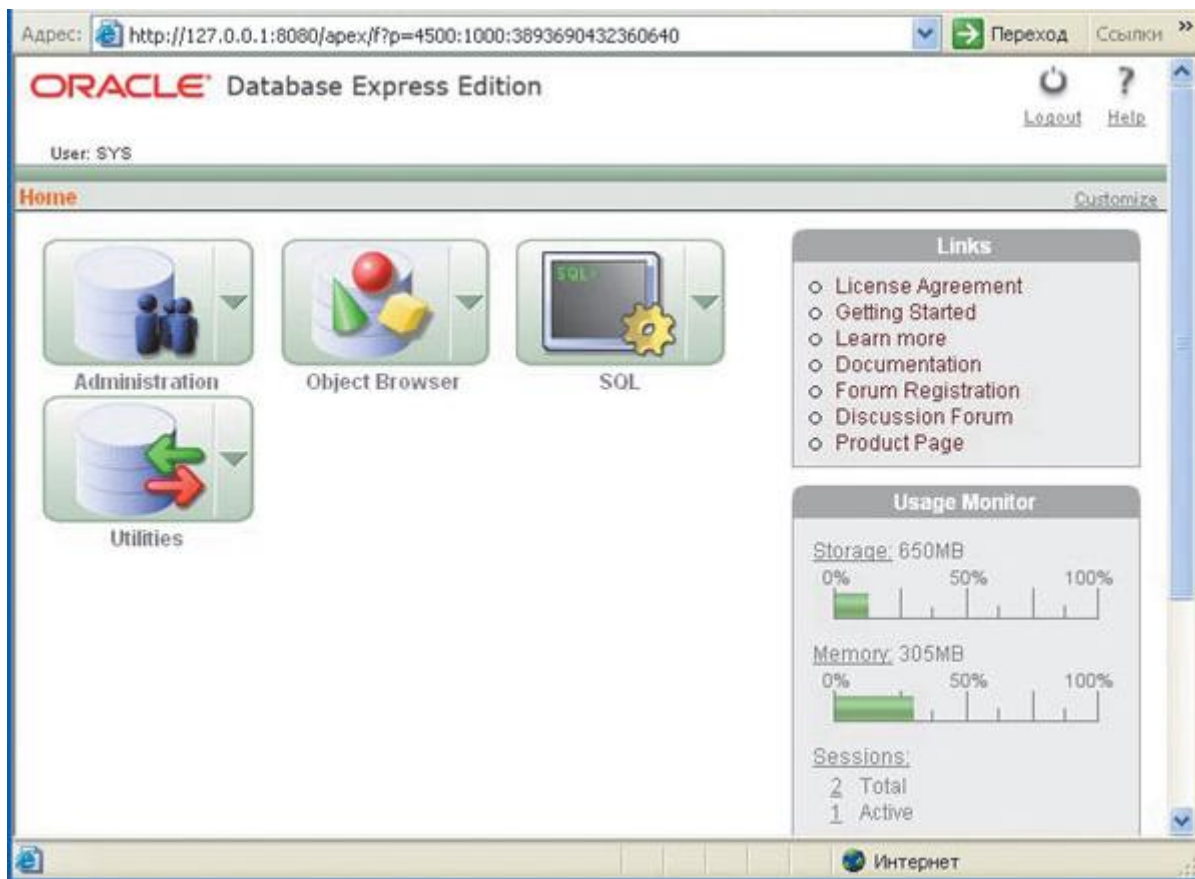


Рис. 5. Главная страница Web-интерфейс СУБД Oracle 10g XE

Сразу после входа пользователя в систему открывается домашняя (главная) страница (Home), на которой содержится четыре пиктограммы с кнопкой выпадающего меню для каждой из них (рис. 5). Эти пиктограммы соответствуют следующим группам инструментов: Administration, Object Browser, SQL и Utilities.

Группа ADMINISTRATION

Этот набор средств предназначен для администрирования и мониторинга базы:

- **Storage** – просмотр информации о файлах базы данных и табличных пространствах.
- **Memory** – просмотр распределения памяти с возможностью настройки размера областей SGA (System Global Area) и PGA (Process Global Area).

- **Database Users** – инструмент для просмотра списка пользователей базы данных с возможностью простейшего администрирования (создание, удаление, блокировка учетной записи, смена пароля, основные привилегии).
- **Monitor** – мониторинг экземпляра. В частности, можно просматривать список сессий, блокировок и открытых курсоров. Из списка сессий можно вызывать страницу с детализированной информацией по каждой сессии. На странице с детализированной информацией имеется кнопка для принудительного закрытия сессии. Кроме того, предусмотрены дополнительные виды анализа, в том числе поиск наиболее ресурсоемких запросов и операций, выполняемых длительное время.
- **About Database** – просмотр настроек экземпляра.

Группа OBJECT BROWSER

Данный инструмент позволяет просматривать существующие объекты в базе данных (таблицы, представления, индексы, процедуры, функции, триггеры и др.), а также манипулировать ими. Предоставляемые возможности достаточно широки, причем большинство операций выполняются с помощью визуальных средств, которые не требуют знания языка SQL. По этой причине инструмент очень удобен для начинающих пользователей.

Группа SQL

Сюда входят средства для формирования и последующего выполнения SQL-команд:

- **SQL Commands** – для работы с одиночными операторами (директивами) языка SQL и анонимными блоками PL/SQL (рис. 5). По умолчанию у этого инструмента включена опция Autocommit, что приводит к автоматическому выполнению команды COMMIT и фиксации результата после каждой операции. Пользователь может сохранить любую введенную команду (кнопка **Save**), причем при сохранении задается имя и краткое описание. Сохраненные операторы могут быть впоследствии загружены в редактор из закладки «Saved SQL». В случае выполнения запроса отображаются возвращаемые им данные и план выполнения запроса.
- **Query Builder** – визуальный построитель запросов, принцип работы которого является стандартным для утилит подобного типа и напоминает конструктор запросов в MS Access. Предоставляются возможности задавать простейшие условия фильтрации и правила сортировки по каждому из полей в составе запроса. После завершения визуального построения запроса можно просмотреть и скопировать полученный текст директивы на языке SQL, выполнить запрос и просмотреть результаты его работы. Такое визуальное средство может представлять интерес не столько для разработчиков, сколько для постановщиков задач и специалистов, отвечающих за тестирование и техническую поддержку прикладных продуктов.

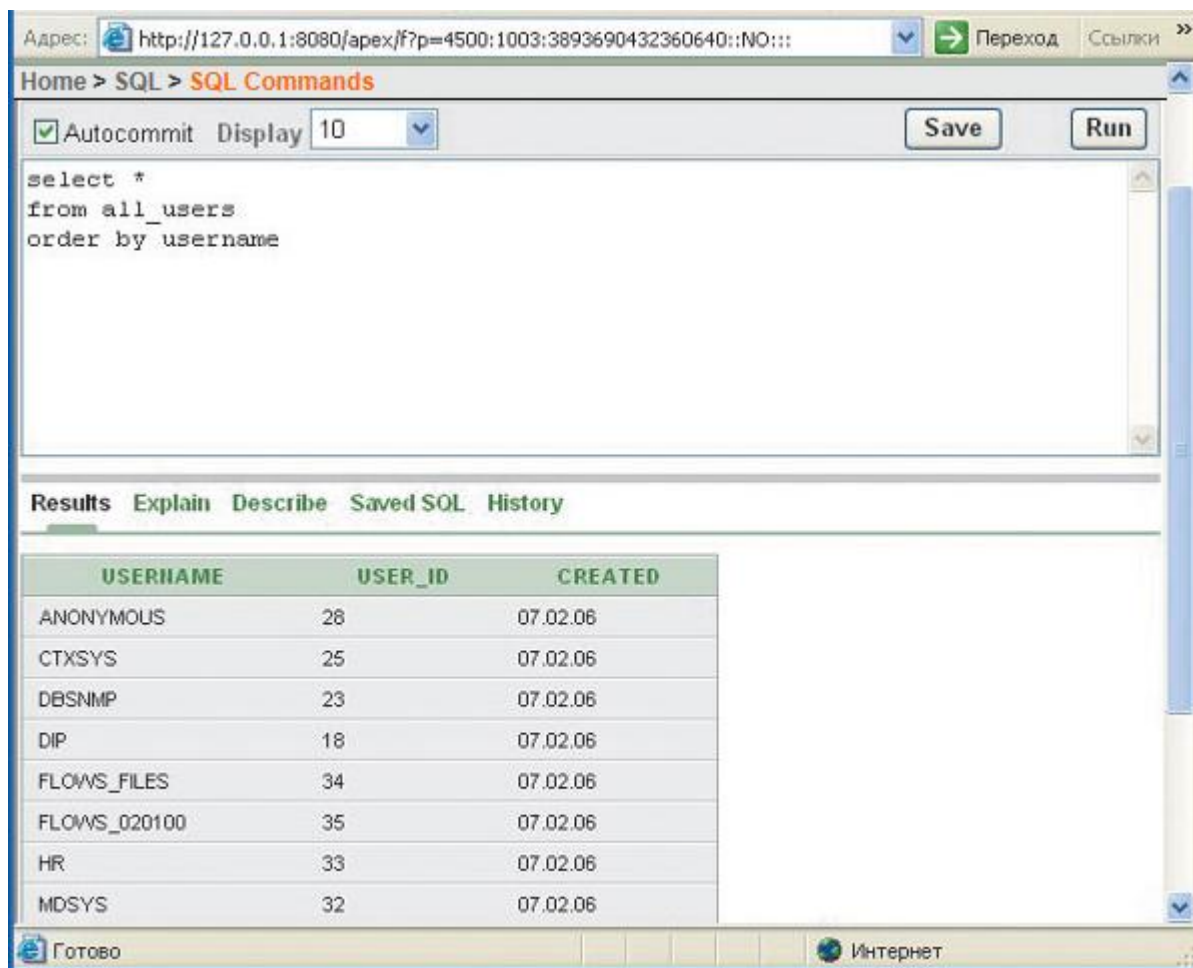


Рис. 5. Ввод запроса к базе данных и результат его выполнения

- **SQL Scripts** – средство для работы с SQL-скриптами, позволяет создавать и редактировать эти объекты, а затем запускать их на выполнение. Полученные результаты можно сохранять под заданными именами, чтобы впоследствии просмотреть их и провести анализ.

Группа UTILITIES

Эта группа содержит различные вспомогательные утилиты, которые бывают очень полезными при работе с базой данных:

- страница **Data Load/Unload** – инструменты для загрузки и выгрузки данных. Поддерживается работа с различными форматами данных, которые могут размещаться в файле или копироваться через буфер обмена: текстовые данные (Text Data), табличные данные (Spreadsheet Data) и формат XML.
- страница **Generate DDL** – инструмент генерации скриптов на языке DDL для указанных объектов схемы данных.
- страница **Object Reports** – инструменты для генерации служебных отчетов, которые необходимы при администрировании и анализе базы данных.

Быстрой навигации по графическому интерфейсу клиентской части СУБД

Oracle 10g XE помогает цепочка ссылок, которая размещается в верхней части каждой страницы. Например, на странице ввода запроса к базе данных (рис. 5) располагается цепочка **Home > SQL > SQL Commands**.

Хвостовая часть цепочки соответствует текущей странице, а любой из предшествующих элементов можно использовать для прямого перехода на другую страницу, которая в иерархии построения интерфейса относится к более высокому уровню. В частности, самая левая ссылка (Home) представляет главную страницу, т.е. вершину иерархии.

3. ОПИСАНИЕ БАЗЫ ДАННЫХ HR

Структурная схема базы данных HR

В комплекте СУБД Oracle XE имеется встроенная демонстрационная база данных HR (Human Resources), которая предоставляет существенные удобства для практического изучения языка SQL. Эта БД иллюстрирует пример учета данных о сотрудниках некоторой крупной компании. Отделы (департаменты) компании, состоящей из многочисленных филиалов, размещаются в разных странах света.

Каждый сотрудник имеет идентификационный номер, адрес электронной почты, код занимаемой должности и соответствующую зарплату. Помимо стабильных ежемесячных выплат, некоторым сотрудникам полагается комиссионная надбавка.

Каждая должность в компании имеет идентификационный код, который связывает ее с полным названием и «вилкой» по заработной плате.

Сотрудники приписаны к департаментам, каждый из которых идентифицируется уникальным номером, а также характеризуется полным названием и кодом местоположения. Полное описание каждого местоположения, где находятся департаменты (подразделения) компании, включает в себя почтовый адрес с указанием почтового индекса, улицы, города, штата (или провинции) и идентификационного кода страны.

Перечисленные данные хранятся в виде следующих таблиц:

- REGIONS – список географических регионов (Europe, Asia и др.);
- COUNTRIES – список стран (с привязкой каждой страны к соответствующему региону);
- LOCATIONS – данные о местах расположения департаментов компании;
- DEPARTMENTS – данные о департаментах (подразделениях), из которых состоит компания;
- JOBS – данные о должностях, которые имеются в компании;
- EMPLOYEES – данные о сотрудниках, которые работают в компании;
- JOB_HISTORY – данные, которые характеризуют «трудовую биографию» сотрудников компании.

Подробное описание структуры указанных таблиц представлено ниже.

Таблица 1

REGIONS

Названия столбцов	Типы данных	Комментарии
REGION_ID	NUMBER	Код региона
REGION_NAME	VARCHAR2(25)	Название региона

Таблица 2

COUNTRIES

Названия столбцов	Типы данных	Комментарии
COUNTRY_ID	CHAR(2)	Код страны
COUNTRY_NAME	VARCHAR2(40)	Название страны
REGION_ID	NUMBER	Код региона

Таблица 3

LOCATIONS

Названия столбцов	Типы данных	Комментарии
LOCATION_ID	NUMBER(4,0)	Код местоположения
STREET_ADDRESS	VARCHAR2(40)	Почтовый адрес
POSTAL_CODE	VARCHAR2(12)	Почтовый индекс
CITY	VARCHAR2(30)	Название города
STATE_PROVINCE	VARCHAR2(25)	Название штата (провинции)
COUNTRY_ID	CHAR(2)	Код страны

Таблица 4

DEPARTMENTS

Названия столбцов	Типы данных	Комментарии
DEPARTMENT_ID	NUMBER(4,0)	Код департамента
DEPARTMENT_NAME	VARCHAR2(30)	Название департамента
MANAGER_ID	NUMBER(6,0)	Код руководителя
LOCATION_ID	NUMBER(4,0)	Код местоположения

Таблица 5

JOBS

Названия столбцов	Типы данных	Комментарии
JOB_ID	VARCHAR2(10)	Код должности
JOB_TITLE	VARCHAR2(35)	Название должности
MIN_SALARY	NUMBER(6,0)	Минимальная зарплата
MAX_SALARY	NUMBER(6,0)	Максимальная зарплата

Таблица 6

EMPLOYEES

Названия столбцов	Типы данных	Комментарии
EMPLOYEE_ID	NUMBER(6,0)	Код сотрудника
FIRST_NAME	VARCHAR2(20)	Имя сотрудника
LAST_NAME	VARCHAR2(25)	Фамилия сотрудника
EMAIL	VARCHAR2(25)	Адрес электронной почты
PHONE_NUMBER	VARCHAR2(20)	Номер телефона
HIRE_DATE	DATE	Дата приема на работу
JOB_ID	VARCHAR2(10)	Код должности
SALARY	NUMBER(8,2)	Зарплата сотрудника
COMMISSION_PCT	NUMBER(2,2)	Комиссионная надбавка
MANAGER_ID	NUMBER(6,0)	Код прямого руководителя
DEPARTMENT_ID	NUMBER(4,0)	Код департамента

Таблица 7

JOB_HISTORY

Названия столбцов	Типы данных	Комментарии
EMPLOYEE_ID	NUMBER(6,0)	Код сотрудника
START_DATE	DATE	Начальная дата
END_DATE	DATE	Конечная дата
JOB_ID	VARCHAR2(10)	Код должности
DEPARTMENT_ID	NUMBER(4,0)	Код департамента

Логические связи между рассмотренными таблицами показаны на рис. 6. Все связи относятся к типу «один ко многим» и организованы по схеме «первичный ключ (Primary Key, PK) → вторичный ключ (Foreign Key, FK)» с использованием общих столбцов в главной и подчиненной таблицах.

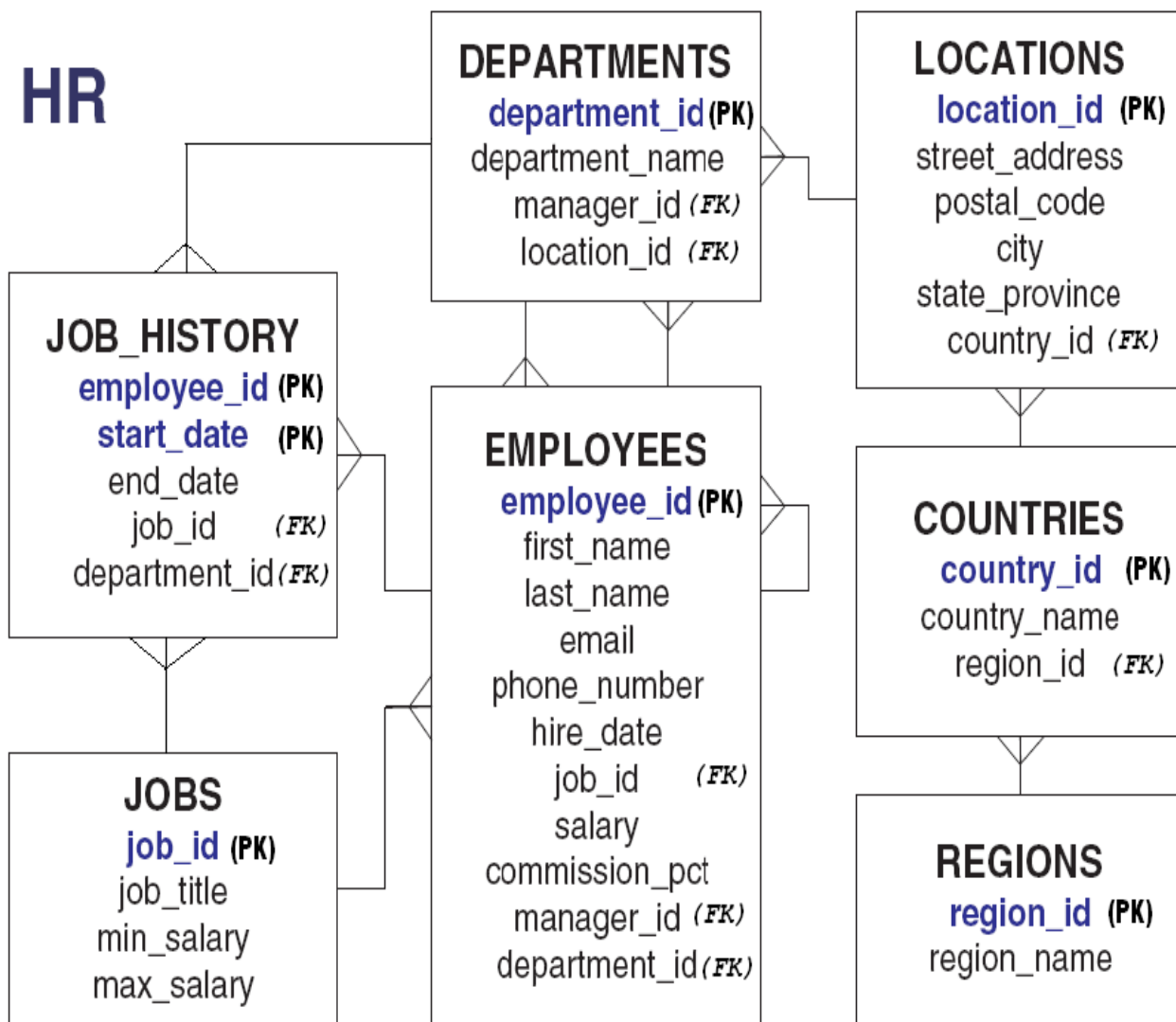


Рис. 6. Логическая схема базы данных HR

Управление базой данных с помощью Object Browser

Инструменты Object Browser позволяют создавать и изменять отдельные объекты базы данных. Например, с помощью Object Browser можно создать новую таблицу или изменить существующую таблицу путем добавления и удаления колонок, включения дополнительных ограничений и т.п. В простейшем случае можно просмотреть объекты, которые в настоящее время присутствуют в составе БД, и по каждому из них получить всю детальную информацию.

При работе с Object Browser нужно, в первую очередь, выбрать тип интересующих объектов. Например, в случае базы данных HR при выборе типа **Tables** на левой панели окна Object Browser (рис. 7) будет показан список перечисленных ранее таблиц. Если в этом списке отметить конкретный элемент, то на вкладках правой панели (Table, Data, Indexes, Model и др.) будут показаны соответствующие сведения, которые относятся к выбранной таблице.

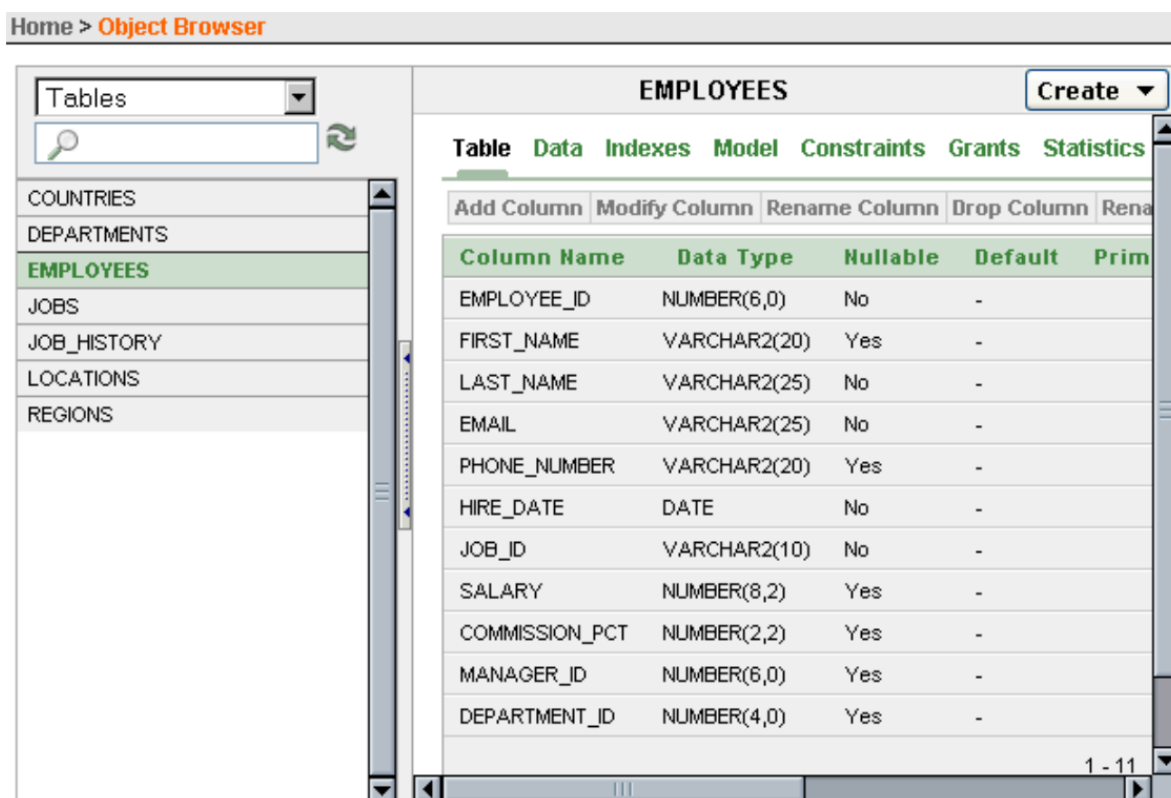


Рис. 7. Окно работы с Object Browser (вкладка **Table**)

Например, на рис. 7 показана структура таблицы EMPLOYEES. Если же перейти на вкладку **Data**, то правая панель будет отображать данные из этой таблицы (рис. 8).










EMPLOYEES										
Table	Data	Indexes	Model	Constraints	Grants	Statistics	UI Defaults	Triggers	Dependencies	SQL
Query	Count Rows	Insert Row								
EDIT	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID			
	100	Steven	King	SKING	515.123.4567	17.06.87	AD_PRES			
	101	Neena	Kochhar	NKOCHHAR	515.123.4568	21.09.89	AD_VP			
	102	Lex	De Haan	LDEHAAN	515.123.4569	13.01.93	AD_VP			
	103	Alexander	Hunold	AHUNOLD	590.423.4567	03.01.90	IT_PROG			
	104	Bruce	Ernst	BERNST	590.423.4568	21.05.91	IT_PROG			
	105	David	Austin	DAUSTIN	590.423.4569	25.06.97	IT_PROG			
	106	Valli	Pataballa	VPATABAL	590.423.4560	05.02.98	IT_PROG			
	107	Diana	Lorentz	DLORENTZ	590.423.5567	07.02.99	IT_PROG			
	108	Nancy	Greenberg	NGREENBE	515.124.4569	17.08.94	FI_MGR			

Рис. 8. Окно работы с Object Browser (вкладка **Data**)

Работа с построителем запросов (Query Builder)

Query Builder – это удобный графический инструмент для конструирования запросов к БД. Благодаря использованию интуитивно понятного интерфейса построение не очень сложных запросов в среде Query Builder становится достаточно легким занятием даже для новичка, не имеющего никаких знаний о языке SQL. Поэтому построитель запросов оказывается весьма удобным средством на начальном этапе изучения SQL, особенно для пользователей, которые не имеют опыта работы с реляционными БД.

Левая панель на странице построителя запросов (рис. 9) содержит список доступных таблиц. Щелчок левой кнопкой мыши на элементе этого списка приводит к тому, что в верхней части правой панели появляется условное графическое изображение (пиктограмма) выбранной таблицы с перечнем доступных полей. Отметками в левом вертикальном ряду пиктограммы нужно выделить те поля, которые будут участвовать в запросе.

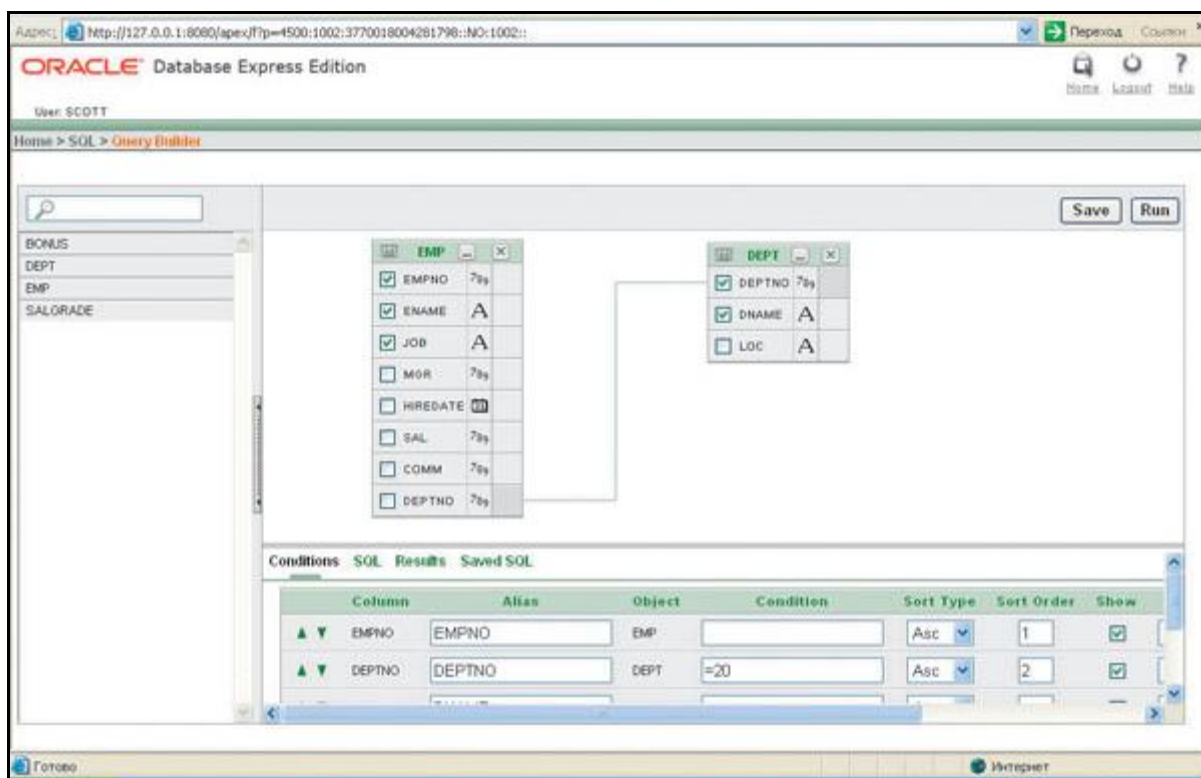


Рис. 9. Визуальное построение запроса к базе данных с помощью Query Builder

В нижней части правой панели на вкладке **Conditions** размещается бланк запроса, оформленный в табличном виде. Строки бланка запроса соответствуют отмеченным ранее полям, а столбцы этой табличной формы дают возможность указать псевдонимы полей (столбец **Alias**), ввести условия отбора записей (столбец **Condition**), определить правила сортировки результатов выборки (столбцы **Sort Type** и **Sort Order**) и др.

Запуск построенного запроса осуществляется с помощью кнопки **Run**. Для просмотра полученных результатов нужно перейти на вкладку **Results**. Вкладка **SQL** позволяет увидеть текст директивы на языке SQL. Эта директива автоматически генерируется посредством Query Builder в соответствии с визуальным

представлением схемы запроса.

4. ИЗУЧЕНИЕ ЭЛЕМЕНТОВ ЯЗЫКА SQL

В современных информационных системах единственным способом взаимодействия между СУБД и прикладными программами, создающими для пользователя удобный визуальный интерфейс при работе с данными, является использование директив (команд, операторов) языка SQL, которые фактически скрываются под оболочкой интерфейса. Помимо этого, языка SQL часто применяется администраторами баз данных и продвинутыми пользователями, которых не устраивают жесткие ограничения регламентированного интерфейса прикладных программ. Таким образом, знание языка SQL сегодня просто необходимо и становится важным квалификационным требованием для многих специалистов, которые по роду своей деятельности тесно связаны с информационными технологиями.

Все команды языка SQL обычно разделяют на четыре категории (подмножества):

- 1) *Язык определения данных* (Data Definition Language – DDL) – включает в себя команды для создания (CREATE), изменения (ALTER) и удаления (DROP) различных объектов базы данных (таблицы, представления, процедуры, триггеры, табличные области, пользователи и др.). Поскольку данные хранятся в виде таблиц, то будет справедливым утверждение, что этот язык позволяет дать полное определение структуры реляционной БД.
- 2) *Язык манипулирования данными* (Data Manipulation Language – DML) – предназначен для работы с существующими таблицами БД, обеспечивая средства вставки (INSERT), изменения (UPDATE), удаления (DELETE) и извлечения (SELECT) данных. Следует отметить, что команда SELECT занимает особое место в этом списке, поскольку ее выполнение не приводит к каким-либо изменениям текущего состояния БД.
- 3) *Язык управления транзакциями* (Transaction Control Language – TCL) – позволяет обеспечить целостное (согласованное, непротиворечивое) состояние БД в случае возникновения затруднений при выполнении группы (последовательности) операторов DML, связанных с некоторой неделимой процедурой изменения данных.
- 4) *Язык управления данными* (Data Control Language – DCL) – используется для административных целей, чтобы регламентировать доступ к данным со стороны пользователей БД и тем самым осуществлять настройку средств информационной безопасности. Команды этого языка позволяют регулировать системные и объектные полномочия для разных категорий пользователей, т.е. выдавать и отменять права на применение определенных операторов DDL и DML к отдельным объектам БД.

На практике активное применение операторов DDL характерно только для начального этапа создания новой БД, а в процессе дальнейшей эксплуатации – при изменении ее структуры. Занесение данных в таблицы БД, а также последующая работа с этими данными осуществляется с помощью многочисленных запросов, относящихся к подмножеству DML. Исходя из более высокой важно-

сти операторов DML с точки зрения рядового пользователя, изучение языка SQL предлагается начать именно с этой категории его команд.

4.1. Средства языка SQL для манипулирования данными

Среди всех команд категории DML наиболее мощной и разнообразной по своим возможностям является директива SELECT. Она также превосходит остальные и по своей популярности. Из этих соображений основная часть материалов, связанных с манипулированием данными, будет посвящена именно директиве SELECT. Однако директивы INSERT, UPDATE и DELETE также не останутся без внимания.

4.1.1. Выборка данных с помощью команды SELECT

Если не учитывать наиболее сложные варианты записи команды SELECT, то ее упрощенную структуру можно представить в виде следующей синтаксической формулы:

```
SELECT < Список_выбора >  
[ INTO < Новая_таблица > ]  
FROM < Набор_источников_данных >  
[ WHERE < Условия_отбора_записей > ]  
[ [ GROUP BY < Ключи_группировки > ] ]  
[ HAVING < Условия_отбора_групп > ] ]  
[ ORDER BY < Ключи_сортировки > ]
```

Здесь четко просматривается разбиение предложения SELECT на последовательные разделы (или фразы), которые нужно записывать **только в указанном порядке**. Признаком очередной фразы является соответствующее служебное слово и, хотя синтаксис языка SQL допускает довольно свободную форму разделения предложений на строки при записи команд, для удобства чтения рекомендуется каждую фразу начинать с новой строки.

В представленной формуле с помощью угловых скобок < ... > указаны некоторые синтаксические элементы. Общий смысл этих элементов можно понять из текста, который содержится в скобках, а более подробная расшифровка будет дана в дальнейших пояснениях.

Квадратными скобками [...] обозначены *необязательные* элементы. Отсюда следует, что обязательными являются только два раздела:

- SELECT, где нужно определить столбцы, которые должны присутствовать в выходной таблице;
- FROM, где задается перечень таблиц и других источников данных, с которыми должен работать запрос.

Важно понимать, что с помощью указанной конструкции пользователь всего лишь описывает желаемый набор данных, которые будут представлены в виде выходной таблицы, но явно не указывает детальный план действий по выполнению запроса. Построение такого плана – это задача оптимизатора запро-

сов, который имеется в составе любой СУБД.

Чтобы на практике освоить возможные варианты использования оператора SELECT, начнем с простейших однотабличных запросов, затем перейдем к более сложным способам выборки данных с применением сортировки и группировки, а в конце будут рассмотрены вложенные и многотабличные запросы.

Отображение заданных столбцов из одиночной таблицы

- полное отображение таблицы

SELECT * FROM < имя_исх._таб. >

Здесь символ * означает «все столбцы исходной таблицы».

- отображение конкретных столбцов таблицы

SELECT < список_столбцов > FROM < имя_исх._таб. >

Задание 1. Выбрать из таблицы EMPLOYEES столбцы FIRST_NAME, LAST_NAME, HIRE_DATE.

Выводимое имя столбца можно поменять при помощи *псевдонима* (alias). В этом случае для элемента списка выбора применяется следующая конструкция:

< элемент_списка > [AS] < псевдоним >

Если псевдоним содержит пробелы или специальные символы, то такой псевдоним нужно поместить в двойные кавычки.

Задание 2. Сделать выборку столбцов из таблицы EMPLOYEES с применением псевдонимов.

В списке выбора могут присутствовать *выражения* языка SQL, что означает включение *вычисляемых полей* в выходную таблицу. Обычно выражение – это некоторая комбинация полей в таблицах, констант и операторов.

Для **числовых** типов используются **арифметические** операции: + , - , * , / . При построении сложных выражений можно применять круглые скобки.

В качестве имени вычисляемого столбца по умолчанию выводится соответствующая формула. Вместо этого гораздо удобнее использовать *псевдоним*.

Задание 3. Используя месячную зарплату сотрудника (столбец SALARY в таблице EMPLOYEES), вычислить годовой доход для каждого из них.

При построении вычисляемых полей можно **сцеплять** элементы типа «строка символов», т.е. выполнять **операцию конкатенации** (||).

Задание 4. При выборке данных из таблицы EMPLOYEES значения полей FIRST_NAME и LAST_NAME для каждого сотрудника поместить в один столбец с названием FULL_NAME.

Если в начале списка выбора указать служебное слово **DISTINCT**, то из результата выборки будут **исключены повторяющиеся строки**.

Задание 5. По данным из таблицы EMPLOYEES выдать неповторяющиеся

значения JOB_ID.

Важно: слово **DISTINCT** относится ко **всему списку выбора**, т.е. его указывают **только один раз в самом начале этого списка**.

Задание 6. По данным из таблицы EMPLOYEES неповторяющиеся значения JOB_ID выдать только в пределах отдельного департамента.

Выборка записей по заданному условию

```
SELECT < список_столбцов >  
FROM < имя_исх._таб.>  
WHERE < условие_отбора >
```

В условие отбора могут входить простые операции сравнения: = , < , > , <= , >=, < > .

Задание 7. По таблице EMPLOYEES получить список сотрудников с низкой зарплатой (не более 2500\$).

В условии отбора может присутствовать оператор **LIKE** ‘< шаблон>’, который выявляет в текстовом поле **наличие подстроки**, задаваемой в виде шаблона.

Шаблон (маска) поиска может включать в себя следующие **специальные символы**:

_ – один произвольный символ;

% – последовательность любых символов (в том числе ни одного).

Задание 8. По таблице EMPLOYEES получить список сотрудников, у которых в фамилии (LAST_NAME) два предпоследних символа – ‘en’.

Попадание в **замкнутый** числовой интервал [a, b] проверяется с помощью условного оператора **BETWEEN** a **AND** b

Задание 9. По таблице EMPLOYEES получить список сотрудников, у которых зарплата находится в замкнутом диапазоне [3000\$, 7000\$].

Принадлежность к множеству (списку) значений, которые заданы в явном виде, можно проверить с помощью условного оператора **IN** (< список >)

Задание 10. По таблице EMPLOYEES получить список сотрудников, которые работают в департаментах 20, 60 и 90.

Важно, что элементы списка сами могут быть списками (множествами) значений.

Пример 1.

```
SELECT department_id, last_name, salary  
FROM employees  
WHERE (department_id, salary) IN ( (20, salary), (60, 4800), (90, 17000) );
```

Отсутствующие (пустые) значения (**NULL**) имеют ряд *особенностей* при

выполнении обычных операций с такими значениями. В упрощенном виде можно полагать, что любая операция с **NULL** всегда дает **NULL**.

В частности, если значение X неизвестно (**NULL**), то сравнение X=X **не дает значения «истина»**. Логика здесь очень простая: если сравнивать «неизвестно что», то и результат будет «неизвестно какой».

Пример 2.

```
SELECT last_name FROM employees
WHERE commission_pct=NULL
```

С помощью этой директивы была предпринята попытка найти сотрудников, у которых нет комиссионной добавки. По указанной выше причине желаемый результат **не получится**.

Проверку на отсутствие значения (т.е. наличие **NULL**) выполняет специальный оператор **IS**.

Пример 3.

```
SELECT last_name FROM employees
WHERE commission_pct IS NULL
```

При построении сложных (составных) условий применяют логические операции **AND**, **OR** и **NOT**.

Задание 11. По таблице EMPLOYEES получить списки сотрудников:

- а) для категории клерков (поле job_id содержит подстроку CLERK), у которых зарплата находится в диапазоне [2500\$, 3000\$];
- б) которые приняты на работу с начала 1999 г. и **не относятся** к департаментам 50, 80, 100.

Сортировка результатов запроса

```
SELECT < список_выбора >
FROM < имя_исх._таб. >
ORDER BY < Ключи_сортировки >
```

В ключе сортировки обычно присутствует имя столбца (или псевдоним) из списка выбора. Вместо этого можно указать порядковый номер элемента из списка SELECT.

Порядок сортировки (для каждого ключа) определяют указатели:

- ❖ ASC – возрастающий порядок (по умолчанию);
- ❖ DESC – убывающий порядок.

Если имеется **несколько** ключей сортировки, то каждый следующий ключ вступает в действие при одинаковых значениях всех предыдущих ключей.

Задание 12. По таблице EMPLOYEES получить список сотрудников, у которых зарплата не ниже 10000\$, с указанием кода подразделения

(department_id) и фамилии (last_name). Список отсортировать сначала по коду подразделения, а в пределах одного подразделения – по фамилии.

4.1.1.1. Встроенные функции языка SQL

Язык SQL в СУБД Oracle располагает большим количеством встроенных (стандартных) функций. Встроенная функция обычно имеет некоторое количество входных аргументов, с учетом которых выдается результирующее значение.

Встроенные функции в СУБД Oracle разделены на **несколько категорий**:

- ❖ символьные функции – манипулируют со строками символов;
- ❖ числовые функции – выполняют расчеты с числовыми данными;
- ❖ функции для работы с датой и временем;
- ❖ функции преобразования типов данных и др.

Некоторые из перечисленных функций рассмотрим с помощью упражнений.

Функция **SUBSTR** (< str>, m [, n]) – из исходной строки < str> выделяет подстроку длиной **n** символов, которая начинается с позиции **m**.

Функция **UPPER** (< str>) – возвращает представление исходной строки < str> с использованием только заглавных букв.

Задание 13. По таблице EMPLOYEES сформировать список сотрудников, у которых первая буква фамилии (last_name) находится в интервале от 'F' до 'K'. Одновременно для каждого сотрудника получить **идентификатор**, который объединяет 3 первых символа имени (first_name) и 2 первых символа фамилии (все это в виде **заглавных букв**).

Функция **LENGTH**(< str>) – возвращает длину (число знаков) исходной строки < str>.

Задание 14. По таблице EMPLOYEES получить список сотрудников с указанием фамилии (last_name) и инициала – первая буквы имени (first_name) с точкой. При выводе сортировать список по убыванию длины фамилии, а фамилии одинаковой длины – по алфавиту.

Функция **MONTHS_BETWEEN** (< d2>, < d1>) – получает число месяцев между датами < d1> (начало интервала) и < d2> (конец интервала).

Функция **SYSDATE** – возвращает текущую дату (аргументы отсутствуют).

Задание 15. По таблице EMPLOYEES получить список сотрудников, где указать фамилию и имя (last_name, first_name), текущий оклад (salary) и дату приема на работу (hire_date). Одновременно для каждого сотрудника определить стаж работы (в месяцах) и начислить бонус в размере 1% от оклада за каждый месяц работы. При выводе сортировать список по убыванию стажа работы.

Функция **TO_CHAR** (< expr> [, < fmt>]) – результат, который получен с

помощью выражения **< expr >** и имеет значение в виде числа (или дата/время), преобразует в символьную строку с учетом формата **< fmt >**.

Задание 16. С помощью числового формата (например, '999.9') и аналогичного денежного формата ('\$999.99') убрать лишние знаки при выводе стажа работы и бонуса в запросе из предыдущего задания.

Функция **NVL (< expr1 > , < expr2 >)** – если выражение **< expr1 >** имеет неопределенное значение (**NULL**), то вместо него использует выражение **< expr2 >**.

Задание 17. По данным из таблицы EMPLOYEES посчитать месячную зарплату сотрудников с учетом комиссионной надбавки (поле commission_pct). Результат упорядочить по убыванию зарплаты.

Функция **DECODE** – производит условную замену конкретных величин. При этом рамках директивы языка SQL реализуется логика IF-THEN-ELSE.

Синтаксис обращения к функции:

```
DECODE ( < expr > , < s_val1 > , < res1 >  
        [ , < s_val2 > , < res2 >  
        [ , ... ] ]  
        [ , < res_def > ] )
```

Функция последовательно сравнивает значение **< expr >** со значениями **< s_val1 >**, **< s_val2 >** и т.д. При первом же совпадении дальнейшие сравнения прекращаются, и функция возвратит значение **< res >** с соответствующим номером. Если ни одного совпадения не произошло, то результатом будет **< res_def >**, а если этот элемент не указан, то функция возвратит **NULL**.

Задание 18. По таблице EMPLOYEES получить список сотрудников, где указать фамилию и имя, а также код должности (job_id). При этом в столбце job_id значения SA_REP и SA_MAN нужно заменить строками 'Торговый представитель' или 'Менеджер по продажам', соответственно. Для остальных значений job_id нужно выводить строку 'Другое'.

4.1.1.2. Групповые (агрегатные) функции

Функции этого типа в процессе получения своего результата работают с *группой записей*. Очень часто это требуется при *статистической обработке*, а также для подсчета *итогов*.

Функция	Результат
COUNT()	количество значений
SUM()	сумма значений
AVG()	среднее значение
MIN()	минимальное значение
MAX()	максимальное значение

В общем случае аргументом групповой функции является обычное выражение, которое будет вычисляться для каждой записи в группе. При записи аргумента можно указать **DISTINCT**, чтобы учесть только различающиеся значения.

Особый случай: COUNT(*) возвратит общее число записей в группе.

Задание 19. Для сотрудников, у которых стаж работы в компании не превышает 15 лет, с помощью запроса по таблице EMPLOYEES найти:

- их число;
- минимальное, максимальное и среднее значение по окладу (SALARY);
- суммарную месячную зарплату с учетом комиссионной надбавки.

4.1.1.3. Запросы с группировкой

Часто при работе с табличными данными нужно выполнить их *группировку*, т.е. сделать так, чтобы в одну группу попадали записи с одинаковыми значениями для заданных атрибутов (*ключи группировки*). В этом случае применяется следующая команда:

```
SELECT < список_столбцов >  
FROM < имя_исх._таб. >  
GROUP BY < Ключи_группировки >
```

Важно: Логика работы запросов с группировкой требует наличия *жесткой связи* между разделами **SELECT** и **GROUP BY**. В частности, любой элемент списка выбора в разделе **SELECT** должен иметь *единственное значение* для каждой группы.

Следовательно, в этом списке могут быть только:

- имена столбцов, которые являются ключами группировки;
- агрегатные функции;
- выражения, состоящие из перечисленных выше элементов.

Задание 20. По таблице EMPLOYEES найти минимальный, максимальный и средний оклад (SALARY) для каждого департамента.

Группировка по **нескольким** ключам позволяет **детализировать** итоговые результаты.

Задание 21. Для каждого департамента сведения о минимальном, максимальном и среднем окладе требуется получить *по отдельным должностям*.

Для отбора *определенных групп* по некоторому условию служит раздел **HAVING**. Это происходит по аналогии с разделом **WHERE**, когда идет отбор определенных записей. Раздел **HAVING** может применяться **только вместе** с разделом **GROUP BY**.

Пример 4. По таблице EMPLOYEES получим список департаментов, в которых число сотрудников больше 5:

```
SELECT DEPARTMENT_ID, COUNT(*)
```

```
FROM EMPLOYEES
GROUP BY DEPARTMENT_ID
HAVING COUNT(*) > 5
```

Задание 22. Получить список департаментов, в которых число низкооплачиваемых сотрудников (с окладом меньше 3000\$) превышает 3.

4.1.1.4. Вложенные запросы (подзапросы)

В разделе **WHERE** оператора **SELECT** может присутствовать *вложенный запрос*. Результат выполнения этого внутреннего запроса (подзапроса) передается внешнему запросу.

Рассмотрим сначала *скалярный* подзапрос, который возвращает *единственное* значение.

Пример 5. Получим список сотрудников, у которых оклад выше среднего:

```
SELECT FIRST_NAME, LAST_NAME
FROM EMPLOYEES
WHERE SALARY >
      ( SELECT AVG (SALARY) FROM EMPLOYEES )
```

Скалярный подзапрос может быть не только частью логического условия в разделе **WHERE**. Следующий пример показывает, как скалярный подзапрос используется в **арифметическом выражении** из списка вывода (раздел **SELECT**).

Пример 6. В списке сотрудников с окладами выше среднего укажем также коэффициент превышения:

```
SELECT FIRST_NAME, LAST_NAME ,
      SALARY / ( SELECT AVG(SALARY)
                  FROM EMPLOYEES ) AS Koeff
FROM EMPLOYEES
WHERE SALARY >
      ( SELECT AVG (SALARY) FROM EMPLOYEES )
```

Задание 23. Получить список сотрудников, у которых период работы в компании ниже среднего стажа.

Задание 24. В списке сотрудников со стажем выше среднего указать также, сколько не хватает до максимального стажа.

С помощью подзапроса можно получить *список*, который затем передается внешнему запросу.

Пример 7. Получим список сотрудников, которые относятся к службам Marketing, Sales и IT:

```
SELECT FIRST_NAME, LAST_NAME
FROM EMPLOYEES
WHERE department_id IN
      ( SELECT department_id
```


FROM DEPARTMENTS
WHERE department_name IN('Marketing', 'Sales', 'IT')

Задание 25. Получить список сотрудников для департаментов, у которых код местоположения (LOCATION_ID) отличается от 1500, 1700 и 2500:

В списке, который формирует вложенный подзапрос, могут быть *составные элементы* (например, пары значений).

Пример 8. Получим список, в котором из каждого департамента должны быть только сотрудники с самым высоким окладом (*элита*):

```
SELECT FIRST_NAME, LAST_NAME
FROM EMPLOYEES
WHERE ( department_id, salary ) IN
        ( SELECT department_id, MAX(salary)
          FROM EMPLOYEES
          GROUP BY department_id )
```

Задание 26. Получить список, в котором из каждого департамента должны быть только сотрудники с максимальным стажем работы (*старожила-могикане*).

Следующий пример демонстрирует еще один возможный вариант использования вложенного подзапроса.

Пример 9. Получим список департаментов, в которых средний оклад ниже среднего по всей компании:

```
SELECT department_id, AVG(salary) avg_sal
FROM EMPLOYEES
GROUP BY department_id
HAVING AVG(salary) <
        ( SELECT AVG(salary) FROM EMPLOYEES )
```

Задание 27. Получить список департаментов, в которых средний стаж работы выше среднего по всей компании.

Любой подзапрос, в свою очередь, также может обращаться к вложенному подзапросу.

Пример 10. Получим список сотрудников, которых приняли на работу после того, как в штате компании появился менеджер отдела продаж (department_name='Sales'):

```
SELECT FIRST_NAME, LAST_NAME
FROM EMPLOYEES
WHERE hire_date >
        ( SELECT hire_date FROM EMPLOYEES
          WHERE employee_id =
            ( SELECT manager_id FROM DEPARTMENTS
              WHERE department_name='Sales' ) );
```

Задание 28. Получить список сотрудников департамента, для которого `POSTAL_CODE='26192'`.

Следует отметить, что в рассмотренных примерах взаимодействие между вложенным подзапросом и запросом, который его охватывает, проходило по достаточно простой схеме – «снизу вверх». Подзапрос такого типа называют *простым*, т.к. он самостоятельно отрабатывает всего *один раз*. Наряду с этим, возможны и другие варианты.

Пример 11. Получим список департаментов (с указанием `department_id` и `department_name`), которые размещаются на территории США (`country_id='US'`):

```
SELECT department_id, department_name
FROM DEPARTMENTS dps
WHERE 'US' = ( SELECT country_id FROM LOCATIONS
              WHERE location_id = dps.location_id )
```

Для последнего примера важно отметить одну *принципиальную особенность*: внешний (основной) запрос управляет работой вложенного подзапроса, т.е. здесь происходит взаимодействие по схеме «сверху вниз». В частности, при работе подзапроса нужно иметь конкретное значение `location_id`, которое передается из основного запроса. Следовательно, подзапрос выполняется *несколько раз* при разных значениях этого параметра. Подзапрос, который обладает такой особенностью, называют *связанным* (или *коррелированным*).

Задание 29. Получить список департаментов (с указанием `department_id` и `department_name`), которые размещаются на территории Европы (`region_name='Europe'`).

Логический оператор **EXISTS** позволяет проверить, дает ли подзапрос требуемый результат.

Пример 12. Сформируем перечень департаментов, для которых отсутствуют данные о сотрудниках:

```
SELECT department_id, department_name
FROM DEPARTMENTS dps
WHERE not EXISTS
      ( SELECT * FROM EMPLOYEES
        WHERE department_id = dps.department_id )
```

Легко заметить, что здесь использован *коррелированный* подзапрос.

Задание 30. Сформировать перечень департаментов, в которых есть сотрудники по имени John.

4.1.1.5. Многотабличные запросы с операцией JOIN

Запрос **SELECT** будет реализован с помощью операции соединения таблиц (**JOIN**), если в разделе **FROM** через запятую перечислить *несколько* источников данных. Такой синтаксис, относящийся к «старому стилю», определяет каждую операцию соединения двух таблиц в *неявном виде*.

Это связано с тем, что дополнительно требуется указать логическое выражение, которое должно содержаться в разделе **WHERE**. Здесь задаются *условия для соединения отдельных строк* (записей) из исходных таблиц. Тем самым *конкретно* определяется *вид соединения*.

Пример 13. По таблицам EMPLOYEES и JOBS получим общий список сотрудников с указанием полного названия должности для каждого сотрудника.

```
SELECT FIRST_NAME, LAST_NAME, JOB_TITLE
FROM EMPLOYEES E, JOBS J
WHERE E.JOB_ID = J.JOB_ID
```

В этом запросе для сокращенного обозначения таблиц используются их *псевдонимы* – **E** и **J**.

Наряду с условием сцепления строк (записей) из исходных таблиц, раздел **WHERE** может содержать и дополнительные условия, по которым происходит отбор результатов соединения.

Задание 31. По таблицам DEPARTMENTS и LOCATIONS получить список департаментов, которые размещены на территории США (country_id='US').

Соединения, при которых записи из исходных таблиц связываются по условию совпадения значений в заданных полях, относятся к классу *внутренних соединений*. Их называют также *эквисоединениями (equijoin)*, т.е. соединениями по условию равенства.

Чтобы получить *внешнее соединение*, нужно к имени поля, в котором ожидается отсутствие совпадающих значений, добавить (+).

Пример 14. По таблицам DEPARTMENTS и EMPLOYEES получим список сотрудников для каждого департамента. Список будет включать даже те департаменты, для которых отсутствуют данные о сотрудниках.

```
SELECT department_name d_name, last_name, first_name
FROM departments D, employees E
WHERE D.department_id = E.department_id (+)
ORDER BY d_name;
```

Как обычно, с помощью раздела **GROUP BY** можно произвести группировку результатов соединения таблиц.

Задание 32. Выдать список департаментов, указав по каждому департаменту число работающих сотрудников и общий фонд зарплаты. Список должен включать даже те департаменты, для которых данные о сотрудниках отсутствуют.

В случае необходимости для некоторой таблицы можно произвести *самосоединение*, т.е. соединить таблицу саму с собой.

Пример 15. По таблице EMPLOYEES получим список сотрудников с указанием для каждого из них имени и фамилии его непосредственного начальника.

```
SELECT S.last_name || ' ' || S.first_name AS Slave,
```

```

        B.last_name || ' ' || B.first_name AS Boss
FROM employees S, employees B
WHERE S.manager_id = B.employee_id
ORDER BY Boss;

```

Задание 33. По таблице EMPLOYEES получить список сотрудников с указанием для каждого из них фамилии, имени и телефонного номера его непосредственного начальника. Включить в список даже тех сотрудников, у которых нет начальников в штате компании.

Начиная с СУБД Oracle 9i и стандарта SQL:1992, стало возможным **явным образом** в разделе **FROM** оператора **SELECT** указывать операцию соединения в виде следующей конструкции:

```

< left_tab> < join_type> < right_tab> ON < join_cond>

```

Здесь подразумевается соединение таблиц **< left_tab>** и **< right_tab>**, а элемент **< join_cond>** – это условие для соединения строк, которое переносится внутрь раздела **FROM**.

В разделе **WHERE** остаются только дополнительные условия для отбора записей по другим критериям, что делает текст запроса **более понятным**.

Синтаксический элемент **< join_type>** может принимать следующие значения:

- [INNER] JOIN — внутреннее соединение (применяется по умолчанию);
- LEFT [OUTER] JOIN — левое внешнее соединение;
- RIGHT [OUTER] JOIN — правое внешнее соединение;
- FULL [OUTER] JOIN — полное внешнее соединение.

Пример 16. Запрос из задания 31 можно записать в следующем виде:

```

SELECT department_id, department_name
FROM departments D JOIN locations L
        ON (D.location_id = L.location_id)
WHERE country_id='US';

```

В данном случае имеет место **эквисоединение**, причем совпадение значений проверяется в столбцах с **одинаковыми названиями**. По этой причине для раздела **FROM** есть более простая конструкция:

```

FROM departments JOIN locations USING (location_id)

```

Задание 34. По таблицам DEPARTMENTS и EMPLOYEES с помощью внешнего соединения получить список сотрудников для каждого департамента. Выбрать тип соединения, который в данном случае будет наиболее подходящим.

Во многих случаях операция **JOIN** исключает необходимость применения вложенных запросов.

Задание 35. Получить список сотрудников, которые относятся к службам

Marketing, Sales и IT.

Задание 36. Получить список сотрудников департамента, для которого POSTAL_CODE='26192'.

В конструкции **ON** < **join_cond** > можно указать соединение строк по *произвольному* условию с применением любых операций сравнения (<=, <>, LIKE, BETWEEN и др.). Это позволяет реализовать так называемое **Θ-соединение**.

Пример 17. Получим распределение сотрудников по разным категориям в зависимости от уровня оклада (salary). Эти категории заданы в виде специальной таблицы **Sal_Grade**:

CAT_ID	LOW_LIMIT	HIGH_LIMIT
1	2000	5000
2	5001	10000
3	10001	15000
4	15001	20000
5	20001	25000

```
SELECT Low_limit, High_limit, count(*) Freq
FROM employees JOIN sal_grade
      ON (salary BETWEEN Low_limit AND High_limit)
GROUP BY Low_limit, High_limit
ORDER BY Low_limit;
```

Для рассмотренного примера существуют и другие варианты решения поставленной задачи. Например, можно вообще обойтись без операции **JOIN**:

```
SELECT
SUM (CASE WHEN salary BETWEEN 2000 AND 5000 THEN 1 ELSE 0 END)
"<=5000",
SUM (CASE WHEN salary BETWEEN 5001 AND 10000 THEN 1 ELSE 0 END)
"<=10000",
SUM (CASE WHEN salary BETWEEN 10001 AND 15000 THEN 1 ELSE 0 END)
"<=15000",
SUM (CASE WHEN salary BETWEEN 15001 AND 20000 THEN 1 ELSE 0 END)
"<=20000",
SUM (CASE WHEN salary BETWEEN 20001 AND 25000 THEN 1 ELSE 0 END)
"<=25000"
FROM employees;
```

Однако предложенный ранее вариант с использованием специальной таблицы **Sal_Grade** имеет некоторые *важные достоинства*. Самое главное *преимущество* заключается в том, что пользователь может самостоятельно выбирать границы категорий, соответствующим образом изменяя содержимое этой таблицы.

Задание 37. Получить распределение сотрудников по разным категориям в зависимости от стажа работы. Эти категории задать в виде специальной табли-

цы **Stg_Grade**.

Самый простой вариант построения раздела **FROM** относится к *естественному* соединению двух таблиц:

FROM < left_tab> **NATURAL JOIN** < right_tab>

Такая конструкция, которая очень привлекательна в силу своей простоты, подразумевает, что при соединении таблиц равенство значений будет проверяться попарно *во всех столбцах с одинаковыми названиями*.

Задание 38. С помощью естественного соединения таблиц EMPLOYEES и JOBS найти среднюю заработную плату по отдельным должностям.

Однако нужно помнить об *опасностях*, которые возникают из-за *отсутствия явного контроля* за реальным условием соединения строк. Конкретным *примером* такой опасности является следующий запрос для получения списка сотрудников по каждому департаменту:

```
SELECT department_name, last_name, first_name
FROM departments D NATURAL JOIN employees E
ORDER BY 1;
```

Здесь по правилам NATURAL JOIN *формально (по умолчанию)* применяется следующее условие соединения строк из рассматриваемых таблиц:

E.department_id = D.department_id AND E.manager_id = D.manager_id

Но при построении требуемого списка нужна *только левая часть* этого условия, а вторая часть условия вообще *не отражает логику связей* между таблицами departments и employees. В результате запрос дает список только тех сотрудников, у которых непосредственный начальник — менеджер подразделения.

4.1.2. Работа с результатами нескольких запросов

Чтобы комбинировать результаты (наборы данных), полученные от нескольких запросов, в языке SQL имеются реляционные операции, которые очень похожи на операции над множествами: 1) объединять (**union**); 2) пересекать (**intersect**); 3) исключать (**except** или **minus**).

Если имеются результаты выборки от двух запросов (query1 и query2), то в упрощенном виде обращение к этим операциям выглядит так:

```
< query1 >
{ UNION | UNION ALL | INTERSECT | MINUS }
< query2 >
```

Здесь знак | указывает на необходимость выбора одного из вариантов.

Основные правила для рассматриваемых операций:

- все исходные наборы данных должны иметь одинаковую структуру (с точностью до *совместимости* типов столбцов);
- за исключением **UNION ALL**, дубликаты строк убираются автоматически;

– названия столбцов в результирующей таблице определяются первым запросом.

Пример 18. Получим список руководящего персонала компании, включив в него руководителей департаментов, а также сотрудников, у которых есть подчиненные.

```
SELECT first_name, last_name
FROM employees E JOIN departments D
                ON (E.employee_id = D.manager_id)

UNION

SELECT first_name, last_name
FROM employees
WHERE employee_id IN
      ( SELECT DISTINCT manager_id FROM employees );
```

Задание 39. С помощью таблиц EMPLOYEES и JOB_HISTORY построить «карьерную лестницу» для отдельных сотрудников компании.

Подсказка. Первый запрос (query1) должен выбирать из таблицы JOB_HISTORY столбцы employee_id, job_id, start_date и end_date. Второй запрос (query2) выбирает из таблицы EMPLOYEES столбцы employee_id и job_id только тех сотрудников, для которых есть данные в таблице JOB_HISTORY. При этом столбцы start_date и end_date заполняются пустыми значениями (NULL). Результат объединения (**union**) нужно отсортировать по столбцам employee_id и start_date.

Задание 40. С помощью операции **MINUS** получить названия департаментов, в которых отсутствуют сотрудники.

Подсказка. Первый запрос (query1) должен выбирать столбец department_name из таблицы DEPARTMENTS. Второй запрос (query2) в результате соединения таблиц DEPARTMENTS и EMPLOYEES также выдает один столбец department_name.

4.1.3. Добавление (вставка) новых записей в таблицу с помощью команды INSERT

Для директивы **INSERT** существует два варианта.

Вариант 1 дает возможность вставки единственной строки:

```
INSERT INTO < имя_таб > [ (< список_столбцов > ) ]
                        VALUES ( < список_значений > )
```

При построении такого запроса необходимо учитывать следующие вполне понятные **ограничения**:

- между позициями в списке столбцов и списке значений должно существовать **строгое соответствие**;
- в списке столбцов можно не указывать только те столбцы, для которых допускается значение **NULL** или существует значение по умолчанию

(DEFAULT);

- соответствующие элементы в списке столбцов и списке значений должны быть совместимы по типу данных.

Пример 19.

```
INSERT INTO EMPLOYEES ( EMPLOYEE_ID, LAST_NAME,  
                        EMAIL, HIRE_DATE, JOB_ID, SALARY)  
VALUES ( 300, 'Ivanov', 'IVAN', TO_DATE('1.06.2013'),  
        'IT_PROG', 3000)
```

Пример 20. Порядок перечисления атрибутов, которым будут присвоены требуемые значения при вставке записи, может отличаться от того, что было в предыдущем примере:

```
INSERT INTO EMPLOYEES (LAST_NAME, HIRE_DATE, JOB_ID,  
                        MANAGER_ID, EMAIL, EMPLOYEE_ID)  
VALUES ( 'Baranov', TO_DATE('10.06.2013'), 'MK_MAN',  
        107, 'BARAN', 301)
```

Задание 41. Добавить в таблицу JOBS новую запись, указав при этом следующие значения: JOB_ID='HR_MAN', JOB_TITLE='Human Resources Manager' и MIN_SALARY=4500.

Задание 42. Добавить новую запись в таблицу JOB_HISTORY, указав при этом EMPLOYEE_ID=111, START_DATE=TO_DATE ('28.09.97'), END_DATE=TO_DATE('31.12.09'),.

Задание 43. Запрос из примера 19 модифицировать таким образом, чтобы для создаваемой учетной записи сотрудника в столбец SALARY автоматически добавлялось минимальное значение заработной платы для указанной должности.

Если список столбцов отсутствует, то список значений должен обязательно содержать элементы для всех столбцов (в порядке их описания при создании таблицы).

Пример 21.

```
INSERT INTO EMPLOYEES  
VALUES (302, 'John', 'Lemon', 'JLEMON', NULL,  
        TO_DATE('15.06.2013'), 'IT_PROG', 3000, NULL, 107, 60)
```

Задание 44. Запишите директиву языка SQL для добавления новой записи в таблицу COUNTRIES без указания списка столбцов этой таблицы.

Необходимо отметить, что такой вариант директивы **INSERT** считается очень ненадежным по следующим причинам:

- легко допустить ошибку, не имея перед глазами списка названий столбцов таблицы;

- после изменения структуры таблицы операция становится некорректной.

Вариант 2 обеспечивает копирование множества строк из одной таблицы в другую. При этом добавляемые строки являются результатом работы подзапроса SELECT, который входит в состав директивы **INSERT**:

```
INSERT INTO < имя_таб > [ (< список_столбцов > ) ]  
SELECT .....
```

Очевидно, что для этого варианта добавления (вставки) новых записей в некоторую таблицу также действуют все указанные выше ограничения.

Пример 22. Создать временную таблицу EMP_TEMP со столбцами EMPLOYEE_ID, FIRST_NAME, LAST_NAME и SALARY. Скопировать в эту таблицу данные о сотрудниках, у которых значения DEPARTMENT_ID принадлежат списку (20, 50, 70).

```
CREATE TABLE EMP_TEMP  
  ( EMP_ID NUMBER(3), FNAME CHAR(20), LNAME CHAR(20),  
    EMAIL CHAR(10) );  
INSERT INTO EMP_TEMP  
  SELECT EMPLOYEE_ID, FIRST_NAME, LAST_NAME, EMAIL  
  FROM EMPLOYEES WHERE DEPARTMENT_ID IN(20, 50, 70);
```

Задание 45. Добавить в таблицу EMP_TEMP данные о сотрудниках, которые занимаются закупками товаров (Purchasing) и их продажей (Sales).

Единственный оператор **INSERT**, относящийся ко второму типу, позволяет «раскидать» результаты запроса сразу по нескольким таблицам.

Пример 23. На основе таблицы **EMPLOYEES** параллельно заполним данными таблицы EMP_LOW, EMP_MID и EMP_HIGH:

```
INSERT ALL  
  WHEN SALARY > 8000 THEN INTO EMP_HIGH  
  WHEN SALARY BETWEEN 4000 AND 8000  
    THEN INTO EMP_MID  
  WHEN SALARY < 4000 THEN INTO EMP_LOW  
SELECT * FROM EMPLOYEES
```

После выполнения записанного оператора языка SQL в таблицы EMP_LOW и EMP_HIGH попадут, соответственно, данные о низкооплачиваемых ($SALARY < 4000\$$) и высокооплачиваемых ($SALARY > 8000\$$) сотрудниках, а таблица EMP_MID будет содержать данные о сотрудниках со средним уровнем зарплаты ($8000\$ \geq SALARY \geq 4000\$$). Предполагается, что все таблицы, которые участвуют в рассмотренном запросе, одинаковы по своей структуре.

Задание 46. Используя в качестве источника записей соединение таблиц LOCATIONS и COUNTRIES, заполнить с помощью единственного оператора

INSERT таблицы LOCS_1, LOCS_2 и LOCS_3, поместив туда данные, которые относятся к Европе (REGION_ID=1), Америке (REGION_ID=2) и Азии (REGION_ID=3), соответственно. Структуру таблиц LOCS_1, LOCS_2 и LOCS_3 выбрать по своему усмотрению.

Подсказка 1. Первоначально для создания новой таблицы (например, LOCS_1) рекомендуется воспользоваться следующим оператором:

```
CREATE TABLE LOCS_1
AS SELECT * FROM LOCATIONS NATURAL JOIN COUNTRIES
WHERE 1=2
```

После выполнения этого оператора появится пустая таблица LOCS_1, структура которой будет объединять столбцы исходных таблиц LOCATIONS и COUNTRIES. Теперь остается вставить в нее требуемые записи.

Подсказка 2. Если в подзапросе, который применяется для создания новой таблицы, вместо символа * указать только некоторые столбцы исходных таблиц, то новая таблица будет содержать только эти столбцы.

4.1.4. Изменение данных в таблицах с помощью операторов UPDATE и DELETE

Оператор **UPDATE** позволяет изменить содержимое существующих записей в указанной таблице:

```
UPDATE < имя_таб >
SET col_name1 = expr1 [ , col_name2 = expr2 ... ]
[ WHERE < условия_отбора_записей > ]
```

Здесь необязательный раздел **WHERE** предназначен для того, чтобы изменить данные только в тех записях, которые удовлетворяют определенным условиям отбора.

Пример 24. Всему персоналу компании повысить оклад на 5%.

```
UPDATE EMPLOYEES SET SALARY=SALARY*1.05
```

Пример 25. Сотруднику, у которого EMPLOYEE_ID=300, установить новую зарплату в размере 4000\$.

```
UPDATE EMPLOYEES SET SALARY=4000
WHERE EMPLOYEE_ID=300
```

Задание 47. Повысить оклад на 10% всем руководителям департаментов.

Задание 48. Сотрудника, у которого LAST_NAME='Kozlov', перевести в департамент Accounting на должность Accounting Manager и установить ему новую зарплату в размере 8500\$.

В составе оператора UPDATE могут присутствовать подзапросы, с помощью которых определяются подставляемые значения.

Пример 26. Сотруднику, у которого EMPLOYEE_ID=300, установить такие же значения DEPARTMENT_ID и PHONE_NUMBER, как в случае EMPLOYEE_ID=104.

```
UPDATE EMPLOYEES SET (DEPARTMENT_ID, PHONE_NUMBER)=  
    ( SELECT DEPARTMENT_ID, PHONE_NUMBER  
      FROM EMPLOYEES WHERE EMPLOYEE_ID=104 )  
WHERE EMPLOYEE_ID=300
```

Задание 49. Сотрудника, у которого EMPLOYEE_ID=300, перевести в подчинение к тому же менеджеру, как у EMPLOYEE_ID=105, и установить зарплату на 10% выше минимума для занимаемой должности.

С помощью оператора **DELETE** можно удалить записи из указанной таблицы:

```
DELETE FROM < имя_таб >  
[ WHERE < условия_отбора_записей > ]
```

Задание 50. В таблице EMP_TEMP оставить только данные о сотрудниках, которые занимаются продажами (Sales) и маркетингом (Marketing).

В разделе **WHERE**, с помощью которого определяются условия для отбора удаляемых записей, могут присутствовать подзапросы.

Задание 51. В таблице EMP_LOW оставить только данные о сотрудниках, у которых заработная плата ниже половины средней зарплаты по компании.

Для *полного удаления строк* существует более быстрая команда:

```
TRUNCATE TABLE < имя_таб >
```

Высокая скорость выполнения этой команды обусловлена следующими факторами:

- строки удаляются не поштучно, как при DELETE, а путем «усечения» сегмента (зоны) для хранения данных;
- информация, которая обеспечит отмену этой операции, *не сохраняется*.

4.1.5. Использование оператора MERGE

Этот оператор считается очень эффективным средством вливания порции новых данных из источника (SRC) в таблицу-приемник (DEST). Схематично оператор выполняется следующим образом:

- если запись из SRC уже существует в DEST, то выполняется операция

UPDATE или DELETE;

- иначе выполняется вставка новой записи в DEST, т.е. операция **INSERT**.

Отсюда понятно, почему эту команду иногда называют **UPsert (UPdate + inSERT)**.

Общий синтаксис оператора MERGE можно представить следующей формулой:

```
MERGE INTO < целевая_таб >  
      USING < исходная_таб >  
      ON < условие_слияния >  
      < конструкция_обновления >  
      < конструкция_вставки >
```

В этой синтаксической конструкции оператора MERGE элемент < целевая_таб > задает таблицу назначения для результирующего набора данных. Выражением < исходная_таб > может быть таблица, представление или подзапрос, который будет служить источником строк, объединяемых в целевой таблице. Это выражение определяет данные, которые будут использованы при выполнении действий INSERT и UPDATE. Как правило, этому источнику данных, который играет роль исходной таблицы, приписывают псевдоним, чтобы в других разделах оператора можно было легко ссылаться на отдельные столбцы.

Выражение < условие_слияния > выполняет функцию переключателя между операциями UPDATE и INSERT: если результат вычисления этого условия равен TRUE, то вступает в действие < конструкция_обновления >. Если же это значение равно FALSE, то должна выполняться < конструкция_вставки >. Как правило, в условии слияния участвует первичный ключ целевой таблицы.

Раздел, в котором содержится < конструкция_обновления >, должен обязательно начинаться с ключевой последовательности слов **WHEN MATCHED THEN UPDATE**, а в самом начале следующего раздела, где содержится < конструкция_вставки >, необходимо указать **WHEN NOT MATCHED THEN INSERT**. После такого стандартного начала этих разделов необходимо задать конкретные действия по выполнению операций UPDATE и INSERT.

В сочетании с выражением < конструкция_обновления > можно использовать выражение < конструкция_удаления >. Эта конструкция, которая должна начинаться с ключевого слова **DELETE**, обеспечивает удаление строки, если она соответствует условию, указанному в выражении. Условие удаления вычисляется после обновления строки и применяется только к тем строкам, которые были обновлены.

Пример 27. Пусть в БД имеются две однотипные таблицы TOP_LIST и NEWS с полями GAMER и RATING. Таблицу TOP_LIST нужно регулярно обновлять с учетом данных из таблицы NEWS, т.е.

- вставлять новые записи, которых до этого не было;
- изменять поле RATING для имеющихся записей;
- удалять из таблицы TOP_LIST существующие записи, для которых новые

значения RATING стали ниже заданного порога (например, 50).

Поставленную задачу легко решить с помощью следующего оператора:

```
MERGE INTO TOP_LIST d
USING NEWS s
ON (d.GAMER = s.GAMER)
WHEN MATCHED THEN
    UPDATE SET d.RATING = s.RATING
    DELETE WHERE s.RATING < 50
WHEN NOT MATCHED THEN
    INSERT (d.GAMER, d.RATING) VALUES (s.GAMER, s.RATING)
```

Задание 52. Создать следующие таблицы:

Таблица TOP_LIST	
GAMER	RATING
2	100
3	150
4	120
5	110
6	160

Таблица NEWS	
GAMER	RATING
6	40
4	130
5	120
7	140

Запустить SQL-оператор, рассмотренный в примере 27, и проанализировать полученный результат.

Задание 53. На основе таблицы JOBS создать таблицу JOBS_MAN и скопировать туда данные о должностях, у которых в названии (JOB_TITLE) присутствует слово Manager. Затем создать таблицу JOB_NEWS, в которую ввести следующие данные:

JOB_ID	JOB_TITLE	MIN_SALARY	MAX_SALARY
FI_MGR	Finance Manager	10000	16000
ST_MAN	Stock Manager	5500	1100
MK_MAN	Marketing Manager	8000	13500
PR_MGR	Public Relations Manager	7000	9000

Произвести слияние таблиц JOBS_MAN и JOB_NEWS с помощью оператора MERGE, обеспечивая при этом только обновление данных в таблице JOBS_MAN и добавление в нее новых записей с учетом данных, содержащихся в таблице JOB_NEWS. Проанализировать полученный результат.

Подсказка. Создание таблицы JOBS_MAN и копирование в нее данных из таблицы JOBS легко осуществить с помощью оператора CREATE TABLE с встроенным подзапросом SELECT.

4.2. Средства языка SQL для определения данных

4.2.1. Работа с представлениями

Представление (view) часто называют **виртуальной таблицей**. В отличие от реальной (физической) таблицы, соответствующий набор данных не хранится постоянно, а в случае необходимости создается **«на лету»** во временной памяти. Это делается с помощью **запроса на выборку**, который выполняется каждый раз при активизации представления. В общем случае представления и таблицы неразличимы с точки зрения пользователя.

Необходимость использования представлений обусловлена несколькими причинами:

- 1) обеспечивается независимость приложений (пользовательских программ) от изменения логической структуры БД, т.к. все эти изменения достаточно учесть только на уровне представления;
- 2) пользователи разных категорий получают только те данные, которые реально нужны им для выполнения своих функций (обязанностей). В результате интерфейс пользователя не будет перегружен лишними данными;
- 3) возникает возможность настройки формата данных для каждой категории пользователей, поэтому все пользователи будут получать данные в наиболее удобной форме;
- 4) появляются средства управления санкционированным доступом к данным;
- 5) достигается экономия места на дисковом пространстве для постоянного хранения данных, т.к. СУБД сохраняет только определяющий запрос, который формирует представление;
- 6) запросы к представлению как к единственной виртуальной таблице имеют более низкую сложность (особенно при работе с несколькими реальными таблицами).

Пример 28. Создать представление, которое менеджерам по продажам (Sales) предоставит доступ к данным только по своим сотрудникам.

```
CREATE OR REPLACE VIEW sales_emp  
AS SELECT * FROM employees  
WHERE department_id=80  
WITH READ ONLY;
```

Теперь этим менеджерам нужно запретить прямой доступ к таблице employees, но они получают право пользоваться представлением sales_emp.

Задание 54. С помощью представления sales_emp выдать список сотрудников отдела продаж, у которых оклад (SALARY) не ниже 10000\$.

Представления могут ограничивать доступ к данным не только по записям, но и по столбцам (иногда говорят «по горизонтали» и «по вертикали»).

Пример 29. Доступ к данным о сотрудниках ограничить только сведениями для контактов.

```
CREATE OR REPLACE VIEW emp_cont
AS SELECT FIRST_NAME, LAST_NAME,
        EMAIL, PHONE_NUMBER
FROM employees
```

При обращении к нескольким таблицам представление скрывает от пользователя сложную логическую структуру БД:

```
CREATE OR REPLACE VIEW emp_dep_job
AS SELECT first_name, last_name, department_name, job_title
FROM employees JOIN jobs USING(job_id)
        JOIN departments USING(department_id)
```

Задание 55. С помощью представления emp_dep_job вывести список менеджеров среднего звена.

Подсказка. Менеджерами являются сотрудники, у которых поле job_title включает в себя подстроку 'Manager'.

Задание 56. С помощью представления emp_dep_job подсчитать количество менеджеров среднего звена в составе разных департаментов.

ВОПРОСЫ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

- 1) Что означает аббревиатура SQL?
- 2) Будет ли правильным утверждать, что SQL является непроцедурным языком?
- 3) На какие категории разделяются команды языка SQL?
- 4) Какие функции выполняют команды управления транзакциями?
- 5) Перечислите основные предложения оператора SELECT. Какие из них являются обязательными?
- 6) Проанализируйте корректность следующих запросов:
 - a) Select *
 - b) Select * from checks
 - c) Select amount name payee FROM checks
- 7) Какая из агрегатных функций SUM, COUNT, MIN, MAX, AVG возвращает множество значений?
- 8) Будут ли корректными следующие утверждения:
 - a) степень вложенности подзапросов не может превышать 2;
 - b) коррелированные подзапросы являются полностью самостоятельными (независимыми).
- 9) Можно ли в выражении для ключевого слова WHERE задать несколько условий?
- 10) Будет ли правильным утверждение, что при использовании ключевого слова IN проверяемое значение должно совпадать с каждым элементом списка?
- 11) Будет ли правильным утверждение, что при наличии ключевого слова

HAVING необходимо также использовать ключевые слова GROUP BY?

- 12) Какой тип соединения таблиц необходимо использовать в запросе, чтобы извлечь записи из некоторой таблицы независимо от наличия связанных записей в другой таблице?
- 13) В чем состоит особенность натурального (естественного) соединения двух таблиц?
- 14) Каким будет результат, если при выборке из двух таблиц, не указать условие для связывания их записей?
- 15) Какие разделы могут использоваться при составлении команды SELECT для выбора данных из таблицы?
- 16) Какая директива языка SQL используется для создания таблиц?
- 17) Какие основные типы данных могут использоваться при создании таблиц?
- 18) Что такое псевдоним (алиас) и каким образом он определяется?
- 19) Какой формат имеет SQL-команда для уничтожения таблицы?
- 20) Поясните форматы SQL-команды для вставки новых записей в таблицу?
- 21) Каким образом можно указать конкретные колонки таблицы в запросе на выборку данных?
- 22) Каким образом оператор SELECT позволяет осуществить выбор строк таблицы, удовлетворяющим заданным условиям?
- 23) Какие специальные символы могут использоваться в функции LIKE для сравнения по образцу?
- 24) Какое ключевое слово задает режим запрета вывода строк-дубликатов?
- 25) Какими способами в операторе SELECT можно задать условия для соединения таблиц?
- 26) Какие объекты базы данных обеспечивают функционирование старых приложений без их модификации в случае изменения структуры БД?
- 27) Можно ли создать представление (view), которое будет работать одновременно с несколькими таблицами БД?
- 28) Каким образом можно перенести (по заданному условию) данные из одной таблицы в другую?
- 29) Каким целям служат представления, чем они функционально отличаются от таблиц?
- 30) Как NULL-значения влияют на результаты арифметических операций?
- 31) Как правильно выполнить сравнение с NULL-значением?
- 32) Перечислите основные агрегатные функции и правила их использования.
- 33) Можно ли выполнить запрос на выборку, если данные расположены в нескольких таблицах?
- 34) В чем разница между INNER JOIN и OUTER JOIN?
- 35) Объясните отличие в использовании предложений WHERE и HAVING.
- 36) Укажите причины, по которым операция добавления в таблицу новой записи может завершиться с ошибкой.
- 37) Можно ли одной командой добавить в таблицу несколько новых строк?
- 38) Как быстро очистить всю таблицу? В чем недостаток этой операции?
- 39) Укажите реляционную операцию (UNION, UNION ALL, INTERSECT, MINUS) для работы с результатами двух запросов на выборку, если требу-

ется:

- а) показать совпадающие записи;
- б) показать все данные;
- с) показать все данные, но без повторений;
- д) показать только те строки первого запроса, которые не выдаются вторым.

40) Поясните, в чем разница между командами DROP TABLE TAB_1 и DELETE FROM TAB_1.

Список рекомендуемой литературы

1. Пржиялковский В.В. Введение в Oracle SQL: Учеб. пособие. – М.: Нац. откр. ун-т "ИНТУИТ": Бином. Лаборатория знаний, 2011. – 319 с.
2. Бобровский С. Oracle Database XE для Windows. Эффективное использование: Пер. с англ. – М.: Изд-во «Лори», 2009. – 486 с.
3. Кириллов В.В., Громов Г.Ю. Введение в реляционные базы данных: Учеб. пособие. – СПб.: БХВ-Петербург, 2009. – 464с.
4. Наместников А.М. Построение баз данных в среде Oracle. Практический курс: Учеб. пособие. – Ульяновск: УлГТУ, 2008. – 118с.
5. Пери Дж., Пост Дж. Введение в Oracle 10g: Пер. с англ. – М.: ООО «Изд. дом Вильямс», 2006. – 704 с.