

Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

Е. В. Калабухов

РАБОТА С РЕЛЯЦИОННЫМИ БАЗАМИ ДАННЫХ В СУБД ORACLE

*Рекомендовано УМО по образованию в области информатики
и радиоэлектроники в качестве пособия для специальности
1-40 02 01 «Вычислительные машины, системы и сети»*

Минск БГУИР 2021

УДК 004.655(076)
ББК 32.97.134я73
К17

Р е ц е н з е н т ы:

кафедра программного обеспечения информационных систем и технологий
Белорусского национального технического университета
(протокол №11 от 20.05.2020);

заведующий лабораторией идентификации систем
государственного научного учреждения «Объединенный институт
проблем информатики Национальной академии наук Беларуси»
доктор технических наук, профессор А. А. Дудкин

Калабухов, Е. В.

К17 Работа с реляционными базами данных в СУБД Oracle : пособие /
Е. В. Калабухов. – Минск : БГУИР, 2021. – 70 с. : ил.
ISBN 978-985-543-595-3.

Рассмотрены вопросы работы с СУБД Oracle на языке SQL по формированию схем, объектов схем, выборки и модификации данных, транзакций. Приведены сведения о синтаксисе операторов и примеры для изучения особенностей диалекта SQL СУБД Oracle.

Пособие адресовано студентам, изучающим дисциплину «Базы данных», а также может использоваться для самостоятельной подготовки при изучении языка SQL.

**УДК 004.655(076)
ББК 32.97.134я73**

ISBN 978-985-543-595-3

© Калабухов Е. В., 2021
© УО «Белорусский государственный
университет информатики
и радиоэлектроники», 2021

СОДЕРЖАНИЕ

ВВЕДЕНИЕ.....	4
1 ИСПОЛЬЗУЕМОЕ ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ORACLE	5
1.1 Oracle Database Express Edition и средства разработки.....	5
1.2 Работа с СУБД Oracle в виртуальной машине.....	6
1.3 Онлайн-среда разработки Live SQL.....	6
2 ФОРМИРОВАНИЕ РЕЛЯЦИОННОЙ СХЕМЫ ДАННЫХ.....	7
2.1 Введение в Oracle SQL	7
2.2 Формирование реляционной схемы данных.....	9
2.2.1 Понятие схемы данных Oracle.....	9
2.2.2 Управление объектами схемы	13
2.2.3 Заполнение таблиц схемы данными.....	20
2.3 Создание объектов схемы по логической модели	24
2.4 Схема HR.....	26
3 ВЫБОРКА ДАННЫХ.....	31
3.1 Оператор выборки SELECT	31
3.1.1 Представление результата (предложение SELECT).....	32
3.1.2 Сортировка результатов (предложение ORDER BY)	37
3.1.3 Фильтр строк данных (предложение WHERE).....	38
3.1.4 Агрегатные функции, группировка данных и фильтр групп (предложения GROUP BY и HAVING)	42
3.1.5 Сложные источники данных (предложение FROM).....	44
3.2 Подзапросы.....	49
3.3 Операторы работы с множествами (UNION, INTERSECT, MINUS).....	52
3.4 Иерархические запросы.....	54
3.5 Рекурсивные запросы	56
3.6 Перекрестные запросы	58
4 МОДИФИКАЦИЯ ДАННЫХ.....	61
4.1 Операторы модификации данных	61
4.2 Импорт данных в схему пользователя.....	65
Список использованных источников	68

ВВЕДЕНИЕ

Настоящее пособие описывает основы языка SQL для работы с реляционными таблицами и в основном предназначено для помощи студентам в подготовке к лабораторным и практическим занятиям по дисциплине «Базы данных», но также может использоваться и для самостоятельной подготовки в этой области.

В качестве рабочей платформы для выполнения обработки данных использована широко распространенная СУБД Oracle, поэтому в пособии описывается не только теория языка, но и работа с конкретными приложениями, хотя и без углубления в вопросы администрирования СУБД. В описании конструкций синтаксиса языка SQL также была рассмотрена специфика диалекта языка для СУБД Oracle, но для простоты изучения конструкции языка приводятся в урезанном виде, актуальном для работы с обычными базовыми таблицами.

В пособии освещены вопросы, возникающие при работе с учетными записями пользователей и объектами схемы данных, при выполнении различных видов запросов на выборку данных, рассмотрены операторы модификации данных и простейшее управление транзакциями, а также представлены процедуры работы с данными, такие, например, как экспорт-импорт данных. Для простоты понимания применения операторов SQL приведены примеры решения задач, которые базируются на стандартной учебной схеме данных Oracle, поэтому для большинства примеров не даются результаты работы запросов, так как рекомендуется их получить самостоятельно. Некоторые моменты работы с программным обеспечением для лучшего понимания пояснены с помощью скриншотов.

В связи с необходимостью работы с конкретной версией СУБД в пособии приводятся ссылки на документацию Oracle, которая находится в открытом доступе в виде электронных ресурсов. Также в списке литературы представлены классические печатные работы для рассмотрения связи языка SQL с основами реляционных баз данных и для изучения оригинальных примеров его использования.

1 ИСПОЛЬЗУЕМОЕ ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ORACLE

Oracle (www.oracle.com) – один из крупных в мире производителей программного обеспечения, владеющий около 30 % глобального рынка программного обеспечения. Ключевой продукт компании – Oracle Database – объектно-реляционная СУБД. В этом разделе приведен краткий обзор версий СУБД и других средств разработки, рекомендуемых для работы с пособием.

1.1 Oracle Database Express Edition и средства разработки

Oracle Database Express Edition (XE) [1] – бесплатная версия СУБД, которая, несмотря на значительные ограничения относительно корпоративной версии Enterprise Edition по максимальному размеру базы данных, объему используемой памяти и потокам CPU, имеет практически одинаковый функционал с корпоративной версией, просто развертывается и поэтому идеально подходит для образовательных целей.

На время написания пособия актуальной версией XE является Oracle Database Express Edition 18c, которая и рекомендуется как основной вариант СУБД для работы с пособием. Образы СУБД и инструкции по их установке для операционных систем Linux и Microsoft Windows можно найти в [2]. Также можно воспользоваться архивными версиями XE [3], например Oracle Database Express Edition 11gR2.

При установке СУБД необходимо запомнить пароль администратора и информацию о доступе к базе данных (обычно `localhost:1521`, но с версии 12c дополнительно есть еще и информация об имени сервиса для подключаемой базы (см. подраздел 2.1), например для XE 18c это `localhost:1521/XEPDB1`).

Существенным недостатком установки СУБД на учебную ПЭВМ является фоновое использование ресурсов системы, даже тогда, когда работа с СУБД не ведется. Для исключения этой проблемы рекомендуется использовать ручной режим включения-выключения служб СУБД [4].

Работать с СУБД можно с использованием консольной утилиты SQL*Plus, которая устанавливается вместе с СУБД, но рекомендуется использовать более функциональные бесплатные интегрированные средства разработки, такие как SQL Developer [5] и SQL Developer Data Modeler [6] (примеры использования приведены в пособии). Установка этих приложений проста – требуется разархивировать скачанные архивы в директорию, где располагаются программы, например, для ОС Windows в "C:\Program Files\Oracle\". Работа с этими средствами разработки идет в графическом интерфейсе пользователя и некоторые действия будут пояснены по ходу изложения материала.

1.2 Работа с СУБД Oracle в виртуальной машине

Для обучения новым технологиям Oracle предлагает использовать средство кросс-платформенной виртуализации Oracle VM VirtualBox [7] и ряд подготовленных для нее образов [8]. Такой подход позволяет не тратить время на установку и настройку программного обеспечения, а сразу работать с предустановленным набором средств. Другими достоинствами такого подхода являются быстрое и простое удаление программного обеспечения, возможность работы с программными средствами, которые лицензионным соглашением запрещено устанавливать напрямую. Недостатками такого подхода можно считать дополнительную нагрузку на ресурсы компьютера, которые тратятся на работу гостевой ОС образа, и невозможность обновления программного обеспечения образа.

Из готовых образов для виртуальной машины [8], существующих на момент написания пособия, рекомендуется использовать Database App Development VM [9], в состав которой входит гостевая ОС Oracle Linux 7, СУБД Oracle Database 19.3 и используемые в пособии средства разработки Oracle SQL Developer 19.1 и Oracle SQL Developer Data Modeler 19.1. При первом запуске образа в виртуальной машине необходимо ознакомиться с файлом информации о настройках, который открывается в терминале, подключить диск дополнения гостевой ОС (меню: «Устройства» → «Подключить образ диска Дополнений гостевой ОС...») и выполнить его автозапуск, для того чтобы после перезапуска гостевой ОС можно было настроить параметры дисплея виртуальной машины (меню: «Вид» → «Виртуальный экран 1» → «Запросить разрешение в ...») на удобное для работы разрешение. Возможно, при запуске гостевой ОС потребуются ввод имени пользователя и пароля, тогда для первого запуска с установкой дополнений желательно выполнить вход как `root`, а последующие сеансы работы лучше вести с помощью пользователя `oracle`. Все пароли в такой системе заданы как `oracle`.

1.3 Онлайн-среда разработки Live SQL

Еще один вариант работы, подходящий для изучения языка SQL, – бесплатный доступ к СУБД Oracle онлайн на сайте Live SQL (livesql.oracle.com). Фактически для работы с последней версией СУБД Oracle необходимы только браузер, учетная запись на сайте `oracle.com` и доступ в Интернет. Пользователь получает возможность выбора рабочей схемы, редактора для написания запросов на языке SQL, возможность просмотра предыдущих сеансов работы, доступ к документации и примерам. Такой вариант подходит для рассмотрения всех примеров на выборку данных для схемы HR, которые приведены далее в пособии. Основные недостатки онлайн-подхода: ограничения на работу с встроенными схемами, упрощенный функционал по сравнению с рекомендованными в подразделе 1.1 средствами разработки, зависимость от доступа в Интернет.

2 ФОРМИРОВАНИЕ РЕЛЯЦИОННОЙ СХЕМЫ ДАННЫХ

2.1 Введение в Oracle SQL

SQL – декларативный язык программирования четвертого поколения, предназначенный для работы с реляционными базами данных [10]. Язык SQL стандартизован ISO (www.iso.org). На момент создания пособия текущим стандартом SQL является версия ISO/IEC 9075:2016, в разработке находится версия ISO/IEC 9075:2019.

В основе SQL лежат операции реляционной алгебры и реляционное исчисление кортежей [11], но в связи с развитием технологий баз данных язык постепенно был дополнен рядом новых операций и расширений, в том числе реализованных для конкретных версий СУБД. В данном пособии рассматриваются основы языка SQL применительно к диалекту языка для СУБД Oracle (Oracle SQL). Изучать этот диалект языка SQL лучше по оригинальной документации Oracle, размещенной на сайте docs.oracle.com. Например, для СУБД Oracle 18c [12] описание диалекта языка SQL приведено в источнике [13].

К базовым элементам языка SQL относятся идентификаторы объектов, операторы языка и типы данных.

Рекомендуется создавать простые идентификаторы объектов, которые не должны быть зарезервированным словом языка. Первым символом такого идентификатора обязательно должна быть латинская буква, а последующие символы могут обозначаться латинской буквой, цифрой или символом подчеркивания. Простые идентификаторы нечувствительны к регистру, поэтому, например, идентификаторы `employees` и `EMPLOYEES` будут символизировать в Oracle один и тот же объект.

Oracle позволяет создавать идентификаторы в более свободной форме, задавая их в двойных кавычках (""). Такие идентификаторы не требуют соблюдения вышеизложенных правил, но при использовании их везде необходимо заключать в кавычки.

В ситуациях, когда имя объекта в запросе неоднозначно, необходимо полностью описывать путь к объекту, разделяя отдельные компоненты пути знаком точки. Например, запись `HR.employees.employee_id` будет представлять столбец `employee_id` таблицы `employees` схемы `HR`.

Операторы языка SQL строятся на основе синтаксиса, приближенного к предложениям английского языка [11, 13]. Обычно принято приводить описание синтаксических конструкций операторов в виде графических диаграмм [13] или в виде текстовых описаний в BNF-нотации (форме Бэкуса – Наура) [10]. Для BNF-нотации следует обратить внимание на следующие обозначения, которые описывают возможные варианты конструкции и не включаются в реальный оператор:

- имя синтаксического блока (`<метка> ::=`) – применяется для описания отдельной синтаксической конструкции с уникальным именем *метка*, которая обычно входит в состав других блоков;

- фигурные скобки (`{ }`) – для указания обязательных цельных элементов синтаксиса;
- квадратные скобки (`[]`) – для указания необязательных элементов синтаксиса;
- символ вертикальной линии (`|`) – для разделения элементов синтаксиса и указания того, что может быть использован только один из этих элементов;
- (`, . . .`) – для указания, что предшествующий элемент можно задать несколько раз в виде списка, все элементы которого разделяются запятыми;
- (`. . .`) – применяется для указания, что предшествующий элемент можно задать несколько раз в виде списка, все элементы которого разделяются пробелами.

СУБД Oracle поддерживает много различных типов данных, но при начальном ознакомлении рекомендуется использовать встроенные оригинальные типы данных (*Oracle built-in data types*). Описание часто применяемых типов данных приведено в таблице 1.

Таблица 1 – Краткое описание некоторых типов данных Oracle

Тип данных	Краткое описание
CHAR [(size [BYTE CHAR])]	Строка фиксированной длины от 1 до 2000 символов. Дополняется до заданного размера пробелами справа. Без указания размерности size длина строки равна 1 BYTE
VARCHAR2 (size [BYTE CHAR])	Строка переменной длины от 1 до 4000 символов, правые пробелы не хранятся
NUMBER [(precision [, scale])]	Число с плавающей точкой, где precision – число разрядов числа в диапазоне [1...38]; scale – число значимых разрядов от десятичной точки [-84...127], для которого производится округление (минус – в сторону целой части); диапазон абсолютных значений числа [10^{-130} ... 10^{126}]
FLOAT [(precision)]	Число с плавающей точкой, где precision – число разрядов числа в диапазоне [1...126]
DATE	Дата-время (включает поля YEAR, MONTH, DAY, HOUR, MINUTE и SECOND) в диапазоне [01.01.4712 BC...31.12.9999 AD] с точностью до секунды
TIMESTAMP [(fsp)]	Расширение типа DATE для хранения точных значений времени, где fsp – число цифр для хранения долей секунды, задается в диапазоне [0...9], по умолчанию fsp равно 6

2.2 Формирование реляционной схемы данных

2.2.1 Понятие схемы данных Oracle

Задачей данного руководства не является обучение администрированию СУБД, поэтому далее будут приводиться только краткие пояснения, необходимые для организации доступа к данным, которые используются для работы с примерами. Более подробно с администрированием СУБД XE 18c можно ознакомиться в [14].

По стандарту ISO SQL все объекты существуют в некоторой среде (*environment*), которая состоит из множества каталогов (*catalog*), в свою очередь включающих схемы данных (*schema*). Схема данных – именованная коллекция различных объектов базы данных, которые особым образом связаны друг с другом, а также с пользователями, имеющими права на работу с этими объектами.

В СУБД Oracle имя схемы соответствует имени пользователя (*user*), то есть каждая учетная запись пользователя связана только с одной схемой, в которой и расположены данные этого пользователя. Кроме имени, учетная запись пользователя также имеет пароль и набор привилегий для работы.

СУБД Oracle после установки предоставляет ряд автоматически создаваемых учетных записей пользователей:

1) учетные записи администратора предназначены для выполнения административных задач по управлению СУБД:

- SYS – для хранения словаря данных (данные этой схемы никогда не должны изменяться вручную);
- SYSTEM – для создания дополнительной административной информации (желательно работать в этой схеме только для выполнения административных задач, например для создания нового пользователя);

2) внутренние учетные записи СУБД создаются для того, чтобы отдельные функции СУБД или приложения могли иметь свои собственные данные, эти аккаунты нельзя удалять и модифицировать их данные;

3) учетные записи для готовых примеров схем пользователей (например, HR) – заранее подготовленные схемы, предназначенные для обучения и экспериментов.

Начиная с версии 12c СУБД Oracle основывается на новой архитектуре Oracle Multitenant [12, 16]. Данная архитектура содержит единую контейнерную супербазу (*multitenant Container DataBase (CDB)*) и множество подключаемых подбаз данных (*Pluggable DataBases (PDBs)*). Такой подход предназначен для повышения уровня интеграции баз данных предприятия, упрощения их администрирования и обслуживания, но структурно более сложен по сравнению с базой данных неконтейнерной (*non-CDB*) архитектуры, например в СУБД версии 11gR2. В неконтейнерной архитектуре все пользователи имеют уникальные имена, в архитектуре Oracle Multitenant есть два класса пользователей:

- общие пользователи (*common users*) – уникальные глобальные учетные

записи, которые могут подключаться к любым контейнерам при наличии соответствующих привилегий; к таким пользователям относятся учетные записи администратора SYS и SYSTEM, а также новые пользователи, которые должны создаваться с особыми префиксами имен (C## или c##) и, как правило, только в целях выполнения административных задач над несколькими контейнерами;

- локальные пользователи (*local users*) – учетные записи обычных пользователей, создаваемые и действующие в одной конкретной PDB; имена таких локальных пользователей, в отличие от общих, уникальны только в рамках одной PDB.

Для подключения к существующей учетной записи (создания сессии) необходимо указать следующие параметры соединения:

- имя пользователя и его пароль;
- сетевой адрес и порт слушателя (*listener*) – специальной службы, обеспечивающей связь клиента и хоста, на котором развернута база данных; по умолчанию задается как localhost/1521;
- имя сервиса – уникальный идентификатор для доступа к конкретному экземпляру (*instance*) базы данных [12].

Значения настроек слушателя и имя сервиса(ов) можно получить, выполнив в консоли команду `lsnrctl status`.

Для подключения в SQL Developer к учетной записи общего пользователя SYSTEM в CDB (или учетной записи администратора SYSTEM в non-CDB архитектуре) необходимо в закладке Connections создать новое соединение (рисунок 1), в котором следует задать имя SID или имя Service name следующим образом:

- для СУБД XE 18c или XE 11gR2 как XE (см. рисунок 1);
- для виртуальной машины с СУБД Enterprise Edition 19.3 как ORCLCDB;
- для виртуальной машины с СУБД Enterprise Edition 11gR2 как ORCL.

Здесь SID (*System Identifier*) – уникальное имя, которое однозначно идентифицирует конкретный экземпляр (*instance*) базы данных [12]. Service name – имя сервиса операционной системы, используемого для подключения к конкретному экземпляру базы данных. Под экземпляром базы данных понимается набор структур памяти и фоновых процессов, которые используются для управления файлами базы данных на конкретном хосте (*host*) [14].

Для проверки возможности установки соединения (см. рисунок 1) используется кнопка «Test», кнопка «Save» позволяет сохранить параметры соединения, для выполнения соединения используется кнопка «Connect». После установления соединения (рисунок 2) становятся доступны объекты схемы (1), действия над которыми можно выполнять как с помощью контекстных меню, так и в окне редактора SQL-команд (2), результаты работы SQL-команд выводятся в окно (3).

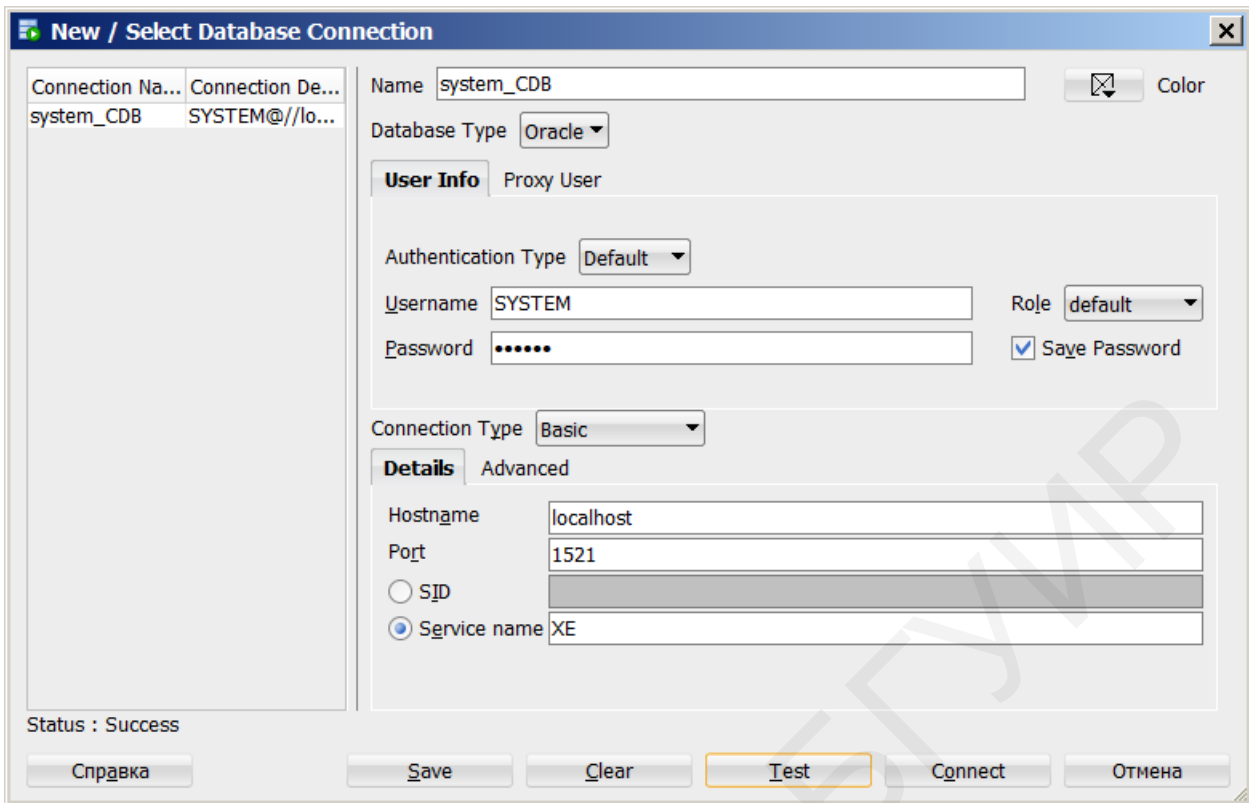


Рисунок 1 – Создание соединения с учетной записью общего пользователя SYSTEM в CDB

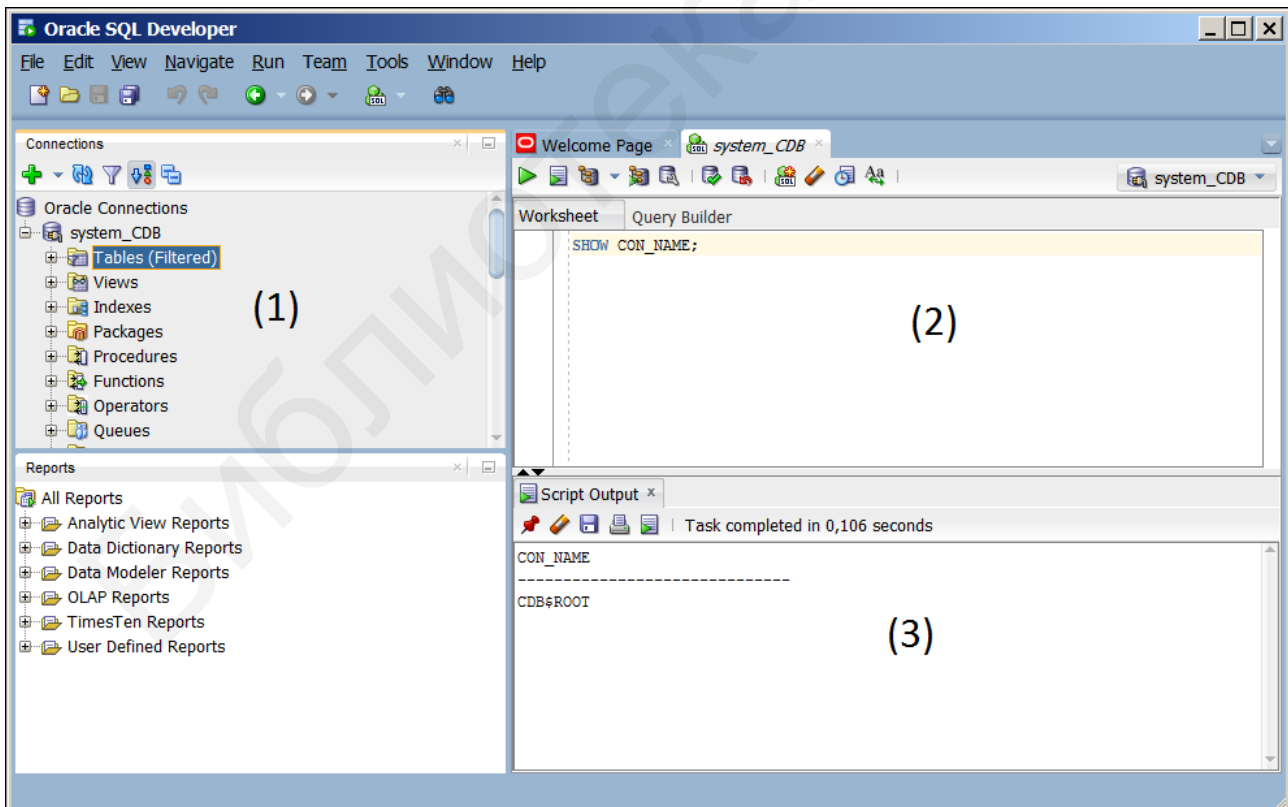


Рисунок 2 – Работа с объектами схемы после подключения к ней

Для подключения в SQL Developer к учетной записи общего пользователя SYSTEM в PDB необходимо в закладке Connections создать новое соединение, в котором следует задать имя Service name следующим образом:

- для СУБД XE 18c как XEPDB1;
- для виртуальной машины с СУБД Enterprise Edition 19.3 как ORCL.

Для создания новой учетной записи пользователя, размещаемой в PDB, необходимо создать соединение с SYSTEM в PDB, затем в этом соединении в списке элементов выбрать пункт «Other Users» и вызвать на нем правой кнопкой мыши контекстное меню, выбрать в контекстном меню пункт «Create User...», заполнить поля закладки «User» во всплывшем окне:

- имя и пароль пользователя выбираем сами, например NEW_USER и oracle соответственно, и заполняем поля «User Name», «New Password» и «Confirm Password»;
- поле «Default Tablespace» устанавливаем как USERS;
- поле «Temporary Tablespace» устанавливаем как TEMP.

Затем необходимо заполнить поля закладки «Granted Roles»: в столбце «Granted» выбрать пункты CONNECT и RESOURCE. Затем в закладке «Quotas» следует поставить галочку в столбце «Unlimited» для пункта USERS, что даст ему права на использование пространства под данные. Результат этих действий можно просмотреть в закладке «SQL» в виде SQL-операторов. Для завершения создания учетной записи нажать кнопку «Apply». Получить такой же результат можно, выполнив команды, указанные в примере 1 (рисунок 3).

Пример 1. Создание учетной записи пользователя NEW_USER с паролем oracle и назначением ему ролей CONNECT и RESOURCE.

```
CREATE USER "NEW_USER" IDENTIFIED BY "oracle"  
DEFAULT TABLESPACE "USERS"  
TEMPORARY TABLESPACE "TEMP"  
QUOTA UNLIMITED ON "USERS"  
ACCOUNT UNLOCK;  
GRANT "CONNECT" TO "NEW_USER";  
GRANT "RESOURCE" TO "NEW_USER";
```

Оператор CREATE USER предназначен для создания учетной записи, но работать с этой записью без определения для нее ряда прав будет невозможно. Оператор GRANT предназначен для выдачи пользователю определенных привилегий. Для выше указанного примера пользователю NEW_USER назначаются следующие привилегии на объекты его схемы, входящие в описание ролей CONNECT и GRANT: CREATE SESSION, CREATE CLUSTER, CREATE INDEXTYPE, CREATE OPERATOR, CREATE PROCEDURE, CREATE SEQUENCE, CREATE TABLE, CREATE TRIGGER, CREATE TYPE. Более подробно познакомиться с синтаксисом операторов CREATE USER и GRANT можно в [13], а с администрированием пользователей – в [17]. Подключение к учетной записи NEW_USER выполняется аналогично приведенному ранее подключению пользователя PDB.

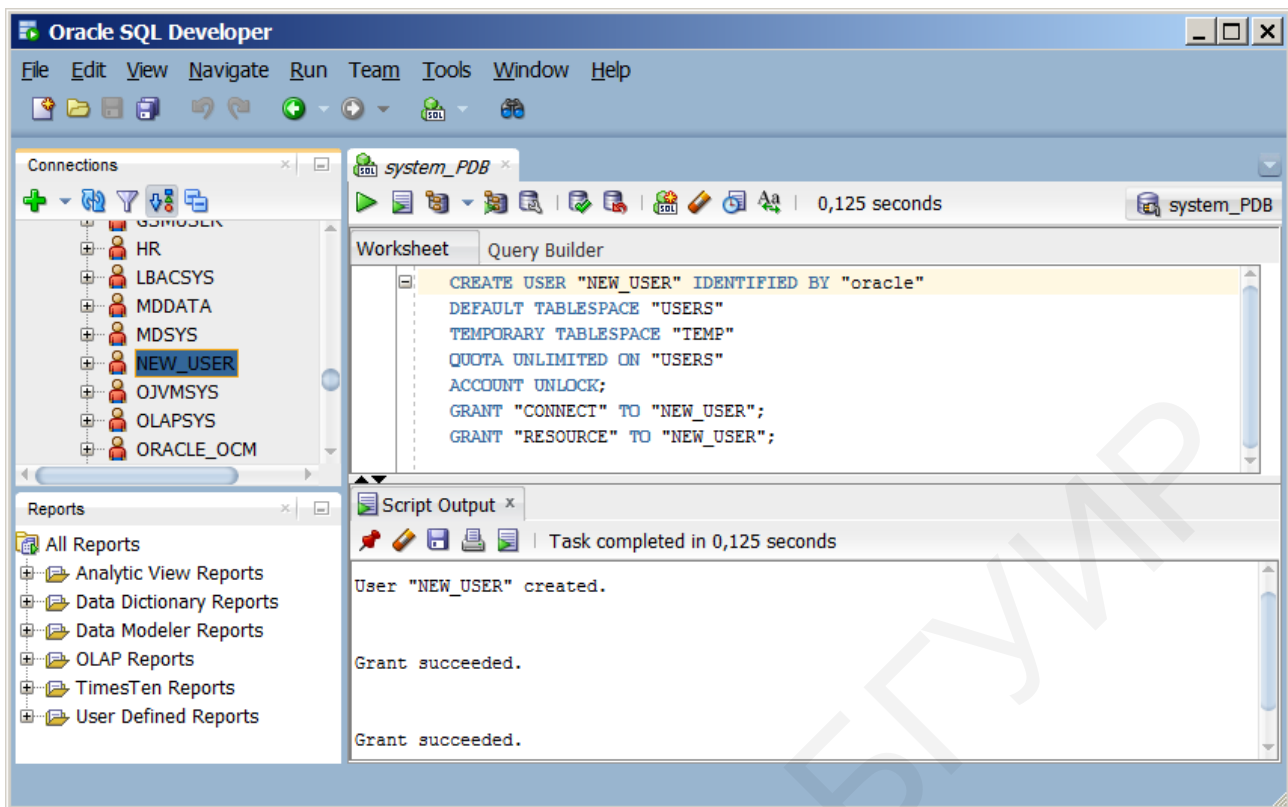


Рисунок 3 – Создание учетной записи NEW_USER с помощью скрипта

Для удаления учетной записи пользователя необходимо под администратором выполнить команду (CASCADE следует указывать для предварительного удаления всех объектов этого пользователя):

```
<удаление_пользователя>::=
  DROP USER имя_пользователя [CASCADE];
```

2.2.2 Управление объектами схемы

Схема данных пользователя может содержать множество разнотипных объектов [12], к которым относятся:

- таблица (*table*) – основной объект схемы, предназначенный для описания структуры объектов из реального мира с одинаковым набором свойств [10, 11] и хранения данных экземпляров таких объектов;
- индекс (*index*) – объект, предназначенный для ускорения доступа к строкам данных таблицы;
- последовательность чисел (*sequence*) – объект, предназначенный для генерирования последовательности целых чисел;
- синоним (*synonym*) – псевдоним для объекта схемы;
- представление (*view*) – объект, предназначенный для хранения временных результатов операций над базовыми таблицами.

Для управления структурой объектов схемы применяются следующие основные операторы DDL SQL:

- CREATE – создание объекта путем задания имени объекта, а также структуры объекта или его параметров;
- ALTER – модификация структуры объекта;
- DROP – удаление структуры объекта вместе со всеми хранимыми в нем данными.

Здесь и далее синтаксические конструкции SQL-операторов будут приводиться упрощенно в BNF-нотации (см. подраздел 2.1), так как полное описание ряда приводимых операторов достаточно объемно, включает в себя множество необязательных режимов и настроек, которые выходят за рамки данного пособия. С полным синтаксисом операторов лучше знакомиться по документации конкретной версии СУБД, например в [13] для версии 18с.

Создание структуры базовой реляционной таблицы выполняется с помощью следующей упрощенной формы оператора CREATE TABLE:

```

<создание_реляционной_таблицы> ::=
CREATE TABLE имя_таблицы
(
    { <описание_столбца> } [, ...]
    [ <ограничение_на_объект> [, ...] ]
);

<описание_столбца> ::=
имя_столбца тип_данных [ VISIBLE | INVISIBLE ]
[ DEFAULT выражение ] [ <ограничение_на_столбец> [...] ]

<ограничение_на_столбец> ::=
[ CONSTRAINT имя_ограничения ]
{ [NOT] NULL | UNIQUE | PRIMARY KEY | CHECK (условие) }
[ <состояние_ограничения> ]

<ограничение_на_объект> ::=
[ CONSTRAINT имя_ограничения ]
{ UNIQUE (имя_столбца [, ...]) |
  PRIMARY KEY (имя_столбца [, ...]) |
  FOREIGN KEY (имя_столбца [, ...])
  REFERENCES имя_таблицы [(имя_столбца [, ...])]
  [ ON DELETE { CASCADE | SET NULL } ] |
  CHECK (условие) }
[ <состояние_ограничения> ]

```

Фактически для создания базовых таблиц достаточно описать их структуру в виде списка описаний столбцов, каждое из которых состоит только из имен и типов данных (см. пример 2).

Пример 2. Создание простых таблиц.

```

CREATE TABLE groups (
    group_id NUMBER(6),
    name      VARCHAR2(20) );
CREATE TABLE students (

```

```

student_id NUMBER(6),
name        VARCHAR2(20),
group_id    NUMBER(6) );
CREATE TABLE subjects (
subject_id  NUMBER(6),
name        VARCHAR2(20) );
CREATE TABLE marks (
student_id  NUMBER(6),
subject_id  NUMBER(6),
mark        NUMBER(2),
mark_date   DATE );

```

Для реализации правил целостности реляционной модели [10, 11] необходимо дополнить оператор CREATE TABLE ограничениями, которые вводятся с помощью предложения CONSTRAINT, и делятся на ограничения, действующие на отдельный столбец, и на всю таблицу в целом.

Виды ограничений:

- NOT NULL – запрет неопределенных значений в столбце;
- UNIQUE – указание потенциального ключа размером от 1 до 32 столбцов (NULL-значения в его столбцах допустимы);
- PRIMARY KEY – указание первичного ключа размером от 1 до 32 столбцов (NULL-значения недопустимы во всех столбцах);
- FOREIGN KEY – указание ссылки (внешнего ключа), которая указывает на первичный ключ таблицы, имя которой указано после REFERENCES, если дополнительно не указан список целевых столбцов, который используется для ссылки на потенциальный ключ; если для ссылки указано ON DELETE CASCADE, то при удалении строки, на которую ссылается внешний ключ, будет удаляться и строка с этой ссылкой; если для ссылки указано ON DELETE SET NULL, то при удалении строки, на которую ссылается внешний ключ, эта ссылка будет установлена как неопределенная (NULL);
- CHECK – задание бизнес-правила в виде условия, которое должно быть истинно (*true*) для добавляемой или изменяемой строки данных таблицы; здесь *условие* – выражение, которое при вычислении имеет результат булева типа: либо истину (*true*), либо ложь (*false*) (см. пункт 3.1.3).

Если при создании ограничения не указывать фразу CONSTRAINT *имя_ограничения*, то ограничению будет автоматически назначено системное имя, начинающееся с префикса SYS_C.

Состояние ограничения (<*состояние_ограничения*>) позволяет указать, как будет проводиться проверка конкретного ограничения. По умолчанию ограничение включено – находится в состоянии ENABLE VALIDATE.

Также в описании столбца таблицы можно указать видимость столбца (VISIBLE или INVISIBLE) и значение по умолчанию (DEFAULT).

Пример 2 можно конкретизировать, добавив в него ограничения, на основании которых СУБД будет контролировать целостность данных (см. пример 3). При добавлении таких ограничений, как внешние ключи, необходимо соблюдать порядок выполнения операторов CREATE TABLE: сначала создаются таблицы, которые не содержат ссылок, а только потом те, в которых есть ссылки.

Пример 3. Создание таблиц с добавлением ограничений.

```
CREATE TABLE groups (
    group_id NUMBER(6) NOT NULL PRIMARY KEY,
    name      VARCHAR2(20) CONSTRAINT group_name_unique UNIQUE );
CREATE TABLE students (
    student_id NUMBER(6) NOT NULL PRIMARY KEY,
    name       VARCHAR2(20),
    group_id   NUMBER(6),
    FOREIGN KEY (group_id) REFERENCES groups
        ON DELETE SET NULL );
CREATE TABLE subjects (
    subject_id NUMBER(6) NOT NULL PRIMARY KEY,
    name       VARCHAR2(20) CONSTRAINT subject_name_unique UNIQUE);
CREATE TABLE marks (
    student_id NUMBER(6) NOT NULL,
    subject_id NUMBER(6) NOT NULL,
    mark       NUMBER(2) CHECK (mark >= 1 AND mark <= 10),
    mark_date  DATE,
    PRIMARY KEY (student_id, subject_id, mark_date),
    FOREIGN KEY (student_id) REFERENCES students
        ON DELETE CASCADE,
    FOREIGN KEY (subject_id) REFERENCES subjects
        ON DELETE SET NULL );
```

Для добавления описания таблиц и столбцов можно использовать оператор COMMENT в следующей форме:

```
<задание_комментария>::=
COMMENT ON { TABLE имя_таблицы |
            COLUMN имя_таблицы.имя_столбца }
IS строка_комментария ;
```

Здесь *строка_комментария* это строчная константа, которая задается в одинарных кавычках (''). Если ранее заданный комментарий требуется удалить, то данный оператор повторяется, и *строка_комментария* в нем задается пустой (' ') (см. пример 4).

Пример 4. Создание комментариев на таблицу и ее столбцы.

```
COMMENT ON TABLE groups IS 'Student''s group data';
COMMENT ON COLUMN groups.group_id
    IS 'Group''s unique number. Primary key column.';
COMMENT ON COLUMN groups.name IS '';
```


Изменение структуры таблицы выполняется оператором ALTER TABLE (приведены только важные для нас элементы оператора):

```
<изменение_структуры_таблицы> ::=
ALTER TABLE имя_таблицы
{ <изменение_свойств_таблицы> |
  <изменение_свойств_столбца> |
  <изменение_свойств_ограничения> } ;

<изменение_свойств_таблицы> ::=
RENAME TO новое_имя_таблицы

<изменение_свойств_столбца> ::=
{ { ADD (<описание_столбца> [, ...]) |
  MODIFY ( { имя_столбца [тип_данных] [DEFAULT выражение]
    [ <ограничение_на_столбец> [...] ] } [, ...] |
    { имя_столбца { VISIBLE | INVISIBLE } } [, ...] ) |
  DROP { COLUMN имя_столбца | (имя_столбца [, ...]) } } [...] } |
{ RENAME COLUMN имя_столбца TO новое_имя_столбца }

<изменение_свойств_ограничения> ::=
{ ADD (<ограничение_на_объект> [...]) |
  MODIFY { CONSTRAINT имя_ограничения |
    PRIMARY KEY | UNIQUE (имя_столбца [, ...]) }
  <состояние_ограничения> |
  RENAME CONSTRAINT имя_ограничения TO новое_имя_ограничения |
  DROP { CONSTRAINT имя_ограничения |
    PRIMARY KEY | UNIQUE (имя_столбца [, ...]) } [...] } ;
```

Использование оператора ALTER TABLE позволяет выполнить добавление ограничений на столбцы и таблицы без внедрения их в оператор CREATE TABLE. Однако с использованием оператора ALTER TABLE следует быть осторожным при наличии в таблицах данных.

Удаление структуры таблицы выполняется оператором DROP TABLE:

```
<удаление_таблицы> ::=
DROP TABLE имя_таблицы [CASCADE CONSTRAINTS] ;
```

Удаление структуры таблицы также удаляет все ее данные. Если на первичный или потенциальный ключ удаляемой таблицы существуют внешние ссылки, то удаление таблицы будет заблокировано. При указании опции CASCADE CONSTRAINTS выполняется удаление всех таких внешних ограничений ссылочной целостности.

При создании в таблице ограничений первичных или потенциальных ключей СУБД Oracle автоматически создает уникальные индексы для этих столбцов, а при удалении таблицы – автоматически удаляет эти индексы, поэтому операторы управления индексами в данном пособии не рассматриваются.

Для работы с данными на практике часто используются представления, но при их создании используется оператор SELECT, рассмотрение которого до-

статочно объемно (см. раздел 3), поэтому ниже приводятся только операторы управления представлениями:

```
<создание_представления> ::=
CREATE [ OR REPLACE ] VIEW имя_представления
[ ( { имя_столбца [ VISIBLE | INVISIBLE ]
    [ <ограничение_на_столбец> [...] ] |
    <ограничение_на_объект> } [, ...] ) ]
AS подзапрос ;
```

```
<изменение_структуры_представления> ::=
ALTER VIEW имя_представления
{ ADD <ограничение_на_объект> |
  MODIFY CONSTRAINT имя_ограничения { RELY | NORELY } |
  DROP { CONSTRAINT имя_ограничения | PRIMARY KEY |
    UNIQUE ( имя_столбца [, ...] ) } |
  COMPILE | READ { ONLY | WRITE } |
  { EDITIONABLE | NONEDITIONABLE } } ;
```

```
<удаление_представления> ::=
DROP VIEW имя_представления [CASCADE CONSTRAINTS] ;
```

Для формирования целочисленных значений суррогатных первичных ключей таблицы [10] будет полезен такой объект схемы, как последовательность чисел:

```
<создание_последовательности> ::=
CREATE SEQUENCE имя_последовательности
{ { INCREMENT BY | START WITH } целое_число |
  { MAXVALUE целое_число | NOMAXVALUE } |
  { MINVALUE целое_число | NOMINVALUE } |
  { CYCLE | NOCYCLE } } [...] ;
```

```
<изменение_последовательности> ::=
ALTER SEQUENCE имя_последовательности
{ INCREMENT BY целое_число |
  { MAXVALUE целое_число | NOMAXVALUE } |
  { MINVALUE целое_число | NOMINVALUE } |
  { CYCLE | NOCYCLE } } [...] ;
```

```
<удаление_последовательности> ::=
DROP SEQUENCE имя_последовательности ;
```

Основные параметры последовательности чисел:

- INCREMENT BY – интервал между соседними числами последовательности; может задаваться как положительное, так и отрицательное число; по умолчанию равен 1;
- START WITH – первое число в последовательности;
- MAXVALUE – максимальное число последовательности;
- NOMAXVALUE – $(10^{28}-1)$ для возрастающей или -1 для убывающей последовательности;

- MINVALUE – минимальное число последовательности;
- NOMINVALUE – 1 для возрастающей или $-(10^{27}-1)$ для убывающей последовательности;
- CYCLE – генерация по циклу;
- NOCYCLE – генерация значений только до максимального (минимального) значения.

Для каждого суррогатного ключа необходимо создать отдельную последовательность, которая будет источником значений этого ключа при вставке строк данных в таблицу (см. пример 5).

Пример 5. Создание последовательности.

```
CREATE SEQUENCE group_id_seq
START WITH 10 INCREMENT BY 10 MAXVALUE 5000 NOCYCLE;
```

Генерация значений первичных ключей с использованием последовательностей будет рассмотрена в пункте 2.2.3.

Синонимы предназначены для задания альтернативных имен объектов:

```
<создание_синонима> ::=
CREATE [ OR REPLACE ] [ PUBLIC ] SYNONYM имя_синонима
FOR имя_объекта ;
```

```
<удаление_синонима> ::=
DROP SYNONYM имя_синонима ;
```

В качестве имен объектов можно задать имя любого объекта схемы, кроме индекса. При указании слова PUBLIC синоним будет доступен всем пользователям, но для доступа к объекту конкретный пользователь все равно должен иметь соответствующие привилегии.

Ряд вышеописанных операторов можно собрать в единый текстовый скрипт, например: вначале удалить ранее созданные таблицы, затем создать таблицы по примеру 3 и в конце добавить объекты для обслуживания этих таблиц. Затем в редакторе SQL-команд SQL Developer для ранее созданной учетной записи NEW_USER этот скрипт можно выполнить, нажав кнопку «Run Script (F5)» (рисунок 4).

Такого же результата можно добиться, если использовать контекстные меню на списке объектов схемы и вводить данные в окнах вызываемых ими мастеров, но этот путь будет менее производительным, потому что один раз созданный скрипт можно использовать многократно, например, для восстановления объектов схемы.

Для детального изучения элементов (описаний столбцов, строк данных, ограничений, прав, индексов и т. п.) конкретной таблицы можно щелкнуть кнопкой мыши на ее названии в SQL Developer в списке объектов схемы (1) и использовать закладки (2) в ее описании (рисунок 5).

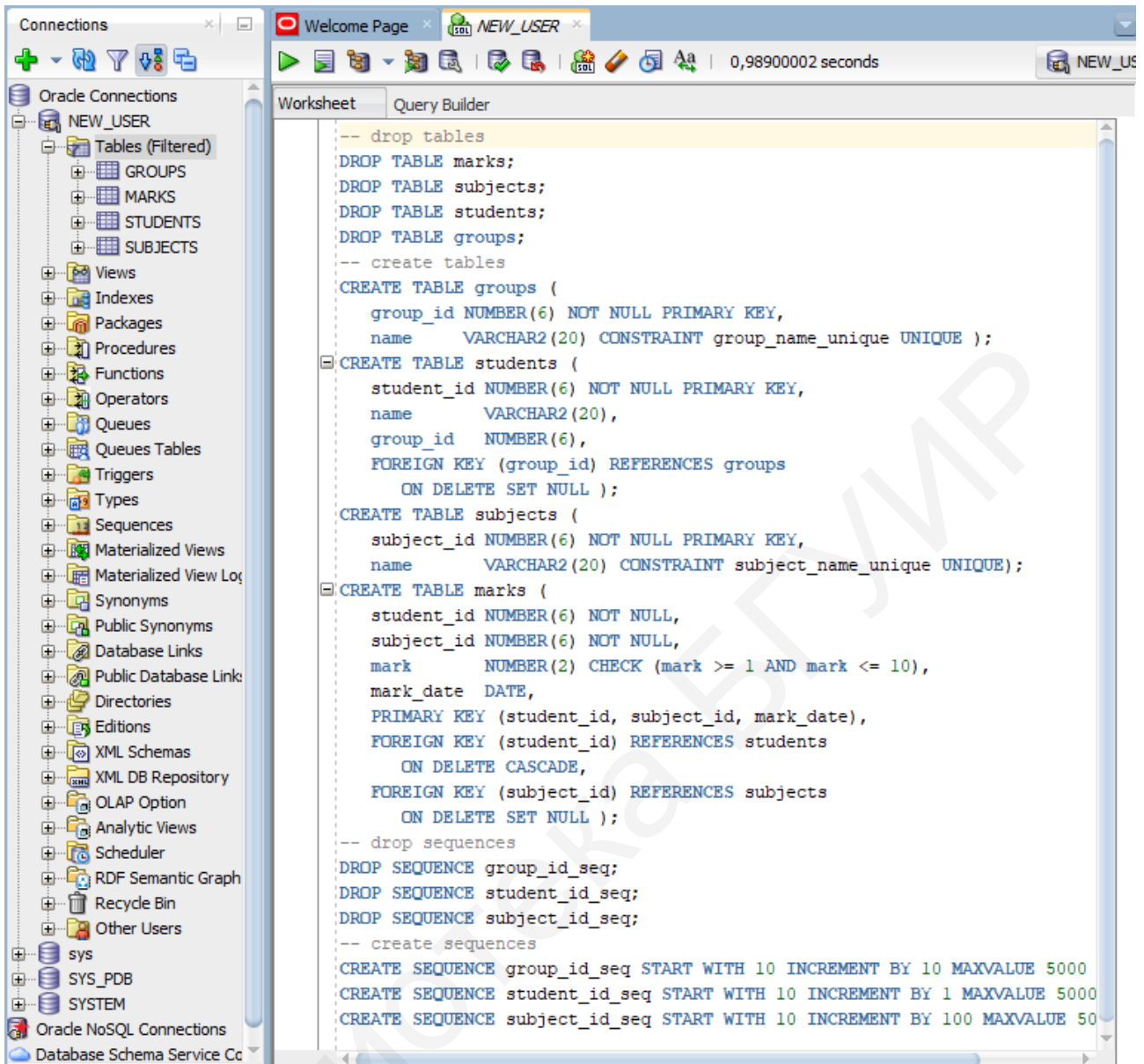


Рисунок 4 – Выполнение скрипта создания объектов схемы

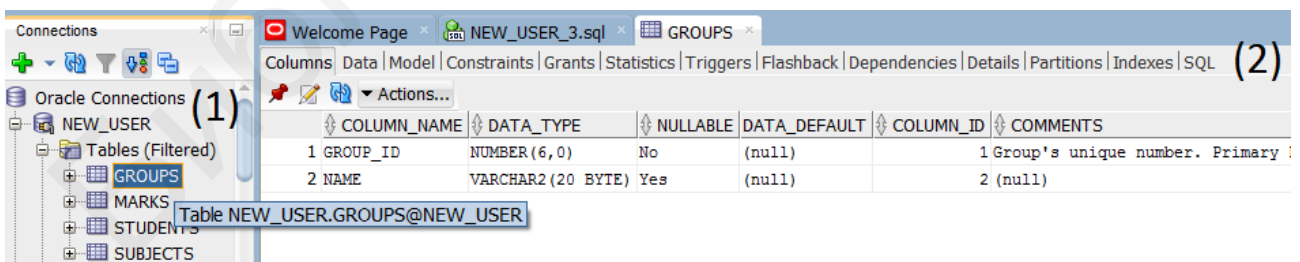


Рисунок 5 – Изучение параметров созданной таблицы

2.2.3 Заполнение таблиц схемы данными

Для заполнения таблиц данными в SQL Developer можно воспользоваться встроенным редактором (рисунок 6), который находится в закладке «Data» свойств таблицы. Добавление одной строки выполняется с помощью нажатия

на пиктограмму «Insert Row (Ctrl+I)», заполнения полей данными и нажатия на пиктограмму «Commit Changes (F11)». Отказ от внесения изменений в данные таблицы выполняется с помощью использования пиктограммы «Rollback Changes (F12)».

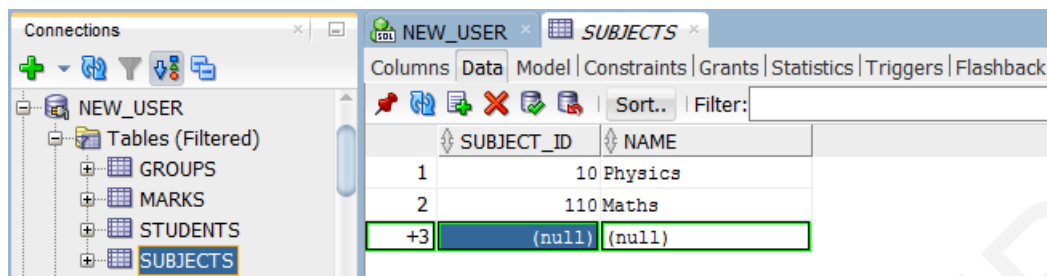


Рисунок 6 – Добавление данных в таблицу во встроенном редакторе

В примере 3 (см. пункт 2.2.2) таблицы содержат ограничения, поэтому заполнение данными не должно противоречить этим ограничениям, в противном случае при нажатии на пиктограмму «Commit Changes (F11)» возникнет ошибка, пояснения к которой будут выведены ниже в окно сообщений. Рекомендуется сначала заполнить данными таблицы, которые не имеют ссылок на другие таблицы, и только потом заполнять таблицы, ссылающиеся на них.

Заполнение таблиц данными также можно выполнить с помощью простого варианта команды INSERT для вставки строки данных в одну таблицу, расширенные варианты команды вставки приводятся в подразделе 4.1:

```
<вставка_одной_строки> ::=
  INSERT INTO имя_таблицы [ ( имя_столбца [, ...] ) ]
  VALUES ( выражение [, ...] );
```

Здесь:

- *имя_столбца* [, ...] – список столбцов, которым будут назначены конкретные значения при вставке; столбцы, отсутствующие в этом списке, получают значения по умолчанию или NULL, если это возможно для конкретного столбца приемника; если список столбцов не указан, то используются все столбцы таблицы в порядке, определенном при создании;
- *выражение* [, ...] – список данных, обычно констант, для вставки одной строки, который по числу выражений и их типам данных должен соответствовать заданному списку столбцов.

Использование оператора INSERT желательно сочетать с ранее созданными последовательностями чисел (см. пункт 2.2.2). Для доступа к данным последовательности используются псевдостолбцы:

- CURRVAL – текущее значение последовательности (требует хотя бы одно обращение к NEXTVAL);
- NEXTVAL – получение следующего значения в последовательности.

Работа с реляционной базой данных организована в СУБД в виде транзакций – наборов операторов, результат работы которых может быть либо за-

фиксирован в базе данных, либо полностью отклонен. Начало транзакции в СУБД Oracle выполняется неявно по выполнении одного из SQL-операторов, продолжительность транзакции не ограничивается. Транзакция завершается, если выполняется одно из следующих событий:

- выполнен специальный оператор управления транзакциями;
- выполняется DDL-оператор (CREATE, DROP, RENAME или ALTER);
- процесс, выполняющий транзакцию, завершается аварийно (при этом выполняется автоматический откат транзакции).

SQL Developer позволяет включить режим автоматической фиксации транзакций (*autocommit*), когда результаты выполнения каждого оператора модификации данных фиксируются неявно. Тем не менее, чтобы помнить об управлении транзакциями и для повышения производительности при выполнении больших скриптов вставки данных в таблицы, рекомендуется выключить автоматическое завершение транзакции (меню: «Tools» → «Preferences» → «Database» → «Advanced», убрать галочку с пункта «Autocommit» и нажать кнопку «OK») и завершать каждую транзакцию явно с помощью одного из следующих TCL-операторов:

- COMMIT – фиксация результатов транзакции;
- ROLLBACK – откат внесенных изменений.

Операторы для вставки данных в таблицы с учетом их ограничений и с использованием числовых последовательностей приведены в примере 6. Этот пример несколько утяжелен использованием оператора SELECT, который предназначен для поиска сгенерированных значений первичных ключей, но тем не менее читаем. Если все значения первичных ключей будут формироваться вручную, то списки выражений в VALUES упростятся до констант.

Пример 6. Вставка данных в таблицы.

```
INSERT INTO subjects (subject_id, name)
  VALUES (subject_id_seq.NEXTVAL, 'Physics');
INSERT INTO subjects (subject_id, name)
  VALUES (subject_id_seq.NEXTVAL, 'Maths');
INSERT INTO groups (group_id, name)
  VALUES (group_id_seq.NEXTVAL, '550501');
INSERT INTO groups (group_id, name)
  VALUES (group_id_seq.NEXTVAL, '550502');
INSERT INTO students (student_id, name, group_id)
  VALUES (student_id_seq.NEXTVAL, 'Ivanov',
    (SELECT group_id FROM groups WHERE name = '550501'));
INSERT INTO students (student_id, name, group_id)
  VALUES (student_id_seq.NEXTVAL, 'Petrov',
    (SELECT group_id FROM groups WHERE name = '550502'));
COMMIT;
INSERT INTO marks (student_id, subject_id, mark, mark_date)
  VALUES ((SELECT student_id FROM students
    WHERE name = 'Ivanov'),
    (SELECT subject_id FROM subjects
```

```

        WHERE name = 'Physics'),
    9,
    SYSDATE);
COMMIT;

```

Текущее состояние объектов схемы данных можно сохранить в SQL Developer, используя мастер экспорта базы данных (меню: «Tools» → «Database Export...»). В этом мастере для конкретной учетной записи можно задать описание экспортируемых объектов, формат представления результата и множество других настроек. В простейшем случае для сохранения только состояния таблиц достаточно выбрать экспорт данных в формате *insert* с сохранением в текстовый файл (рисунок 7). Затем на следующих шагах работы мастера выбрать в список объектов для экспорта все интересующие таблицы схемы в таком порядке, чтобы вставка данных не нарушала ограничения целостности, при этом для поиска таблиц схемы лучше воспользоваться кнопкой «Lookup». В результате выполнения будет получен файл, содержимое которого приведено в примере 7. Этот скрипт можно использовать для восстановления состояния таблиц схемы после модификации их данных.

Пример 7. Результат экспорта состояния схемы данных NEW_USER.

```

-----
-- File created - вторник-февраля-10-2020
-----
REM INSERTING into NEW_USER.GROUPS
SET DEFINE OFF;
Insert into NEW_USER.GROUPS (GROUP_ID,NAME) values
('10','550501');
Insert into NEW_USER.GROUPS (GROUP_ID,NAME) values
('20','550502');
commit;
REM INSERTING into NEW_USER.STUDENTS
SET DEFINE OFF;
Insert into NEW_USER.STUDENTS (STUDENT_ID,NAME,GROUP_ID) values
('10','Ivanov','10');
Insert into NEW_USER.STUDENTS (STUDENT_ID,NAME,GROUP_ID) values
('11','Petrov','20');
commit;
REM INSERTING into NEW_USER.SUBJECTS
SET DEFINE OFF;
Insert into NEW_USER.SUBJECTS (SUBJECT_ID,NAME) values
('10','Physics');
Insert into NEW_USER.SUBJECTS (SUBJECT_ID,NAME) values
('110','Maths');
commit;
REM INSERTING into NEW_USER.MARKS
SET DEFINE OFF;
Insert into NEW_USER.MARKS (STUDENT_ID,SUBJECT_ID,MARK,MARK_DATE)
values ('10','10','9',to_date('11.02.20','DD.MM.RR'));
commit;

```

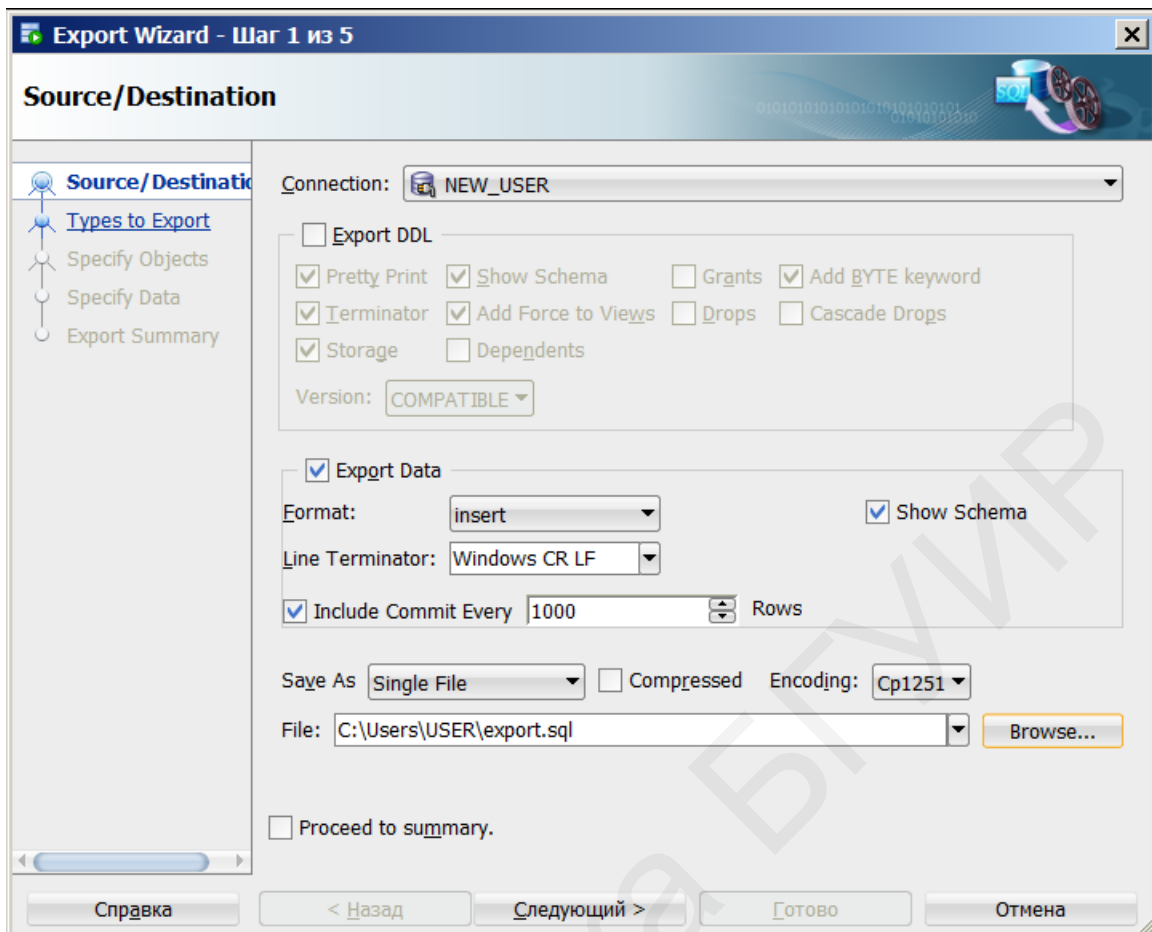


Рисунок 7 – Мастер экспорта базы данных

2.3 Создание объектов схемы по логической модели

Создание таблиц, описанное в пункте 2.2.2, можно автоматизировать с использованием SQL Developer Data Modeler.

Сначала требуется построить логическую модель (*Logical Model*) в нотации Баркера (Barker). Для этого после запуска SQL Developer Data Modeler необходимо выполнить следующие действия:

1) Правой кнопкой мыши вызвать контекстное меню на пункте «Logical Model» и выбрать в этом меню пункт «Show».

2) Задать требуемые типы объектов (*entities*), выбрав пиктограмму «New Entity» и выделив в окне редактора схемы прямоугольную область курсором для новой сущности. Сразу после добавления новой сущности всплывает окно ее свойств, в котором можно задать рабочее название и требуемые свойства сущности, включая ключевые поля. Для удобства размещения объектов можно включить привязку к сетке в контекстном меню редактора (правая кнопка мыши на странице редактора). Свойства объекта можно изменять, используя пункт «Properties» в контекстном меню объекта (щелчком правой кнопки мыши на выделенном объекте).

3) После задания объектов требуется задать типы отношений (*relations*), выбрав одну из пиктограмм: «New M:N Relation» или «New 1:N Relation» или в редких случаях «New 1:1 Relation» (нотация Баркера допускает только бинарные отношения) и последовательно указав курсором сущности, участвующие в новом отношении. Сразу после добавления нового отношения всплывает окно его свойств, в котором можно задать рабочее название, мощность связи и ее требуемые свойства. Свойства отношения также можно изменять, используя пункт «Properties» в контекстном меню отношения (щелчком правой кнопки мыши на выделенной связи).

Полученную логическую диаграмму (рисунок 8) можно сохранить в виде картинки, используя меню «File» → «Print Diagram» → «To Image File» (например, в формате .png).

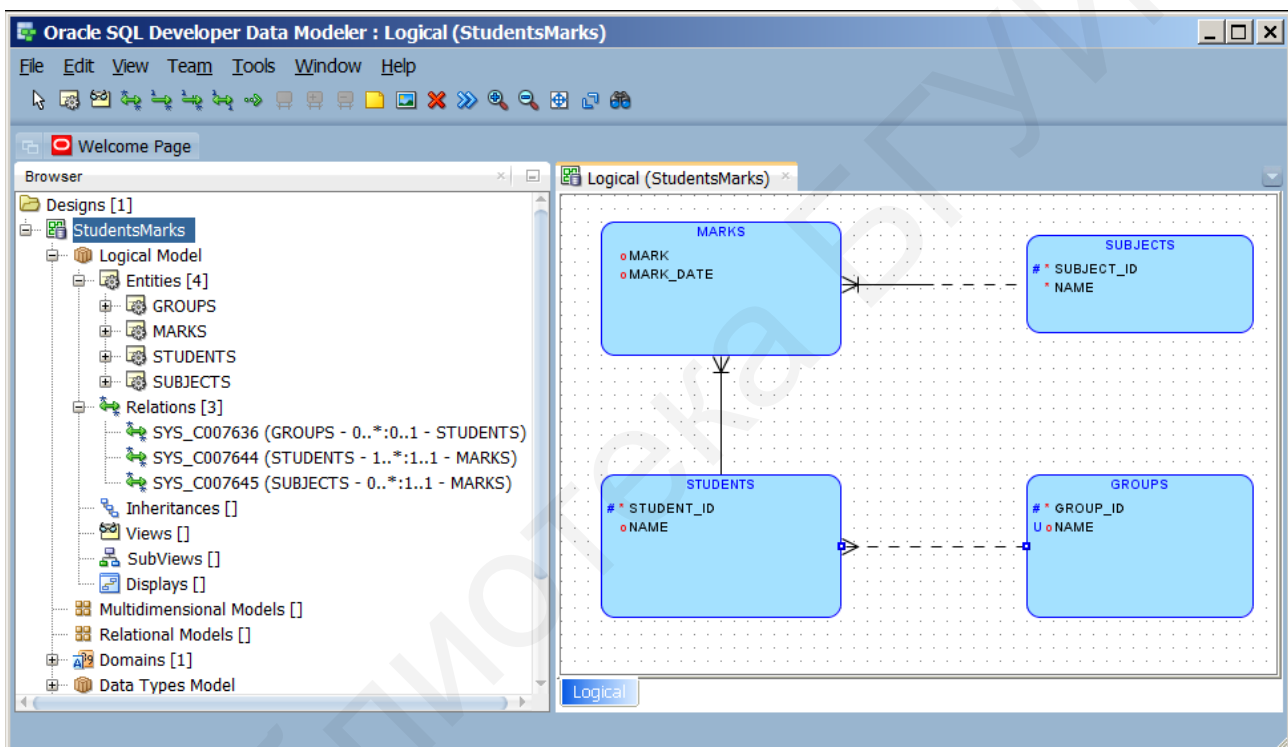


Рисунок 8 – Логическая диаграмма

Затем для преобразования логической диаграммы в реляционную диаграмму требуется использовать пиктограмму «Engineer to Relational Model», во всплывающем окне настроить свойства преобразования логической диаграммы в реляционную модель и нажать кнопку «Engineer».

Полученную реляционную модель (рисунок 9) также можно сохранить в виде картинки и экспортировать в DDL-файл, используя меню «File» → «Export» → «DDL File», а затем во всплывшем окне использовать кнопки «Generate», «OK» и «Save». Этот DDL-файл с набором SQL-команд можно скорректировать в редакторе кода, а затем выполнить в SQL Developer для создания объектов схемы данных (см. пункт 2.2.2). Особенности такого скрипта будет наличие операторов ALTER TABLE для добавления ограничений.

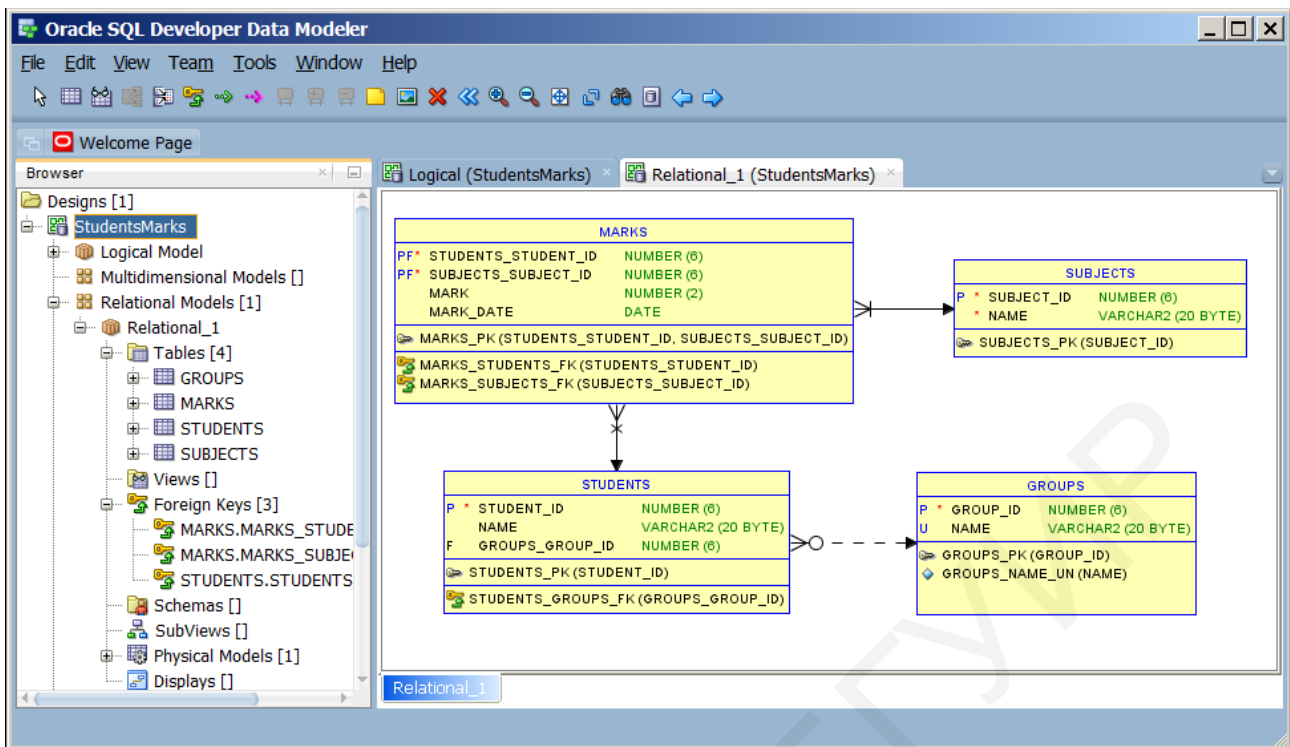


Рисунок 9 – Реляционная диаграмма

2.4 Схема HR

В данном подразделе дается информация, как получить доступ к учебной схеме HR и изучить ее объекты. Это необходимо, так как примеры по операторам выборки SELECT и модификации данных будут приводиться для объектов этой схемы (см. разделы 3 и 4).

Схема HR – одна из учебных схем Oracle. Готовые учебные схемы сформированы с учетом опыта проектирования реляционных схем данных, содержат не только таблицы, но и сопутствующие им объекты схемы, а также заполнены достаточно полной и реальной информацией, которая также важна при формировании запросов.

В СУБД XE 18c все учебные схемы размещены в создаваемой при установке СУБД PDB с именем сервиса XEPDB1. Сразу подключиться к схеме HR не получится, так как она изначально заблокирована. Чтобы снять эту блокировку, надо подключиться в SQL Developer к учетной записи администратора (например, SYSTEM) в PDB, затем в списке объектов схемы найти пункт «Other Users», в нем – подпункт «HR», вызвать на нем контекстное меню и выбрать пункт «Edit User...» (рисунок 10). Для снятия блокировки необходимо в окне «Edit User» в закладке «User» снять галочки в чек-боксах «Password Expired» и «Account is Locked», там же задать новый пароль в полях «New Password» и «Confirm Password», затем проверить в закладке «Granted Roles», заданы ли роли CONNECT и RESOURCE в столбце «Granted» и нажать кнопку «Apply». После выполнения этих действий можно отключиться

от учетной записи SYSTEM и подключиться к учетной записи HR, как это было описано в пункте 2.2.1.

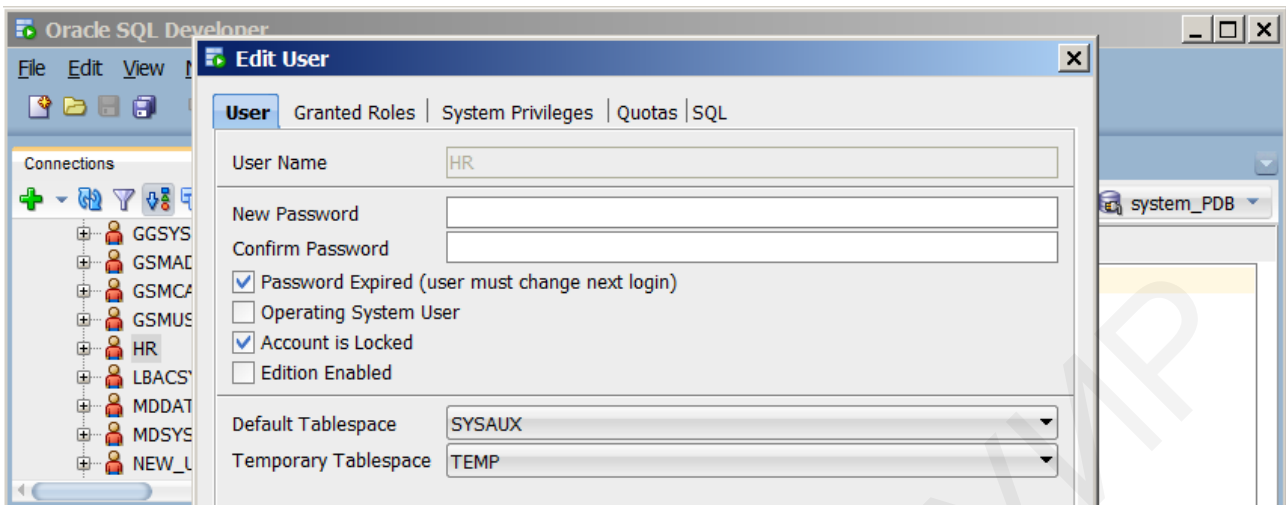


Рисунок 10 – Редактирование свойств учетной записи пользователя HR

Схема HR (Human Resources) предназначена для описания данных отдела кадров и в своих таблицах содержит информацию о сотрудниках (*employees*), должностях (*jobs*), отделах (*departments*), размещении отделов (*locations*), странах (*countries*), регионах (*regions*) и истории назначения сотрудников на должности (*job_history*). Для детального изучения таблиц и других объектов схемы можно выполнить навигацию по списку объектов в SQL Developer и рассмотреть их свойства (см. пункт 2.2.2 и рисунок 5).

Для лучшего понимания ссылок между таблицами можно в свойствах таблиц использовать закладку «Model» (см. рисунок 5), которая выводит диаграмму взаимосвязей между изучаемой таблицей и остальными таблицами схемы. Также можно построить полную реляционную диаграмму схемы, используя SQL Developer Data Modeler:

- 1) Запустить мастер «Data Dictionary Import Wizard», выбрав в меню «File» → «Import» → «Data Dictionary» → «Add», затем, нажав кнопку «Add», создать соединение со схемой HR и, выбрав это соединение, нажать кнопку «Next >>».
- 2) Выбрать схему HR и нажать кнопку «Next >>».
- 3) Выбрать все таблицы схемы и нажать кнопку «Next >>».
- 4) Нажать кнопку «Finish», полученную диаграмму (рисунок 11) сохранить, используя меню «File» → «Print Diagram» → «To Image File».

Практически все ссылки между таблицами (см. рисунок 11) сформированы на основе одноименных названий внешнего ключа ссылающейся таблицы и первичного ключа целевой таблицы, например FK HR.countries.region_id ссылается на PK HR.regions.region_id. Но две ссылки не отвечают этому принципу:

- departments.manager_id ссылается на employees.employee_id – данная связь показывает, какой сотрудник является руководителем отдела;
- employees.manager_id ссылается на employees.employee_id –

данная связь показывает, какой сотрудник является руководителем данного сотрудника.

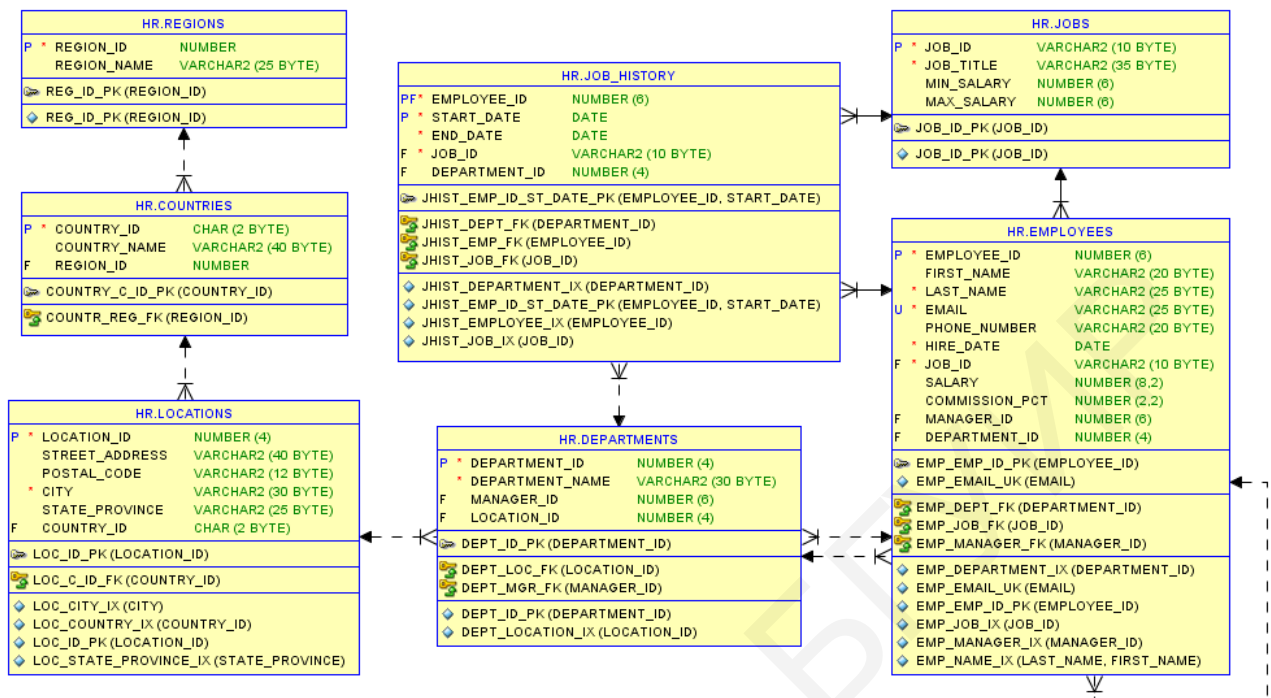


Рисунок 11 – Реляционная диаграмма схемы HR

При выполнении модификации данных учебной схемы лучше работать с ее копией. Дубль схемы можно создать на основе экспорта-импорта схемы данных (см. пункт 2.2.3), но такой вариант потребует исправления имени схемы в скрипте экспорта. Более простой вариант создания копии схемы данных в SQL Developer – использование Data Pump – мощного инструмента, предназначенного для быстрого перемещения данных. Например, для создания копии схемы HR с использованием Data Pump необходимо выполнить следующие действия:

1) Вывести окно DBA с помощью меню «View» → «DBA». DBA – это автоматически создаваемая административная роль в СУБД Oracle, которая содержит практически все системные привилегии и доступ к инструментам обслуживания базы данных [14].

2) В окне DBA с помощью кнопки «Add Connection...» создать соединение с учетной записью администратора, в нашем случае это соединение с учетной записью SYSTEM в PDB (см. пункт 2.2.1).

3) Развернуть соединение SYSTEM в закладке DBA. Выбрать в этом списке пункт «Data Pump» → «Export Jobs» и вызвать на нем контекстное меню, в котором выбрать «Data Pump Export Wizard...».

4) В мастере экспорта Data Pump выбрать экспорт схемы данных (рисунок 12) и нажать кнопку «Next >» для перехода к следующему этапу. В списке доступных схем выбрать схему HR и переместить ее в список выбранных схем. На следующем этапе выбрать все объекты схемы HR для экспорта. Все осталь-

ные шаги мастера, кроме шага «Output Files», можно оставить настроенными по умолчанию, подтверждая их с помощью кнопки «Next >». На шаге «Output Files» можно указать желаемое имя файла дампа, а также отключить компрессию данных, из-за которой возможны ошибки. На последнем шаге экспорта можно просмотреть отчет и нажать кнопку «Finish» и дождаться выполнения процесса.

5) Для импорта экспортированных данных в новую схему необходимо выбрать в списке пункт «Data Pump» → «Import Jobs» и вызвать на нем контекстное меню, в котором выбрать «Data Pump Import Wizard...», указав типом импорта схему (рисунок 13). Затем на следующем шаге выбрать схему HR из доступных схем в файле дампа. На шаге «Remapping» нужно переназначить импорт вместо схемы HR в схему HR_COPY в панели «Re-Map Schemas», для этого использовать кнопку «Add Row» и заполнить поля «Source» и «Destination» соответствующими названиями схем. На последнем шаге импорта можно просмотреть отчет, нажать кнопку «Finish» и дождаться выполнения процесса.

6) К новой схеме HR_COPY можно подключиться точно так же, как и к схеме HR, с тем же паролем и именем сервиса.

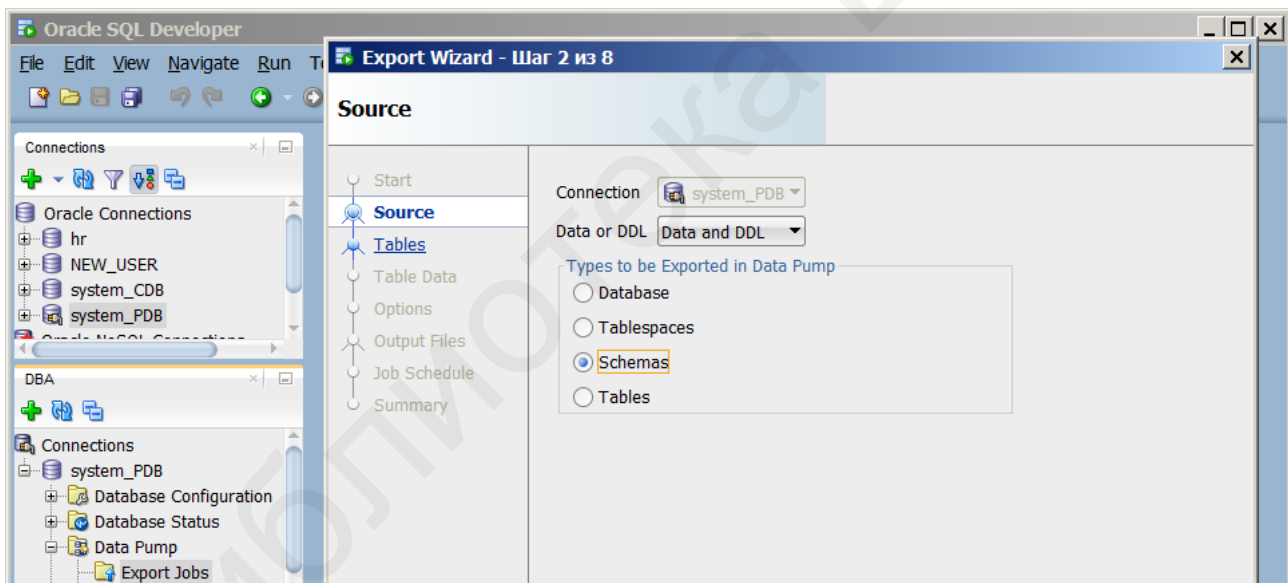


Рисунок 12 – Мастер экспорта данных Data Pump

В дальнейшем восстановление исходного состояния схемы HR_COPY можно проводить в SQL Developer с использованием мастера копирования схем (меню: «Tools» → «Database Copy...») из схемы HR с установкой полей «Source Connection» и «Destination Connection» как HR и HR_COPY соответственно, и выбором из опций копирования варианта «Objects Copy».

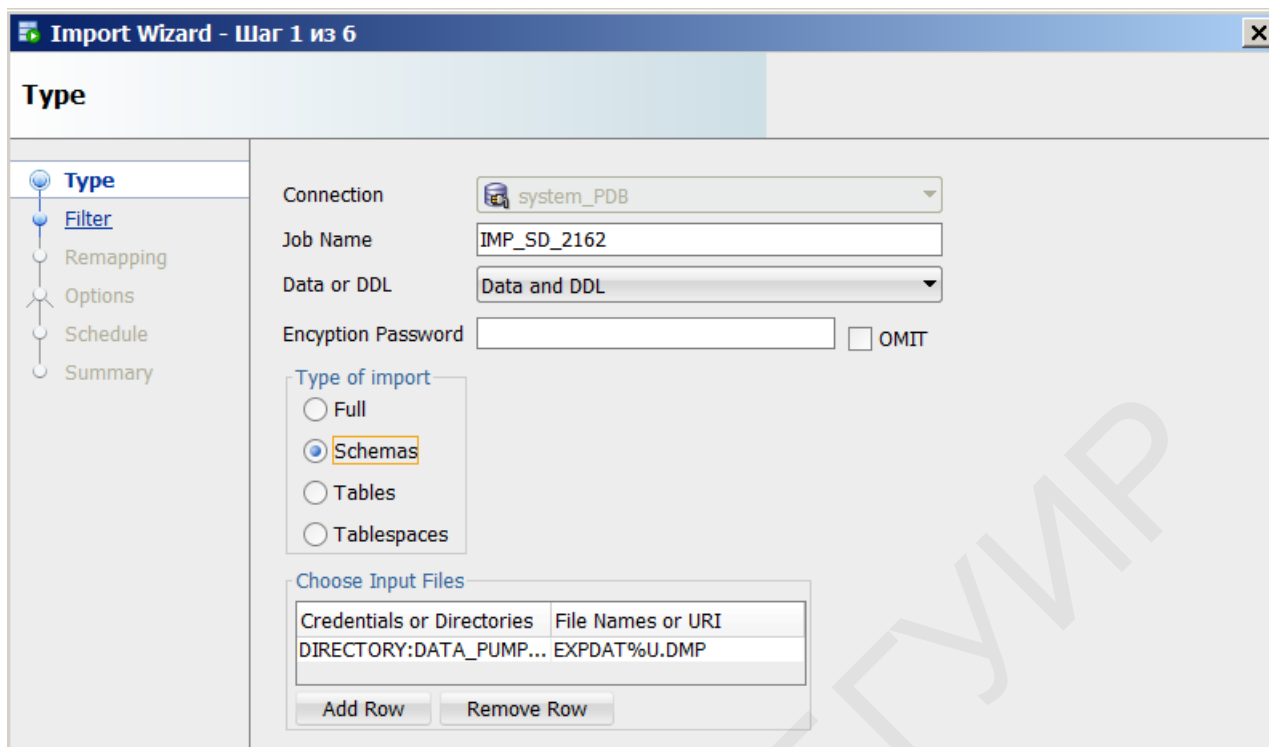


Рисунок 13 – Мастер импорта данных Data Pump

Сравнение двух схем на предмет отличия в структуре объектов можно выполнить в SQL Developer мастером сравнения (меню: «Tools» → «Database Diff...»).

3 ВЫБОРКА ДАННЫХ

3.1 Оператор выборки SELECT

Оператор SELECT выполняет выборку данных из таблиц базы данных и представляет результаты в виде одной финальной таблицы. При его работе данные в таблицах-источниках не изменяются. Структура оператора SELECT очень широкая, но при этом гибкая и зависит от конечной цели запроса. Ниже приведена упрощенная структура оператора SELECT, в которой указаны только те предложения, которые будут использованы в данном пособии:

```
<выборка_данных> ::=
SELECT [ { { DISTINCT | UNIQUE } | ALL } ] <список_выборки>
FROM { <таблица_данных> |
      <соединение_данных> |
      ( <соединение_данных> ) } [ , ... ]
[ <фильтр_строк_данных> ]
[ <иерархия_данных> ]
[ <группировка_данных> ]
[ <сортировка_данных> ]
```

В данном операторе только предложения SELECT и FROM являются обязательными.

В предложении SELECT ключевое слово DISTINCT или UNIQUE означает запрет дублирования строк результата выборки, а ALL – разрешение вывода дубликатов строк, которое действует по умолчанию.

Предложение FROM в варианте с одним источником данных обычно содержит только имя одной таблицы:

```
<таблица_данных> ::=
{ { [ имя_схемы. ] { имя_таблицы | имя_представления } } |
  ( подзапрос ) } [ имя_псевдонима_таблицы ]
```

Здесь:

- *имя_псевдонима_таблицы* – альтернативное уникальное имя для источника данных, которое может использоваться в запросе на выборку для замены основного имени и подчеркивания уникальности источника данных;
- *подзапрос* – запрос на выборку данных, вложенный в основной запрос (см. подраздел 3.2).

Работа с несколькими источниками данных приводится в пункте 3.1.5, до этого пункта все примеры будут приводиться только для выборки данных из одной таблицы.

Детализация предложений оператора выборки приводится в пунктах 3.1.1 – 3.1.5. Все приводимые примеры для схемы HR рекомендуется выполнить самостоятельно и изучить результаты их выполнения.

3.1.1 Представление результата (предложение SELECT)

Предложение SELECT начинает оператор выборки и служит для описания формата результирующей таблицы в виде подмножества столбцов, расположенных в заданном порядке, из исходного множества всех столбцов источника данных, что реализует такую унарную операцию реляционной алгебры, как проекция (*projection*) [10, 11]:

```
<список_выборки> ::=
{ * |
  { [ имя_схемы. ] { имя_таблицы | имя_представления }.* |
    имя_псевдонима_таблицы.* |
    имя_подзапроса.* |
    выражение [ [ AS ] имя_псевдонима_столбца ] } [, ...] }
```

Особенности описания списка выборки:

- * – означает выбор всех столбцов в порядке, определенном при создании таблицы или представления (см. пример 8);
- *выражение* – допустимое имя столбца из <таблица_данных>, допустимое имя псевдостолбца, константа, функция, скалярный подзапрос (см. подраздел 3.2) или допустимое выражение на основе этих элементов;
- *имя_псевдонима_столбца* – новое уникальное имя столбца, которое будет использоваться в результирующей таблице.

Пример 8. Выборка всех строк и столбцов из таблицы employees схемы HR.

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

В выражениях допустимо использовать имена существующих столбцов объектов из источника данных в формате [[*имя_схемы.*] {*имя_таблицы* | *имя_представления* }].*имя_столбца*. Также допустимо использовать в выражениях константы (числовые, строчные и NULL) и псевдостолбцы – столбцы, которые не содержатся в таблице, но позволяют получить специфичную информацию о данных, например:

- псевдостолбец ROWID – позволяет получить уникальный системный идентификатор строки таблицы; задается в формате [[*имя_схемы.*] {*имя_таблицы* | *имя_представления* }].ROWID ;
- псевдостолбец ROWNUM – возвращает порядковый номер строки в результате выборки; обычно применяется для ограничения числа строк результата в фильтре строк данных;
- псевдостолбцы последовательности CURRVAL и NEXTVAL (см. пункт 2.2.3);
- псевдостолбцы, связанные с иерархическими запросами (см. подраздел 3.5).

Для выполнения вычислений в выражениях используются унарные (+, -)

и бинарные операторы (*, /, +, -, ||), которые зависят от типов данных, а также круглые скобки. Вычисления в таких выражениях проводятся в рамках одной строки источника данных, поэтому имя столбца здесь означает данные из ячейки, относящейся в строке к указанному столбцу (см. пример 9).

Пример 9. Выборка данных из таблицы jobs схемы HR с использованием элементов простых выражений и операторов (рисунок 14).

```
SELECT ROWNUM,  
       job_title AS "Position",  
       '==>', -- string constant  
       max_salary AS "Maximal salary (EUR)",  
       (max_salary * 1.08) AS "Maximal salary (USD)"  
FROM HR.jobs;
```

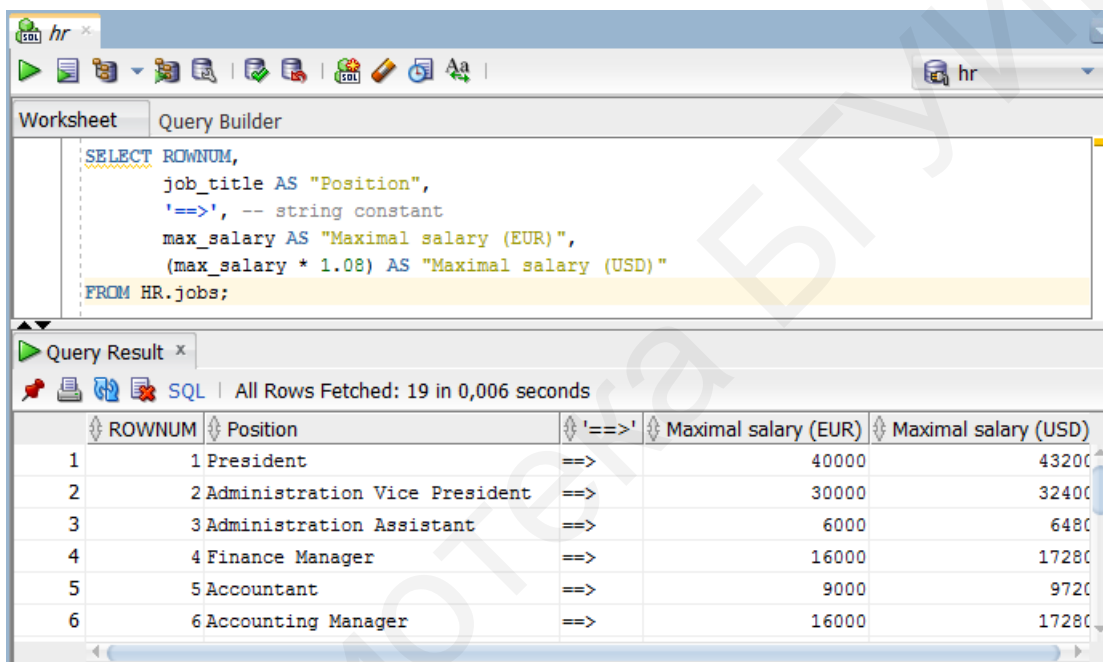


Рисунок 14 – Результат выполнения примера 9 в SQL Developer

Полученные результаты выполнения запроса, отображаемые в окне «Query Result» (см. рисунок 14), можно сохранить в файл, используя «Export Wizard». Для этого необходимо вызвать контекстное меню на таблице результатов запроса и выбрать в нем пункт «Export...». Затем в окне мастера экспорта требуется настроить требуемый формат экспортируемых данных, их описание, а также путь для сохранения файла. На следующем шаге мастера можно проверить заданные режимы экспорта и выполнить этот процесс. Данный вариант экспорта также можно выполнять для описаний таблиц схемы данных (см. рисунок 5).

Еще одним оператором, используемым в списке выборки, является CASE, позволяющий преобразовывать данные с использованием логики условного оператора IF...THEN...ELSE:

<условный_оператор>::=

```
CASE { <простое_условное_выражение> |
      <поисковое_условное_выражение>} [ ELSE выражение ] END;
```

```
<простое_условное_выражение>::=
  выражение { WHEN выражение THEN выражение } [...]
```

```
<поисковое_условное_выражение>::=
  { WHEN условие THEN выражение } [...]
```

Оператор CASE с использованием формы простых условных выражений для каждой пары WHEN...THEN выполняет сравнение выражений, стоящих до и после WHEN, и возвращает для первого найденного совпадения выражение, стоящее после THEN. Если в парах WHEN...THEN совпадений нет, то будет возвращено выражение, стоящее после ELSE. Если же ELSE отсутствует, то будет возвращено NULL-значение.

Оператор CASE с использованием формы поисковых условных выражений (см. пример 10) возвращает выражение, стоящее после THEN, для первой пары WHEN...THEN, для которой условие истинно. Если во всех парах WHEN...THEN условие ложно, то будет возвращено выражение, стоящее после ELSE. Если же ELSE отсутствует, то будет возвращено NULL-значение.

Пример 10. Использование формы поисковых условных выражений оператора CASE для выборки уровня зарплаты сотрудников.

```
SELECT last_name || ' ' || first_name AS "Employee",
       CASE WHEN (salary < 5000) THEN 'Low'
            WHEN (salary > 8000) THEN 'High'
            ELSE 'Middle'
       END AS "Salary level"
FROM HR.employees;
```

Также в выражениях используются SQL-функции, например, встроенные в СУБД скалярные функции. Скалярные функции (*single-row functions*) – функции, которые всегда возвращают одно значение для одной строки данных, поэтому они легко встраиваются в выражения списка выборки. Скалярных функций в СУБД Oracle 18c более 150 [13]. К основным группам скалярных функций можно отнести:

1) Функции для работы с числами – принимают в качестве аргументов числовые значения, результат – также числовое значение. К ним относятся, например:

- ABS (n) – модуль числа n;
- COS (n) – косинус угла n (угол задается в радианах);
- MOD (n, m) – остаток от деления числа n на число m;
- POWER (n, m) – степень m числа n;
- ROUND (n [, m]) – округление числа n до знака m (m может быть как положительным целым числом (округление выполняется по дробной части числа), так и отрицательным (округление по целой части числа));

- SQRT – квадратный корень числа n ;
- WIDTH_BUCKET(n, min, max, k) – номер интервала $[0, k+1]$ из диапазона $[min, max]$, разделенного на k равных интервалов, в который попадает число n ; в интервал с номером 0 попадают числа, меньшие min , а в интервал $k+1$ – числа, большие max .

2) Функции для работы со строками, которые возвращают символьный результат. К этим функциям относятся, например:

- CONCAT($s1, s2$) – склеивает строки $s1$ и $s2$ в одну строку (эквивалент оператора `||`);
- LOWER(s) – преобразует все символы строки s в нижний регистр;
- LPAD($s, n[, t]$) – возвращает строку длиной n символов, в которой строка s сдвинута влево, а недостающие правые символы заполнены из строки-шаблона t (если не указана строка t , то по умолчанию используется символ пробела);
- REPLACE(s, t, r) – заменяет в строке s все подстроки t на r ;
- SUBSTR(s, p, n) – возвращает подстроку длиной n символов из строки s , начиная с позиции p ;
- TRANSLATE($s, t1, t2$) – заменяет каждый найденный в строке s символ из набора $t1$ на соответствующий по позиции символ из набора $t2$;
- TRIM($[{LEADING|TRAILING|BOTH}][c] | c$ FROM s) – удаляет из строки s символ c в указанном расположении.

3) Функции для работы со строками, которые возвращают числовой результат. К этим функциям относятся, например:

- ASCII(c) – возвращает ASCII-код символа c ;
- INSTR($s, t[, p, n]$) – возвращает позицию n вхождения подстроки t в строке s , начиная с позиции p ;
- LENGTH(s) – возвращает длину строки s .

4) Функции для работы с датой и временем, например:

- ADD_MONTHS(d, n) – возвращает дату через n месяцев от даты d ;
- CURRENT_DATE – возвращает текущую дату в часовом поясе сеанса;
- LAST_DAY(d) – возвращает дату последнего дня месяца, в который входит дата d ;
- MONTHS_BETWEEN($d1, d2$) – возвращает число месяцев между датами $d1$ и $d2$;
- NEXT_DAY(d, s) – возвращает дату для следующего дня недели s относительно дня d ;
- SYSDATE – возвращает текущее системное время и дату сервера, на котором работает СУБД;
- TO_CHAR(${d|i}[, f[, 'nlsparam']]$) – преобразует дату d или интервал i в строку в формате f для параметров из национальной языковой поддержки $nlsparam$.

5) Функции для преобразования данных:

- `GREATEST(e [, . . .])` – возвращает выражение с наибольшим значением из списка выражений `e`;
- `LEAST(e [, . . .])` – возвращает выражение с наименьшим значением из списка выражений `e`.

6) Функции для преобразования типов данных, например:

- `CAST(e AS t [DEFAULT v ON CONVERSION ERROR] [, f [, 'nlsparam']])` – преобразование выражения `e` в тип `t`, при ошибке возвращается значение `v`, для ряда преобразований можно указывать формат `f` для параметров из национальной языковой поддержки `nlsparam`;
- `TO_CHAR(n [, f [, 'nlsparam']])` – преобразует число `n` в строку, используя формат `f` для параметров из национальной языковой поддержки `nlsparam`;
- `TO_DATE(s [DEFAULT v ON CONVERSION ERROR] [, f [, 'nlsparam']])` – преобразует строку `s` в дату, используя формат `f` для параметров из национальной языковой поддержки `nlsparam`, при ошибке возвращается значение `v`.

7) Функции для кодирования и декодирования данных, например:

- `DECODE(e, {s, r} [, . . .] [, d])` – возвращает значение `r` из первой пары значений `{s, r}`, если данные из источника `e` совпадают со значением `s` этой пары, иначе возвращается значение `d`.

8) Функции для работы с NULL-значениями, например:

- `NULLIF(e1, e2)` – возвращает `NULL`, если значение выражения `e1` равно `e2`, иначе возвращает `e1`;
- `NVL(e1, e2)` – возвращает `e2`, если значение выражения `e1` равно `NULL`, иначе возвращает `e1`;
- `NVL2(e1, e2, e3)` – возвращает `e3`, если значение выражения `e1` равно `NULL`, иначе возвращает `e2`.

9) Функции для работы с системными данными, например:

- `SYS_CONTEXT('ns', 'p' [, l])` – возвращает строку длиной `l` значения параметра `p` в пространстве `ns`, допустимые значения `ns` и `p` указаны в [13];
- `UID` – уникальный числовой идентификатор пользователя;
- `USER` – имя пользователя;
- `USERENV('p')` – значение параметра `p` текущей сессии пользователя, допустимые значения `p` указаны в [13].

Несмотря на то что ряд скалярных функций не связан с таблицами, в запросе на выборку, в котором они используются, обязательно должно присутствовать предложение `FROM`. Чтобы не применять в таких запросах таблицы

схемы пользователя и гарантированно получить только одно значение результата, можно воспользоваться таблицей `dual` (см. пример 11). Эта таблица расположена в словаре данных СУБД и относится к учетной записи `SYS`, но доступна всем остальным учетным записям. Таблица `dual` имеет только один столбец с именем `dummy` и одну строку, содержащую значение `X`. Данная таблица также полезна для вычисления константных выражений (см. пример 12).

Пример 11. Получение уникального числового идентификатора пользователя и его имени для текущего сеанса.

```
SELECT UID, USER FROM dual;
```

Пример 12. Вычисление длины гипотенузы прямоугольного треугольника с длиной катетов 6 и 8 единиц.

```
SELECT SQRT(POWER(6,2)+POWER(8,2)) AS "Hypotenuse"  
FROM dual;
```

3.1.2 Сортировка результатов (предложение `ORDER BY`)

Результат выполнения запроса на выборку включает строки данных в порядке их расположения в текущем состоянии источника данных. Для лучшего анализа оператором рекомендуется результат выборки упорядочить с помощью предложения `ORDER BY`:

```
<сортировка_данных> ::=  
ORDER [ SIBLINGS ] BY  
{ { выражение | индекс_столбца | имя_псевдонима_столбца }  
  [ ASC | DESC ]  
  [ NULLS FIRST | NULLS LAST ] } [, ...]
```

Особенности синтаксиса предложения сортировки:

- `SIBLINGS` – специальный вид сортировки с сохранением иерархического порядка (применим только для иерархических запросов с `CONNECT BY`, см. подраздел 3.5);
- *выражение* – имя столбца результирующей таблицы;
- *индекс_столбца* – целочисленное значение положения столбца в результирующей таблице, счет позиций ведется с 1;
- *имя_псевдонима_столбца* – уникальное имя столбца, задаваемое в списке выборки;
- `ASC` – сортировка данных столбца по возрастанию (по умолчанию);
- `DESC` – сортировка данных столбца по убыванию;
- `NULLS FIRST` или `NULLS LAST` – порядок расположения `NULL`-значений при сортировке данных столбца (в начале или в конце соответственно); по умолчанию `NULL`-значение считается наибольшим среди данных столбца.

Сортировка с использованием одного столбца обычно используется для анализа данных по одному критерию, например, по зарплате сотрудников (см. пример 13).

Пример 13. Выборка данных по зарплате сотрудников в порядке убывания.

```
SELECT last_name, salary
FROM HR.employees
ORDER BY salary DESC;
```

При использовании сортировки по нескольким столбцам (см. пример 14) сначала выполняется сортировка строк данных по первому столбцу в списке сортировки. Затем строки с одинаковым значением в первом столбце списка сортировки упорядочиваются по второму столбцу списка сортировки и так далее.

Пример 14. Выборка данных по занятости сотрудников в отделах.

```
SELECT last_name, department_id, salary
FROM HR.employees
ORDER BY 2 ASC NULLS FIRST, 1 ASC;
```

Индексы столбцов в списке сортировки (см. пример 14) используются в основном для того, чтобы не дублировать длинные имена столбцов. Если по каким либо причинам будет изменен список выборки в таком запросе, то список сортировки также необходимо скорректировать.

3.1.3 Фильтр строк данных (предложение **WHERE**)

Фильтр строк данных реализует такую операцию реляционной алгебры, как выборка (*selection*) [10, 11], – унарную операцию, которая помещает в результирующую таблицу только такое подмножество строк из всех строк источника данных, для которых *условие* выборки истинно (*true*):

```
<фильтр_строк_данных> ::=
  WHERE условие
```

В СУБД Oracle *условие* задается как набор условий, связанных логическими операторами «И» (AND), «ИЛИ» (OR) и «НЕТ» (NOT). Отдельные условия могут быть заключены в круглые скобки для устранения запутанности. Возможны следующие результаты условия: истина (TRUE), ложь (FALSE) и неопределенность (UNKNOWN). Значение неопределенности (UNKNOWN) возникает при выполнении сравнений с NULL-значением с использованием простых операторов сравнения (см. <условие_простое>). Таблицы истинности для логических операторов приведены в таблицах 2–4.

Таблица 2 – Таблица истинности логического оператора NOT

--	TRUE	FALSE	UNKNOWN
NOT	FALSE	TRUE	UNKNOWN

Таблица 3 – Таблица истинности логического оператора AND

AND	TRUE	FALSE	UNKNOWN
TRUE	TRUE	FALSE	UNKNOWN
FALSE	FALSE	FALSE	FALSE
UNKNOWN	UNKNOWN	FALSE	UNKNOWN

Таблица 4 – Таблица истинности логического оператора OR

OR	TRUE	FALSE	UNKNOWN
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	UNKNOWN
UNKNOWN	TRUE	UNKNOWN	UNKNOWN

Далее приведены синтаксические конструкции условий, часто используемых в операторе выборки:

1) Простые условия задаются практически так же, как и в большинстве языков программирования; выражения для сравнения должны быть совместимы по типам значений:

```
<условие_простое> ::=
{ выражение { = | != | ^= | <> | < | > | <= | >= } выражение } |
{ (выражение[, ...]) { = | != | ^= | <> }
  ( { (выражение [, ...]) [, ...] } | подзапрос ) }
```

2) Кванторные условия позволяют использовать сравнение выражения с множествами значений с использованием обычных операторов сравнения; выражения для сравнения должны быть совместимы по типам значений:

```
<условие_кванторное> ::=
{ выражение { = | != | ^= | <> | < | > | <= | >= }
  { ALL | ANY | SOME } ( {выражение [, ...]} | подзапрос ) } |
{ (выражение[, ...]) { = | != | ^= | <> } {ALL | ANY | SOME}
  ( { (выражение [, ...]) [, ...] } | подзапрос ) }
```

Здесь:

- ANY (SOME) – сравнение должно выполняться хотя бы для одного значения из данного множества; квантор ANY (SOME) сцепляет все условия сравнения с помощью логического OR; если множество значений пустое, то результат всегда равен FALSE;
- ALL – сравнение должно выполняться одновременно для всех значений из данного множества; квантор ALL сцепляет все условия сравнения с помощью логического AND; если множество пустое, то результат всегда равен TRUE; если в таком варианте условия во множестве будет находиться NULL-значение, то результат сравнения никогда не будет TRUE.

3) Конструкция <условие_для_чисел_с_плавающей_точкой> позволяет выполнить проверку числа на соответствие неопределенному результату (NaN) или бесконечности (INFINITE):

`<условие_для_чисел_с_плавающей_точкой> ::=
выражение IS [NOT] { NAN | INFINITE }`

4) Конструкция `<условие_для_диапазона>` выполняет проверку вхождения числа, строки или даты-времени в диапазон `[граница_1, граница_2]`:

`<условие_для_диапазона> ::=
выражение [NOT] BETWEEN граница_1 AND граница_2`

5) Конструкция `<условие_для_множества>` выполняет проверку вхождения значения выражения во множество значений, заданных списком или возвращаемых подзапросом:

`<условие_для_множества> ::=
{ выражение [NOT] IN ({ выражение [, ...] } | подзапрос) } |
{ ({ выражение [, ...] }) [NOT] IN
({ (выражение [, ...]) [, ...] } | подзапрос) }`

Здесь, например:

- `expr IN (10, 20)` эквивалентно `(expr=10) OR (expr=20)`;
 - `expr NOT IN (10, 20)` эквивалентно `(expr!=10) AND (expr!=20)`;
- если в таком варианте условия во множестве будет находиться NULL-значение, то весь результат сравнения никогда не будет TRUE.

6) Конструкция `<условие_для_таблицы>` выполняет проверку наличия данных в таблице, возвращаемой подзапросом; условие будет TRUE, если подзапрос возвращает хотя бы одну ячейку данных:

`<условие_для_таблицы> ::=
[NOT] EXISTS (подзапрос)`

7) Конструкция `<условие_для_символьного_шаблона>` выполняет проверку совпадения выражения с символьным шаблоном; в большинстве случаев необходимо использовать оператор LIKE, остальные варианты нужны для работы с разными формами кодирования символов:

`<условие_для_символьного_шаблона> ::=
выражение [NOT] { LIKE | LIKEC | LIKE2 | LIKE4 }
шаблон [ESCAPE стоп_символ]`

Правила задания шаблона для сравнения:

- символ процента (%) – любая последовательность символов;
- символ подчеркивания (_) – любой одиночный символ;
- `стоп_символ` – символ для возможности использования в поиске спецсимволов (% и _).

8) Конструкция `<условие_для_NULL_значений>` выполняет проверку совпадения выражения с NULL-значением.

`<условие_для_NULL_значений> ::=
выражение IS [NOT] NULL`

Заметим, что пустая строка символов (' ') в СУБД Oracle также трактуется как NULL-значение. Однако числовой нуль (0) и NULL-значение не эквивалентны. Использовать NULL-значения в выражениях с арифметическими операциями не следует, так как это дает в результате NULL-значение (рисунок 15). В таком случае необходимо использовать либо фильтр NULL-значений, либо скалярную функцию, например NVL (см. пункт 3.1.1).

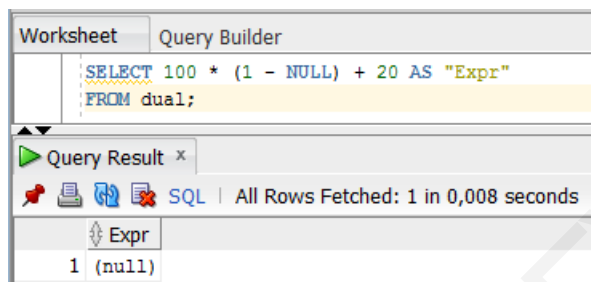


Рисунок 15 – Вычисление арифметического выражения с NULL-значением

При создании выражений необходимо помнить про приоритет операторов и условий и, если это необходимо, вносить круглые скобки для разделения сложных конструкций на более простые и вычисляемые в первую очередь. СУБД Oracle выполняет операции с одинаковым приоритетом слева направо. Приоритет операторов и условий (от высокого к низкому) в СУБД Oracle следующий:

- 1) унарные операторы + и -, PRIOR, CONNECT_BY_ROOT
- 2) *, /
- 3) бинарные операторы + и -, ||
- 4) =, !=, <>, <, >, <=, >=
- 5) IS [NOT] NULL, LIKE, [NOT] BETWEEN, [NOT] IN, EXISTS
- 6) NOT
- 7) AND
- 8) OR

Примеры 15 и 16 демонстрируют варианты реализации фильтра строк данных с использованием различных конструкций условий.

Пример 15. Выборка данных о сотрудниках с использованием условий фильтрации по символному шаблону фамилии, диапазону зарплаты и не NULL-значению комиссионного процента.

```
SELECT last_name, salary  
FROM HR.employees  
WHERE (last_name LIKE '%r%') AND  
      (salary BETWEEN 12000 AND 17000) AND  
      (commission_pct IS NOT NULL)  
ORDER BY salary;
```

Пример 16. Выборка данных о сотрудниках с использованием условия для множества номеров отделов, к которым они принадлежат, и простого условия по зарплате.

```
SELECT last_name, department_id
FROM HR.employees
WHERE department_id IN (10, 20, 30) AND salary > 5000
ORDER BY 2, 1;
```

3.1.4 Агрегатные функции, группировка данных и фильтр групп (предложения GROUP BY и HAVING)

Агрегатные функции (*aggregate functions*) в отличие от скалярных функций формируют один результат для всех строк таблицы, если в операторе выборки не используется предложение GROUP BY, или для каждой группы, формируемой предложением GROUP BY. В качестве параметров такие функции получают выражение, обычно включающее имена некоторых столбцов, по набору значений в которых и формируется единое результирующее значение. Практически все агрегатные функции игнорируют NULL-значения, но если источник данных не содержит ни одной строки или используемый набор содержит только строки с NULL-значением, то функция возвращает NULL-значение. Часть агрегатных функций может быть использована в аналитической (*analytic*) форме для упрощения и ускорения аналитических запросов, но в данном пособии такие запросы рассматриваться не будут.

Агрегатные функции в операторе SELECT могут располагаться в списке выборки, а также в предложениях ORDER BY и HAVING, когда они используются в запросе совместно с предложением GROUP BY.

Список параметров практически всех агрегатных функций, которые имеют только один аргумент, имеет синтаксис:

```
( [ { DISTINCT | UNIQUE } | ALL ] выражение )
```

Здесь:

- DISTINCT или UNIQUE указывает, что при вычислении результата агрегатной функции не учитываются дубликаты значений;
- ALL указывает, что учитываются все значения (действует по умолчанию). В СУБД Oracle более 50 агрегатных функций [13], например:
- AVG – среднее значение в наборе значений;
- CORR – коэффициент корреляции набора пар значений;
- COUNT – число не NULL-значений в наборе значений, но если в качестве аргумента используется звездочка (*), то выполняется подсчет общего числа строк в источнике данных;
- MAX – максимальное значение в наборе;
- MEDIAN – медианное значение в наборе; применимо только к данным числовых и дата-время типов;
- MIN – минимальное значение в наборе;
- STDDEV – стандартное отклонение на наборе значений;

- SUM – сумма значений набора;
- VARIANCE – дисперсия для набора.

Если в операторе SELECT не используется предложение GROUP BY, то агрегатные функции в списке выборки не могут сочетаться с именами столбцов источника данных, а также не могут вкладываться друг в друга (см. пример 17).

Пример 17. Формирование статистики по общему числу сотрудников и их максимальной, минимальной и медианной зарплате по таблице employees.

```
SELECT COUNT(*) AS "Employees number",
       MIN(salary) AS "Minimal salary",
       MAX(salary) AS "Maximal salary",
       MEDIAN(salary) AS "Median salary"
FROM HR.employees;
```

Предложение GROUP BY в операторе SELECT делит все строки источника данных на группы:

```
<группировка_данных>::=
  GROUP BY { выражение } [,...] [ HAVING условие ]
```

Число групп определяется числом уникальных комбинаций данных в столбцах группировки, которые указаны в списке выражений после GROUP BY (см. примеры 18–20). Для каждой группы в результирующей таблице формируется только одна строка, при этом в списке выборки можно использовать только выражения из списка группировки, константы, агрегатные и скалярные функции и выражения на их основе. Агрегатные функции в данном случае применяются к строкам данных каждой группы.

В предложении ORDER BY также могут использоваться агрегатные функции, причем не только те, которые входят в список выборки запроса. В фильтре WHERE применять агрегатные функции нельзя, так как выполнение группировки данных выполняется уже после фильтрации строк данных.

Пример 18. Вывод средней зарплаты по отделам из таблицы employees, отсортированный в порядке убывания числа сотрудников отделов.

```
SELECT department_id,
       ROUND(AVG(salary), 2) AS "Average salary"
FROM HR.employees
GROUP BY department_id
ORDER BY COUNT(*) DESC;
```

Пример 19. Вывод числа различных должностей по отделам из таблицы employees.

```
SELECT department_id,
       COUNT(DISTINCT job_id) AS "Number of different jobs"
FROM HR.employees
GROUP BY department_id
ORDER BY department_id;
```

Пример 20. Вывод числа различных должностей по отделам из таблицы employees.

```
SELECT department_id, job_id, COUNT(*) AS "Number of jobs"
FROM HR.employees
GROUP BY department_id, job_id
ORDER BY department_id, job_id;
```

Предложение HAVING предназначено для фильтрации групп данных – в результирующую таблицу помещаются только те финальные строки групп, для которых условие истинно. Предложения HAVING для своего построения использует те же условия, которые изложены в пункте 3.1.3. Также в условии предложения HAVING могут быть использованы агрегатные функции, которые позволяют выполнять фильтрацию по статистике конкретной группы, причем любые, а не только те, которые были использованы в списке выборки запроса (см. пример 21).

Пример 21. Вывод средней зарплаты только по тем отделам из таблицы employees, в которых работает больше 10 сотрудников, отсортированный в порядке убывания числа сотрудников отделов.

```
SELECT department_id,
       ROUND(AVG(salary), 2) AS "Average salary"
FROM HR.employees
GROUP BY department_id HAVING COUNT(*) > 10
ORDER BY COUNT(*) DESC;
```

При выполнении группировки в запросе выборки разрешено агрегатные функции вкладывать друг в друга (*nested aggregate functions*). Например, в примере 22 вычисляется максимальное значение средней заработной платы всех отделов следующим образом: сначала для каждого отдела с помощью группировки и функции AVG формируется набор средней зарплаты по каждому отделу, а затем функция MAX находит в этом наборе максимальное значение средней зарплаты.

Пример 22. Вычисление максимальной средней зарплаты отделов из таблицы employees.

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

3.1.5 Сложные источники данных (предложение FROM)

Если в запросе на выборку используется несколько источников данных, то предложение FROM применяется для формирования единой таблицы из всех этих источников с использованием операций соединения (*join*) обычно в следующем синтаксисе:

```
<соединение_данных> ::=
    <таблица_данных> { <внутреннее_соединение_данных> |
                      <внешнее_соединение_данных> } [...]
```

```

<внутреннее_соединение_данных>::=
{ [INNER] JOIN <таблица_данных> { ON условие_соединения |
                                USING (имя_столбца [, ...]) } } |
{ { CROSS | NATURAL [INNER] } JOIN <таблица_данных> }

<внешнее_соединение_данных>::=
{ FULL | LEFT | RIGHT } [OUTER] JOIN <таблица_данных>
{ ON условие_соединения | USING (имя_столбца [, ...]) }

```

Соединение данных (*join*) – операция, позволяющая формировать одну общую таблицу данных из нескольких таблиц. В основе операции соединения данных лежит операция декартова произведения (*cartesian product*) [10, 11], которая позволяет получить все возможные сочетания строк двух источников данных. Операцию декартова произведения можно выполнить, указав в предложении FROM список таблиц через запятую (см. подраздел 3.1 и пример 23), а можно использовать специальную форму CROSS JOIN (см. пример 24).

Пример 23. Декартово произведение таблиц employees и jobs на основе списка.

```

SELECT e.last_name, j.job_title
FROM HR.employees e, HR.jobs j
ORDER BY 1 ASC, 2 ASC;

```

Пример 24. Декартово произведение таблиц employees и jobs с использованием CROSS JOIN.

```

SELECT e.last_name, j.job_title
FROM HR.employees e CROSS JOIN HR.jobs j
ORDER BY 1 ASC, 2 ASC;

```

Затем для выделения только строк, содержащих верные сочетания данных, выполняется операция выборки (см. пункт 3.1.3), в которой условие соединения задается следующей формулой:

$$R.a \Theta S.b, \tag{1}$$

где R и S – таблицы; a и b – имена существующих столбцов в таблице R и S соответственно; вместо Θ должен быть указан один из операторов сравнения (=, !=, <>, <, >, <=, >=).

В реляционной базе данных соединение таблиц обычно выполняется с помощью соединения по эквивалентности (или эквисоединения (*equijoin*)) – соединения, для которого формула (1) имеет вид

$$R.a = S.b, \tag{2}$$

причем для получения верного смысла операции соединения на имеющейся структуре реляционной базы данных столбцы a и b должны описывать ссылку между таблицами, то есть один из них должен являться первичным ключом, а другой – внешним ключом, ссылающимся на этот первичный ключ (см. пункт 2.2.2). Эквисоединения между другими случайно выбранными однотипными столбцами дадут в результате бессмысленный набор данных, не совпадающий с реальностью. Эквисоединения между разнотипными столбцами приведут к ошибке выполнения оператора выборки при выполнении соединения.

Внутреннее соединение по эквивалентности позволяет получить одну таблицу из двух исходных, в которой все строки будут иметь данные, сформированные только на основе существующих ссылок (см. примеры 25–27, которые формируют одинаковый результат). Пример 25 использует старую форму синтаксиса, в которой условие соединения находится в предложении WHERE, вместе с другими возможными условиями фильтрации строк данных. Примеры 26 и 27 основаны на новой форме синтаксиса. Ключевое слово USING (см. пример 27) позволяет автоматически формировать условия соединения по эквивалентности для случая, когда имена столбцов первичного ключа и внешнего ключа, который ссылается на этот первичный ключ, имеют одинаковые имена.

Пример 25. Выборка данных о сотруднике и его должности из таблиц employees и jobs на основе списка и условия соединения по эквивалентности в фильтре WHERE.

```
SELECT e.last_name, j.job_title
FROM HR.employees e, HR.jobs j
WHERE e.job_id = j.job_id
ORDER BY 1 ASC, 2 ASC;
```

Пример 26. Выборка данных о сотруднике и его должности из таблиц employees и jobs с использованием INNER JOIN и условия соединения по эквивалентности.

```
SELECT e.last_name, j.job_title
FROM HR.employees e INNER JOIN
      HR.jobs j ON (e.job_id = j.job_id)
ORDER BY 1 ASC, 2 ASC;
```

Пример 27. Выборка данных о сотруднике и его должности из таблиц employees и jobs с использованием INNER JOIN и условия соединения по эквивалентности по столбцу job_id.

```
SELECT e.last_name, j.job_title
FROM HR.employees e INNER JOIN HR.jobs j USING (job_id)
ORDER BY 1 ASC, 2 ASC;
```

Пример 27 также можно реализовать с помощью естественного соединения (*natural join*) – эквисоединения, выполняемого по всем общим полям (полям с одинаковым названием в обеих таблицах) (см. пример 28).

Пример 28. Выборка данных о сотруднике и его должности из таблиц employees и jobs с использованием NATURAL INNER JOIN.

```
SELECT e.last_name, j.job_title
FROM HR.employees e NATURAL INNER JOIN HR.jobs j
ORDER BY 1 ASC, 2 ASC;
```

Однако на практике естественное соединение рекомендуется применять с большой осторожностью, так как при недосмотре или изменении в структурах таблиц могут возникнуть неверные ассоциации при выполнении такого соединения. Например, при выполнении естественного соединения таблиц employees и departments (см. пример 29) в результате будет получено меньше

строк, чем реально существует работающих в отделах сотрудников (см. пример 30), из-за того что в этих таблицах есть одноименный столбец `manager_id`, который не формирует ссылку между этими таблицами (см. описание ссылок схемы HR в подразделе 2.4).

Пример 29. Выборка данных о сотруднике и его отделе из таблиц `employees` и `departments` с использованием `NATURAL INNER JOIN` (пример автоматически сформированного ошибочного условия соединения).

```
SELECT e.last_name, d.department_name
FROM HR.employees e NATURAL INNER JOIN HR.departments d
ORDER BY 1 ASC, 2 ASC;
```

Пример 30. Выборка данных о сотруднике и его отделе из таблиц `employees` и `departments` с использованием `INNER JOIN` и условия соединения по эквивалентности по столбцу `department_id`.

```
SELECT e.last_name, d.department_name
FROM HR.employees e INNER JOIN
      HR.departments d USING (department_id)
ORDER BY 1 ASC, 2 ASC;
```

Внешнее соединение (*outer join*) позволяют сохранить строки, которые были бы утрачены при применении внутреннего соединения:

- левое внешнее соединение (`LEFT OUTER JOIN`) позволяет сохранить все строки таблицы левого операнда соединения – в результирующей таблице будут содержаться не только сочетания, для которых выполняется условие соединения, но и все остальные строки таблицы левого операнда, дополненные в недостающей части сочетания `NULL`-значениями для столбцов таблицы правого операнда (см. пример 31, рисунок 16);
- правое внешнее соединение (`RIGHT OUTER JOIN`) позволяет сохранить все строки таблицы правого операнда, для которых не выполняется условие соединения, аналогичным образом;
- полное внешнее соединение (`FULL OUTER JOIN`) объединяет результаты как левого, так и правого внешних соединений.

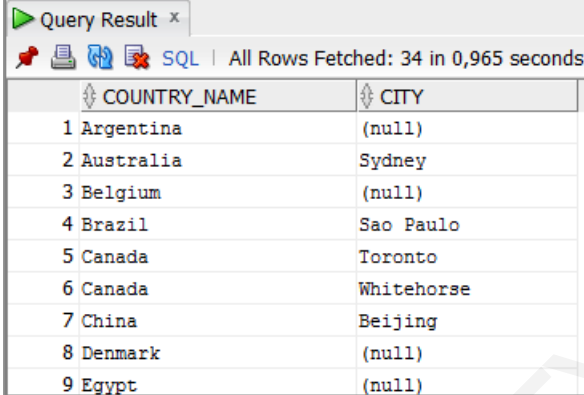
Пример 31. Выборка данных о городах и странах из таблиц `locations` и `countries` с сохранением информации о странах с использованием `LEFT OUTER JOIN`.

```
SELECT c.country_name, l.city
FROM HR.countries c LEFT OUTER JOIN
      HR.locations l USING (country_id)
ORDER BY 1 ASC, 2 ASC;
```

Внешнее соединение можно также реализовать в синтаксисе списка таблиц через запятую в предложении `FROM`, добавив в предложении `WHERE` в часть условия соединения со стороны, дополняемой `NULL`-значениями, оператор `(+)` (см. пример 32, который аналогичен по результату примеру 31), но этот синтаксис считается устаревшим и имеет ряд ограничений.

Пример 32. Выборка данных о городах и странах из таблиц *locations* и *countries* с сохранением информации о странах с использованием оператора (+).

```
SELECT c.country_name, l.city
FROM HR.countries c, HR.locations l
WHERE c.country_id = l.country_id (+)
ORDER BY 1 ASC, 2 ASC;
```



	COUNTRY_NAME	CITY
1	Argentina	(null)
2	Australia	Sydney
3	Belgium	(null)
4	Brazil	Sao Paulo
5	Canada	Toronto
6	Canada	Whitehorse
7	China	Beijing
8	Denmark	(null)
9	Egypt	(null)

Рисунок 16 – Результат выполнения примера 31

Соединения не на основе оператора равенства также могут быть реализованы, но здесь необходимо найти смысл в условии соединения (см. пример 33), поэтому такие виды соединений встречаются редко. Использование псевдонимов в примере 33 позволяет выполнить самосоединение (*self join*) таблицы *employees*.

Пример 33. Выборка комбинаций фамилий сотрудников из таблицы *employees* с использованием соединения не по эквивалентности.

```
SELECT e1.last_name AS "Employee 1",
       e2.last_name AS "Employee 2"
FROM HR.employees e1 INNER JOIN
     HR.employees e2 ON (e1.last_name <> e2.last_name)
ORDER BY 1 ASC, 2 ASC;
```

Для создания источника данных, сформированного более чем в двух таблицах, в предложении FROM выполняется последовательное соединение этих таблиц (см. пример 34).

Пример 34. Выборка данных о фамилии, должности и зарплате сотрудников отдела с названием «IT», отсортированные по убыванию зарплаты.

```
SELECT last_name, job_title, salary
FROM HR.employees INNER JOIN
     HR.departments USING (department_id) INNER JOIN
     HR.jobs USING (job_id)
WHERE department_name = 'IT'
ORDER BY salary DESC;
```

Предложение FROM выполняется первым в операторе выборки для формирования единой таблицы – источника данных, а затем эта таблица обрабаты-

вается во всех остальных предложениях оператора SELECT. В предложении FROM также могут использоваться подзапросы (см. подраздел 3.2).

3.2 Подзапросы

Оператор выборки SELECT, помещенный в круглые скобки и расположенный в некотором предложении другого оператора SQL, называется подзапросом. Подзапрос в свою очередь может содержать другой подзапрос, но подзапрос не может содержать предложение ORDER BY.

Подзапросы в Oracle по структуре возвращаемой таблицы делят на следующие виды:

- скалярный подзапрос (*scalar subquery*) – возвращает не более одного значения, то есть возвращаемая таблица всегда имеет не более одной строки и только один столбец; если такой подзапрос не возвращает ничего (то есть нуль строк), то такой результат рассматривается как NULL-значение;
- однострочный подзапрос (*single-row subquery*) – возвращает всегда только одну строку, число столбцов при этом может быть от одного и более;
- многострочный подзапрос (*multiple-row subquery*) – возвращает несколько строк данных, начиная от нуля и более, и только один столбец;
- многостолбцовый подзапрос (*multiple-column subquery*) – возвращает всегда более чем один столбец данных, число строк в нем может быть от нуля и более.

В операторе SELECT подзапрос может быть расположен с некоторыми ограничениями в следующих предложениях.

1) В предложении FROM могут использоваться все виды подзапросов, которые служат источником данных для основного запроса и называются встроенными представлениями (*inline view*). Встроенные представления служат для упрощения решения некоторой задачи в основном запросе, так как позволяют решать задачу выборки данных по частям, соединения с ними выполняются так же, как и с обычными таблицами (см. пример 35).

Пример 35. Выборка данных для всех отделов о текущем количестве сотрудников и числе архивных записей по этим отделам.

```
SELECT department_name,  
       NVL(emp_num_curr, 0) AS "Current employees number",  
       NVL(emp_num_hist, 0) AS "Number of history records"  
FROM HR.departments  
LEFT OUTER JOIN  
  (SELECT department_id, COUNT(*) AS emp_num_curr  
   FROM HR.employees  
   GROUP BY department_id) USING (department_id)  
LEFT OUTER JOIN  
  (SELECT department_id, COUNT(*) AS emp_num_hist  
   FROM HR.job_history  
   GROUP BY department_id) USING (department_id)  
ORDER BY department_name;
```

2) В предложении SELECT могут быть использованы только скалярные подзапросы, которые применяются для построения выражений (см. примеры 36 и 37). В примере 36 скалярный подзапрос возвращает одно значение за счет использования агрегатной функции. В примере 37 фильтр строк подзапроса подобран так, что подзапрос возвращает только одно значение.

Пример 36. Выборка данных о проценте заработной платы для каждого из отделов от общей суммы заработной платы всех сотрудников.

```
SELECT department_name,  
       NVL(SUM(salary), 0) AS "Salary",  
       ROUND(NVL(SUM(salary), 0) /  
             (SELECT SUM(salary) FROM HR.employees) * 100, 2)  
         AS "Percent from total salary"  
FROM HR.departments LEFT OUTER JOIN  
     HR.employees USING (department_id)  
GROUP BY department_name  
ORDER BY 3 DESC;
```

Пример 37. Выборка данных о коэффициенте заработной платы сотрудников относительно заработной платы сотрудника с номером 100.

```
SELECT last_name,  
       ROUND(salary /  
             (SELECT salary FROM HR.employees  
              WHERE employee_id = 100), 2)  
         AS "Salary rate"  
FROM HR.employees  
ORDER BY 2 DESC;
```

3) В предложениях WHERE и HAVING разрешено использовать все виды подзапросов, но их обработка зависит от используемых условий (см. пункт 3.1.3). В примерах 38 и 39 применяются скалярные подзапросы, использующие агрегатные функции, в примере 40 – однострочный подзапрос, возвращающий два столбца.

Пример 38. Выборка сотрудников, которые получают зарплату больше медианной зарплаты всех сотрудников.

```
SELECT last_name, salary  
FROM HR.employees  
WHERE salary > (SELECT MEDIAN(salary) FROM HR.employees)  
ORDER BY 2 DESC, 1 ASC;
```

Пример 39. Выборка отделов, в которых работает наибольшее количество сотрудников.

```
SELECT department_name AS "Largest department",  
       COUNT(*) AS "Employees number"  
FROM HR.departments LEFT OUTER JOIN  
     HR.employees USING (department_id)  
GROUP BY department_name  
HAVING COUNT(*) = (SELECT MAX(COUNT(*))  
                  FROM HR.employees GROUP BY department_id)  
ORDER BY 1 DESC;
```

Пример 40. Выборка всех сотрудников, которые работают в том же отделе и той же должности, что и сотрудник с номером 120.

```
SELECT employee_id, last_name, job_title
FROM HR.employees INNER JOIN HR.jobs USING (job_id)
WHERE (department_id, job_id) = (SELECT department_id, job_id
                                FROM HR.employees
                                WHERE employee_id = 120)
ORDER BY 1 ASC;
```

При использовании многострочного запроса должен использоваться либо оператор IN (см. примеры 41 и 42), либо кванторные условия (см. пример 43), описанные в пункте 3.1.3, так как обычный оператор сравнения не может быть применен к множеству значений, возвращаемых таким подзапросом.

Пример 41. Выборка всех сотрудников, которые не работают в отделе с названием «IT».

```
SELECT employee_id, last_name
FROM HR.employees
WHERE employee_id NOT IN (SELECT employee_id
                          FROM HR.departments LEFT OUTER JOIN
                          HR.employees USING (department_id)
                          WHERE department_name = 'IT')
ORDER BY 1 ASC;
```

Пример 42. Выборка сотрудников, которые приняты на работу в тот же год и с такой же зарплатой, что и один из сотрудников отдела с номером 30, но не являются сотрудниками этого отдела.

```
SELECT department_id, last_name, salary,
       TO_CHAR(hire_date, 'YYYY')
FROM HR.employees
WHERE (salary, to_char(hire_date, 'YYYY')) IN
      (SELECT salary, TO_CHAR(hire_date, 'YYYY')
       FROM HR.employees
       WHERE department_id = 30) AND
      department_id != 30
order by 1, 2;
```

Пример 43. Выборка сотрудников, у которых зарплата больше, чем у всех сотрудников отдела «IT».

```
SELECT employee_id, last_name, salary
FROM employees
WHERE salary > ALL (SELECT salary
                   FROM departments LEFT OUTER JOIN
                   employees USING (department_id)
                   WHERE department_name = 'IT')
ORDER BY 3 DESC;
```

Для обработки результата многостолбцового подзапроса в том случае, когда важен только факт наличия данных в его результате, используется оператор EXISTS (см. пример 44).

Рассмотренные подзапросы не зависят от данных основного запроса, поэтому такие подзапросы можно выполнить только один раз, а потом использовать результаты этих подзапросов как константы в основном запросе.

Существует и другой вид подзапросов – коррелированные (*correlated subquery*) или соотнесенные. Они зависят от текущей строки данных основного запроса, когда, например, в условии фильтрации подзапроса указан столбец из основного запроса (см. примеры 44 и 45). Такие коррелированные подзапросы должны заново выполняться для каждой строки данных из внешнего оператора.

Пример 44. Выборка названий отделов, которые фигурируют в записях истории назначения сотрудников на должности.

```
SELECT department_name
FROM HR.departments d
WHERE EXISTS (SELECT * FROM HR.job_history jh
              WHERE jh.department_id = d.department_id)
ORDER BY 1;
```

Пример 45. Выборка сотрудников, которые получают наибольшую зарплату, среди сотрудников такой же должности.

```
SELECT j.job_title, e.last_name, e.salary
FROM HR.employees e INNER JOIN HR.jobs j ON(e.job_id = j.job_id)
WHERE salary = (SELECT MAX(esq.salary)
                FROM HR.employees esq
                WHERE esq.job_id = e.job_id)
ORDER BY 1, 2;
```

3.3 Операторы работы с множествами (UNION, INTERSECT, MINUS)

Операции над множествами предназначены для объединения таблиц, одинаковых по своей структуре (заголовкам), в одну общую таблицу с таким же заголовком [10, 11]. Данные для операций над множествами формируются с помощью подзапросов, но при этом подзапросы необязательно заключать в круглые скобки. Таблицы, формируемые подзапросами, должны иметь одинаковые заголовки, то есть число столбцов в заголовках должно совпадать и парные столбцы должны иметь совмещенные типы данных, однако у парных столбцов необязательно должны быть одинаковые имена. Заголовок результирующей таблицы будет иметь имена столбцов по названиям в первом (левом) подзапросе. Синтаксис операций над множествами описан в диаграмме оператора SELECT как

```
<операции_над_множествами> ::=
  { подзапрос { UNION [ALL] | INTERSECT | MINUS }
    подзапрос } [...] [ <сортировка_данных> ]
```

Здесь:

- UNION – объединение исходных множеств; в результате операции будет получена таблица, включающая в себя только различные строки данных из обоих источников;
- UNION ALL – объединение исходных множеств; в результате операции

будет получена таблица, включающая в себя все строки данных из обоих источников, включая и дубликаты;

- INTERSECT – пересечение исходных множеств; в результате операции будет получена таблица, включающая в себя только те строки, которые присутствуют в обоих исходных множествах;
- MINUS – операция вычитания множеств; в результате операции будет получена таблица, включающая в себя только те строки, которые присутствуют во множестве, представленном левым операндом операции, и отсутствуют во втором, представленном правым операндом, поэтому результат операции зависит от расположения данных.

Для запросов, использующих операции над множествами, предложение ORDER BY должно указывать либо позиции, либо псевдонимы столбцов, а не их явные имена.

С помощью операций над множествами можно собирать общие списки данных (см. пример 46) или выполнять поиск различий (см. пример 47, результат выполнения которого аналогичен результату выполнения примера 41).

Пример 46. Выборка всех сотрудников, которые когда-либо работали в отделе с номером 50.

```
SELECT employee_id, last_name
FROM HR.employees
WHERE department_id = 50
UNION
SELECT employee_id, last_name
FROM HR.job_history jh INNER JOIN
     HR.employees e USING (employee_id)
WHERE jh.department_id = 50
ORDER BY 1;
```

Пример 47. Выборка всех сотрудников, которые не работают в отделе с названием «IT».

```
SELECT employee_id, last_name
FROM HR.employees
MINUS
SELECT employee_id, last_name
FROM HR.departments LEFT OUTER JOIN
     HR.employees USING (department_id)
WHERE department_name = 'IT'
ORDER BY 1;
```

Операция UNION также часто используется для совмещения в одной таблице реальных данных и статистики по этим данным, но подзапросы в этом случае должны быть построены таким образом, чтобы быть совместимыми для соединения (см. пример 48). Пример 48 построен с использованием предложения WITH, которое располагается до оператора выборки данных SELECT. В предложении WITH описываются именованные подзапросы, которые затем могут неоднократно использоваться в основном запросе:

```

<факторизация_подзапроса>::=
  WITH
    { имя_подзапроса [ ( имя_псевдонима_столбца [, ...] ) ]
      AS ( подзапрос )
      [ <поиск_данных> ] [ <заикливание_данных> ] } [, ...]

```

Пример 48. Выборка данных о зарплате сотрудников, которые работают в отделе с номером 60, и обобщающая статистика по зарплате этого отдела.

```

WITH
sq_emp_60 AS (SELECT (' ' || last_name) AS employee, salary
                FROM employees WHERE department_id = 60),
sq_avg_60 AS (SELECT 'AVERAGE SALARY' AS avg_info,
                  AVG(salary) AS avg_salary
                FROM employees WHERE department_id = 60),
sq_sum_60 AS (SELECT 'SUMMARY SALARY' AS sum_info,
                  SUM(salary) AS sum_salary
                FROM employees WHERE department_id = 60)
SELECT employee, salary FROM sq_emp_60
UNION
SELECT avg_info, avg_salary FROM sq_avg_60
UNION
SELECT sum_info, sum_salary FROM sq_sum_60
ORDER BY 1;

```

3.4 Иерархические запросы

В реляционной модели можно представить древовидную иерархию однотипных объектов как набор связей «потомок – предок» между строками одной и той же таблицы данных, задаваемых с помощью внешнего ключа. В этой связи строка, на которую указывает ссылка, будет являться предком, а строка, в которой находится ссылка, – потомком. Например, в таблице HR.employees (рисунок 17) строки связаны с помощью внешнего ключа employees.manager_id, который ссылается на первичный ключ этой же таблицы employees.employee_id, что и задает связь «подчиненный (потомок) → начальник (предок)».

В этой иерархии строка со значением employees.manager_id, равным NULL, задает корень дерева. Строка, на которую никто не ссылается, называется листом. От корня дерева до его листа через строки промежуточных уровней проходит ветвь дерева.

Для работы с такой иерархией данных в Oracle используются специальные иерархические запросы (*hierarchical query*), которые и позволяют применять указанные выше ссылки «потомок – предок».

```

<иерархия_данных>::=
  { CONNECT BY [NOCYCLE] условие [START WITH условие] } |
  { START WITH условие CONNECT BY [NOCYCLE] условие }

```

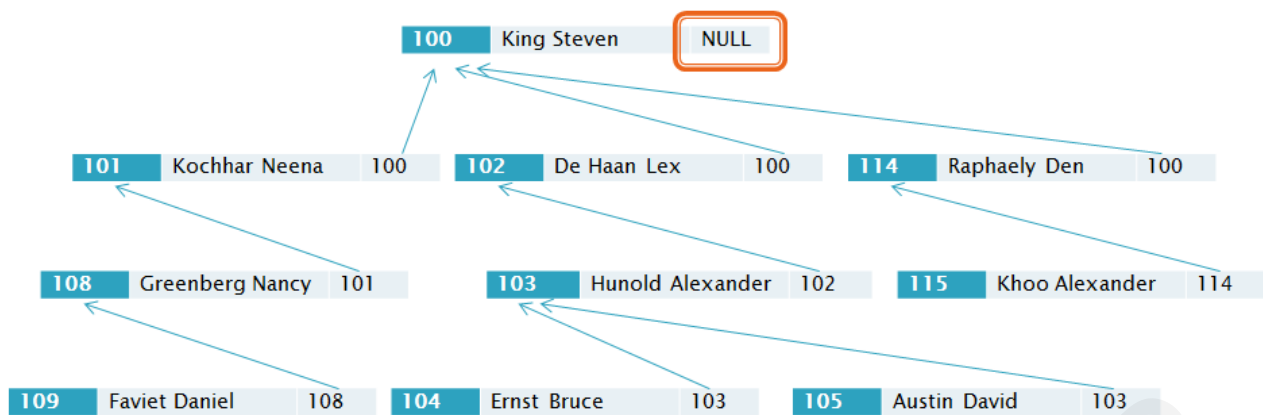


Рисунок 17 – Пример иерархии строк в таблице `HR.employees`

Здесь:

- `START WITH` – предложение, которое определяет корневую строку, начиная с которой будет выполняться построение дерева иерархии; при отсутствии этого предложения все строки будут считаться корневыми и от каждой из них будет построено свое собственное дерево;
- `CONNECT BY` – предложение, необходимое для обхода иерархической структуры; в нем задается условие соединения для столбцов, которые относятся к связи «потомок – предок» и однозначно указан столбец, относящийся к предку; условие соединения обычно содержит сравнение по равенству значений (`=`), в противном случае будут возможны многочисленные варианты связей, в том числе и приводящие к циклам;
- `NOCYCLE` – параметр, указывающий на то, что запрос будет выполняться, даже если в условии соединения возникнет цикл в иерархии данных; в противном случае возникнет ошибка выполнения запроса.

В предложении `CONNECT BY` перед тем из столбцов, который относится к предку, должен быть указан унарный оператор `PRIOR`, с помощью которого происходит формирование направления движения по дереву.

Также в иерархических запросах можно использовать:

- псевдостолбец `LEVEL` – возвращает число, указывающее уровень, на котором в дереве расположена данная строка, где 1 – уровень корневой строки дерева; столбец `LEVEL` нельзя использовать в предложениях `START WITH` и `ORDER SIBLINGS BY`, но можно использовать в предложениях `WHERE`, `GROUP BY` и `ORDER BY`;
- псевдостолбец `CONNECT_BY_ISCYCLE` – возвращает 1, если есть цикл, то есть в текущей строке есть дочерний элемент, который также является его предком, иначе возвращает 0; используется совместно с параметром `NOCYCLE`;
- псевдостолбец `CONNECT_BY_ISLEAF` – возвращает 1, если строка является листом, иначе возвращает 0;
- унарный оператор `CONNECT_BY_ROOT` – при применении его к столбцу возвращается значение этого столбца из корневой вершины;

- функцию `SYS_CONNECT_BY_PATH(выражение, разделитель)` – возвращает строку пути, которая начинается от корня и в которой каждое выражение завершается строкой разделителя.

Формирование списка иерархии происходит рекурсивно, начиная с корневой строки, в указанном порядке от предка к потомку, с использованием поиска в глубину (см. примеры 49 и 50). Для выполнения сортировки строк результата в примере 49 используется предложение `ORDER SIBLINGS BY`, которое задает специальный вид сортировки с сохранением иерархического порядка. Также в примере 49 корень дерева задан с использованием скалярного подзапроса.

Пример 49. Выборка списка сотрудников в иерархическом порядке от начальника к его подчиненному, начиная с сотрудника с фамилией King и именем Steven (см. рисунок 17).

```
SELECT employee_id,
       (last_name || ' ' || first_name) AS employee,
       SYS_CONNECT_BY_PATH((last_name || ' ' || first_name),
                           ' => ') AS path,
       manager_id,
       CONNECT_BY_ROOT last_name AS root_last_name,
       LEVEL,
       CONNECT_BY_ISLEAF
FROM HR.employees
START WITH employee_id = (SELECT employee_id FROM employees
                          WHERE last_name = 'King' AND
                                first_name = 'Steven')
CONNECT BY manager_id = PRIOR employee_id
ORDER SIBLINGS BY employee ASC;
```

Пример 50. Выборка списка сотрудников в иерархическом порядке от подчиненного к его начальнику, начиная с листа дерева, представленного сотрудником с номером 104 (см. рисунок 17).

```
SELECT employee_id, last_name || ' ' || first_name AS employee,
       manager_id, LEVEL
FROM HR.employees
START WITH employee_id = 104
CONNECT BY PRIOR manager_id = employee_id;
```

Для исключения из результата отдельных ветвей в условии предложения `CONNECT BY` следует дополнительно указать условие для исключения вершин, из которых исходят эти ветви, например с использованием `NOT IN`. Для исключения из результата только отдельных вершин необходимо использовать предложение `WHERE`.

3.5 Рекурсивные запросы

Рекурсивные запросы (*recursive subquery factoring*) предназначены для решения задачи выполнения рекурсии стандартным образом. В случае когда данные уже расположены в таблице в иерархическом порядке, обход дерева

проще выполнить с помощью предложения CONNECT BY (см. подраздел 3.4). Однако если данные изменяются по определенному закону, то проще не хранить эти данные в таблице, а формировать их по мере необходимости с помощью рекурсии в специальном запросе.

Для создания рекурсивных запросов используется предложение WITH (см. подраздел 3.3), в котором используется список псевдонимов столбцов, а также предложения <поиск_данных> и <заикливание_данных>.

Предложение <поиск_данных> позволяет указать порядок строк:

```
<поиск_данных> ::=  
SEARCH { DEPTH | BREADTH } FIRST BY  
  { имя_псевдонима_столбца [ ASC | DESC ]  
    [ NULLS FIRST | NULLS LAST ] } [, ...]  
SET имя_столбца
```

Здесь:

- SEARCH DEPTH FIRST BY – поиск в глубину, когда после строки-предка сразу выводится одна из его строк-потомков и так далее вглубь, пока не будет достигнут лист ветви, после чего производится обход с ближайшего необработанного разветвления;
- SEARCH BREADTH FIRST BY – поиск в ширину, когда сначала выводится строка-предок, а затем все ее строки-потомки, расположенные на уровень ниже, и так далее;
- SET – указание специального столбца для сортировки результатов в предложении ORDER BY основного запроса.

Предложение <заикливание_данных> позволяет указать реакцию на появление цикла в указанных столбцах:

```
<заикливание_данных> ::=  
CYCLE { имя_псевдонима_столбца } [, ...]  
  SET имя_столбца TO есть_цикл DEFAULT нет_цикла
```

Здесь:

- SET – указание специального столбца для формирования информации о наличии цикла;
- *есть_цикл* – пометка о наличие цикла;
- *нет_цикла* – пометка об отсутствии цикла.

Для создания рекурсии в именованном подзапросе должно быть однократное использование его же имени (см. пример 51). При этом рекурсивный подзапрос состоит из двух частей: нерекурсивной части (якоря) и рекурсивной части (дополнения). Якорь должен быть расположен в подзапросе первым и может состоять из одного или нескольких блоков запросов, объединенных операторами работы с множествами (см. подраздел 3.3). Рекурсивная часть должна следовать за якорем и ссылаться на имя подзапроса только один раз. Нерекурсивная часть служит для начальной инициализации результата, а рекурсивная –

для формирования строк по данным из текущего результирующего множества с учетом рекурсии. Обе части должны быть объединены с использованием оператора UNION ALL. Количество псевдонимов столбцов, следующих за именем подзапроса, и количество столбцов в списках выборки якоря и рекурсивной части запроса должны быть одинаковыми.

Пример 51. Вычисление факториала числа n с помощью рекурсивного запроса.

```
WITH factorial (n, f) AS
( SELECT 1 AS n, 1 AS f FROM dual -- инициализация результата
  UNION ALL
  SELECT (n + 1) AS n,
         f * (n + 1) AS f
  FROM factorial
  WHERE n < 10
)
SELECT n, f FROM factorial; -- основной запрос
```

Пример 52 демонстрирует использование рекурсивного запроса с поиском в глубину для обхода иерархии сотрудников, схожего по результату с примером 49. Если в примере 52 вариант поиска DEPTH заменить на BREADTH, то результаты будут выведены в порядке поиска в ширину.

Пример 52. Выборка списка сотрудников в иерархическом порядке от начальника к его подчиненному, начиная с сотрудника с фамилией King и именем Steven, с использованием рекурсивного запроса с поиском в глубину.

```
WITH select_subemp (emp_id, emp_last, man_id, sub_level) AS
( SELECT employee_id, last_name, manager_id, 1 AS sub_level
  FROM HR.employees
  WHERE last_name = 'King' AND first_name = 'Steven'
  UNION ALL
  SELECT emp.employee_id, emp.last_name,
         emp.manager_id, (sub_level + 1) AS sub_level
  FROM select_subemp rez, HR.employees emp
  WHERE rez.emp_id = emp.manager_id )
SEARCH DEPTH FIRST BY emp_last DESC SET order_column
SELECT emp_id, emp_last, man_id, sub_level, order_column
FROM select_subemp ORDER BY order_column;
```

3.6 Перекрестные запросы

Перекрестные запросы (*cross-tabulation queries*) или запросы транспонирования данных, позволяют представить данные в виде таблицы, в которой столбцы заголовка принимают значения данных из столбцов базовой таблицы. Таблица, получаемая в результате работы перекрестного запроса, лучше анализируется человеком, по сравнению с обычным реляционным представлением сгруппированных данных.

Транспонировать таблицу, таким образом, можно с использованием функции DECODE (см. пример 53) или оператора CASE (см. пример 54) сов-

местно с использованием агрегатных функций, так как при транспонировании возникают перекрестные связи между разнотипными данными, обычно описываемые количественно.

Пример 53. Выборка зарплаты по должностям сотрудников из отделов с номерами от 10 до 50 в форме транспонированных данных с использованием функции DECODE.

```
SELECT job_title,  
       SUM(DECODE(department_id, 10, salary, 0)) AS dep_10,  
       SUM(DECODE(department_id, 20, salary, 0)) AS dep_20,  
       SUM(DECODE(department_id, 30, salary, 0)) AS dep_30,  
       SUM(DECODE(department_id, 40, salary, 0)) AS dep_40,  
       SUM(DECODE(department_id, 50, salary, 0)) AS dep_50  
FROM HR.employees INNER JOIN HR.jobs USING (job_id)  
GROUP BY job_title ORDER BY 1;
```

Пример 54. Выборка зарплаты по должностям сотрудников из отделов с номерами от 10 до 50 в форме транспонированных данных с использованием оператора CASE.

```
SELECT job_title,  
       SUM(CASE department_id WHEN 10 THEN salary ELSE 0 END) AS dep_10,  
       SUM(CASE department_id WHEN 20 THEN salary ELSE 0 END) AS dep_20,  
       SUM(CASE department_id WHEN 30 THEN salary ELSE 0 END) AS dep_30,  
       SUM(CASE department_id WHEN 40 THEN salary ELSE 0 END) AS dep_40,  
       SUM(CASE department_id WHEN 50 THEN salary ELSE 0 END) AS dep_50  
FROM HR.employees INNER JOIN HR.jobs USING (job_id)  
GROUP BY job_title ORDER BY 1;
```

В СУБД Oracle для упрощения формирования перекрестных запросов введено предложение PIVOT (см. пример 55), которое дополняет предложение FROM (см. подраздел 3.1). Результат выполнения примера 55 приведен на рисунке 18.

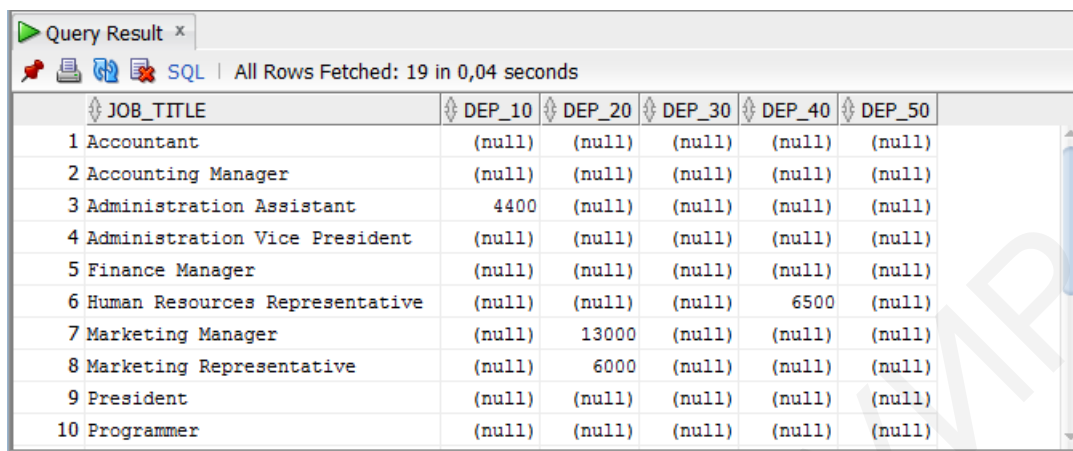
```
<прямое_транспонирование>::=  
PIVOT (  
  { агрегатная_функция (выражение) [ [AS] псевдоним ] } [, ...]  
  FOR { имя_столбца | (имя_столбца [, ...]) }  
  IN { (подзапрос) | ANY [, ...] |  
      {выражение | (выражение [, ...])} [[AS] псевдоним]} [, ...]  
  ) )
```

Пример 55. Выборка зарплаты по должностям сотрудников из отделов с номерами от 10 до 50 в форме транспонированных данных с использованием предложения PIVOT.

```
SELECT *  
FROM (SELECT job_title, department_id, salary  
       FROM HR.employees INNER JOIN HR.jobs USING (job_id))  
PIVOT (SUM(salary) FOR department_id IN (10 AS dep_10,  
    20 AS dep_20, 30 AS dep_30, 40 AS dep_40, 50 AS dep_50))  
ORDER BY job_title;
```

Для выполнения обратного транспонирования результатов перекрестного запроса в форму реляционной таблицы используется предложение UNPIVOT,

которое переводит столбцы таблицы в строки, однако полное восстановление первоначальной структуры табличных данных после преобразований PIVOT – UNPIVOT в большинстве случаев невозможно (см. пример 56).



JOB_TITLE	DEP_10	DEP_20	DEP_30	DEP_40	DEP_50
1 Accountant	(null)	(null)	(null)	(null)	(null)
2 Accounting Manager	(null)	(null)	(null)	(null)	(null)
3 Administration Assistant	4400	(null)	(null)	(null)	(null)
4 Administration Vice President	(null)	(null)	(null)	(null)	(null)
5 Finance Manager	(null)	(null)	(null)	(null)	(null)
6 Human Resources Representative	(null)	(null)	(null)	6500	(null)
7 Marketing Manager	(null)	13000	(null)	(null)	(null)
8 Marketing Representative	(null)	6000	(null)	(null)	(null)
9 President	(null)	(null)	(null)	(null)	(null)
10 Programmer	(null)	(null)	(null)	(null)	(null)

Рисунок 18 – Результат выполнения примера 55

```
<обратное_транспонирование> ::=
UNPIVOT [ { INCLUDE | EXCLUDE } NULLS ] (
  { имя_столбца | (имя_столбца [, ...]) }
  FOR { имя_столбца | (имя_столбца [, ...]) }
  IN ( { { имя_столбца | (имя_столбца [, ...]) }
       [ AS { имя_столбца | (имя_столбца [, ...]) } ] } [, ...] ) )
```

Пример 56. Обратное транспонирование таблицы, представленной подзапросом pivot_select, содержащим код из примера 55, с помощью предложения UNPIVOT.

```
WITH pivot_select AS (SELECT *
                      FROM (SELECT job_title, department_id,
                                      salary
                              FROM HR.employees INNER JOIN
                                      HR.jobs USING (job_id))
                      PIVOT (SUM(salary) FOR department_id IN
                              (10 AS dep_10, 20 AS dep_20, 30 AS dep_30,
                               40 AS dep_40, 50 AS dep_50)))
SELECT * FROM pivot_select
UNPIVOT EXCLUDE NULLS (sum_salary FOR department_id IN
                      (dep_10 AS 10, dep_20 AS 20, dep_30 AS 30,
                       dep_40 AS 40, dep_50 AS 50))
ORDER BY job_title, department_id;
```

4 МОДИФИКАЦИЯ ДАННЫХ

4.1 Операторы модификации данных

Операторы модификации данных предназначены для изменения тела объектов схемы. В данном пособии будут рассмотрены синтаксис и примеры операторов модификации данных только для работы с таблицами, но все эти операторы также могут быть применены для работы с другими объектами, например с представлениями [13]. К операторам модификации данных относятся операторы INSERT, UPDATE, DELETE и MERGE.

Оператор INSERT предназначен для вставки новых строк данных:

```
<вставка_строк> ::=
    INSERT { <вставка_строк_в_одну_таблицу> |
            <вставка_строк_в_несколько_таблиц_без_условия> |
            <вставка_строк_в_несколько_таблиц_по_условию> } ;

<вставка_строк_в_одну_таблицу> ::=
    INTO [имя_схемы.] имя_таблицы [ имя_псевдонима_таблицы ]
    [ ( имя_столбца [, ...] ) ]
    { VALUES ( { выражение | DEFAULT } [, ...] ) | подзапрос }
    [ <журналирование_ошибок> ]

<вставка_строк_в_несколько_таблиц_без_условия> ::=
    ALL { INTO [имя_схемы.] имя_таблицы [ имя_псевдонима_таблицы ]
        [ ( имя_столбца [, ...] ) ]
        VALUES ( { выражение | DEFAULT } [, ...] )
        [ <журналирование_ошибок> ] } [...] подзапрос

<вставка_строк_в_несколько_таблиц_по_условию> ::=
    [ ALL | FIRST ]
    { WHEN условие THEN
        { INTO [имя_схемы.] имя_таблицы [ имя_псевдонима_таблицы ]
          [ ( имя_столбца [, ...] ) ]
          VALUES ( { выражение | DEFAULT } [, ...] )
          [ <журналирование_ошибок> ] } [...] } [...]
    [ ELSE
        { INTO [имя_схемы.] имя_таблицы [ имя_псевдонима_таблицы ]
          [ ( имя_столбца [, ...] ) ]
          VALUES ( { выражение | DEFAULT } [, ...] )
          [ <журналирование_ошибок> ] } [...] ] подзапрос

<журналирование_ошибок> ::=
    LOG ERRORS [ INTO [имя_схемы.] имя_таблицы ] [ ( тэг_ошибки ) ]
    [ REJECT LIMIT { целое_число | UNLIMITED } ]
```

Здесь:

- INTO – предложение для указания приемника данных, в который будет производиться вставка строк;
- (имя_столбца [, ...]) – список столбцов приемника, которым будут

назначены конкретные значения при вставке (если список не указан, то будут использованы все столбцы таблицы); столбцы, которые отсутствуют в этом списке, получают значения автоматически, если это возможно;

- VALUES – это указание списка значений данных для вставки только одной строки в одну таблицу (см. пункт 2.2.3); этот список по числу параметров и их типам данных должен соответствовать заданному списку столбцов приемника; для варианта вставки в несколько таблиц в этом списке также можно указывать, какие столбцы из подзапроса будут предоставлять данные в столбцы приемника; если это предложение отсутствует, то выполняется прямое отражение столбцов подзапроса на столбцы приемника;
- DEFAULT – значение по умолчанию (см. пункт 2.2.2);
- подзапрос – служит для формирования списка строк для вставки из существующих объектов, при этом требуется соблюдение соответствия списка столбцов приемника и списка столбцов, возвращаемых подзапросом, как по числу параметров, так и по их типам;
- ALL | FIRST – указание режима условной вставки каждой строки из подзапроса в приемники: ALL – в каждый подходящий по условию, FIRST – в первый подходящий по условию;
- WHEN *условие* THEN ... – указание приемника для строки, если выполняется условие;
- ELSE – указание приемника для строки в том случае, когда не выполнено ни одно условие в предложениях WHEN;
- LOG ERRORS – предложение для журналирования ошибок, возникающих при работе операторов модификации данных, в указанной таблице;
- ТЭГ_ошибки – обычно строка с признаком ошибки;
- REJECT LIMIT – порог числа ошибок, которые можно сохранить в журнал, до отказа от выполнения оператора модификации данных.

Пример 57. В историю должностей записать данные о работе всех сотрудников отдела «IT», кроме начальника этого отдела, работающих в должности SH_CLERK в отделе Shipping в течение первой недели с момента приема их на работу.

```
INSERT INTO HR.job_history
  (employee_id, start_date, end_date, job_id, department_id)
SELECT e.employee_id, e.hire_date, e.hire_date + 6, 'SH_CLERK',
  (SELECT department_id FROM HR.departments
   WHERE department_name = 'Shipping')
FROM HR.employees e INNER JOIN HR.departments d
  ON (e.department_id = d.department_id)
WHERE d.department_name = 'IT' AND e.employee_id <> d.manager_id;
```

Вставка строк в несколько таблиц наиболее часто применяется для импортированных данных (см. подраздел 4.2) в целях их распределения в основные таблицы с иной структурой или назначением.

В СУБД Oracle поддерживается атомарность на уровне выполнения операторов: если в процессе выполнения отдельного SQL-оператора случится ошибка (*runtime error*), то все изменения, внесенные этим оператором до этой ошибки, будут исключены из базы данных. Для определения причины ошибок в операторах модификации данных служит предложение LOG ERRORS (см. пример 58). Для создания таблицы, хранящей информацию об ошибках, в этом примере использована PL/SQL-процедура CREATE_ERROR_LOG. Выборка данных по ошибке выполнения оператора INSERT из примера 58 приведена на рисунке 19. Такая обработка очень полезна для анализа характера возникающих ошибок при вставке больших объемов данных.

Пример 58. Пример журналирования ошибки выполнения оператора INSERT.

```
EXECUTE DBMS_ERRLOG.CREATE_ERROR_LOG('jobs', 'jobs_errlog');
INSERT INTO HR.jobs (job_id, job_title, min_salary, max_salary)
VALUES ('AD_PRES', 'NEW_JOB', 10000, 30000)
LOG ERRORS INTO jobs_errlog ('INSERT') REJECT LIMIT 1;
```

ORA...	ORA_ERR_MESG\$	ORA...	ORA...	ORA_ERR_TAG\$
1	ORA-00001: нарушено ограничение уникальности...	(null)	I	INSERT

Рисунок 19 – Описание ошибки выполнения вставки строки для примера 58

Оператор UPDATE предназначен для внесения изменений в уже существующие в таблице строки данных (см. пример 59), для которых условие фильтра истинно (см. подраздел 3.1.3). Если фильтр не указан, то будут изменены все строки.

```
<изменение_строк> ::=
UPDATE [имя_схемы.] имя_таблицы [имя_псевдонима_таблицы]
SET { имя_столбца = { выражение | ( подзапрос ) | DEFAULT } |
    ( имя_столбца [, ...] ) = ( подзапрос ) } [, ...]
[ <фильтр_строк_данных> ] [ <журналирование_ошибок> ] ;
```

Пример 59. Сотрудникам самого крупного отдела требуется заполнить поле e-mail в формате ABC, где A – первая буква фамилии, B – имя, C – номер отдела.

```
UPDATE HR.employees
SET email = SUBSTR(last_name, 1, 1) || first_name ||
    TO_CHAR(department_id)
WHERE department_id IN (SELECT department_id FROM HR.employees
    GROUP BY department_id HAVING COUNT(*) =
    (SELECT MAX(COUNT(*)) FROM HR.employees
    GROUP BY department_id));
```

Оператор DELETE предназначен для удаления строк данных (см. пример 60), для которых условие фильтра истинно. Если фильтр не указан, то будут удалены все строки.

```
<удаление_строк>::=
DELETE [FROM] [имя_схемы.] имя_таблицы [имя_псевдонима_таблицы]
[ <фильтр_строк_данных> ] [ <журналирование_ошибок> ] ;
```

Пример 60. Удалить должности, которые никогда не занимал ни один сотрудник.

```
DELETE FROM HR.jobs
WHERE job_id NOT IN (SELECT job_id FROM HR.employees
                    UNION
                    SELECT job_id FROM HR.job_history);
```

В операторах UPDATE и DELETE можно использовать коррелированные подзапросы (см. пример 61).

Пример 61. Всем сотрудникам, имеющим наименьшую зарплату в отделе, в котором они работают, прибавить к зарплате 100 единиц.

```
UPDATE HR.employees emp
SET emp.salary = emp.salary + 100
WHERE emp.salary = (SELECT MIN(salary) FROM HR.employees semp
                   WHERE emp.department_id = semp.department_id);
```

Оператор MERGE выполняет операции вставки, обновления и удаления строк данных в зависимости от условия существования данных и служит особой заменой комбинации операторов INSERT, UPDATE и DELETE (см. пример 62).

```
<слияние_строк>::=
MERGE INTO [имя_схемы.] имя_таблицы [имя_псевдонима_таблицы]
USING { [имя_схемы.] имя_таблицы | подзапрос }
      [имя_псевдонима_таблицы]
ON ( условие )
WHEN MATCHED THEN UPDATE SET
    { имя_столбца = { выражение | DEFAULT } } [, ...]
    [ <фильтр_строк_данных> ] [ DELETE <фильтр_строк_данных> ]
WHEN NOT MATCHED THEN INSERT ( имя_столбца [, ...] )
    VALUES ( { выражение | DEFAULT } [, ...] )
    [ <фильтр_строк_данных> ]
[ <журналирование_ошибок> ] ;
```

Здесь:

- INTO – предложение для указания приемника данных в целях вставки или обновления строк данных;
- USING – указание источника данных;
- ON – указание условия выбора варианта слияния строк, которое должно быть истинно для варианта обновления;
- WHEN MATCHED THEN UPDATE SET – предложение, предназначенное для обновления строк, для которых выполняется условие, указанное после ON, но при этом нельзя обновить столбцы, входящие в это условие; здесь дополнительно можно использовать <фильтр_строк_данных>, а

также удалить обновленные строки с помощью предложения DELETE по заданному условию;

- WHEN NOT MATCHED THEN INSERT – предложение, предназначенное для вставки строк, если не выполняется условие, указанное после ON; здесь также дополнительно можно использовать <фильтр_строк_данных>.

Пример 62. В истории должностей внести данные о работе всех сотрудников отдела «IT», кроме начальника этого отдела, работающих в должности SH_CLERK в отделе Shipping в течение первой недели с момента приема их на работу. При наличии записей в истории должностей на дату приема – не добавлять новую, а обновить имеющуюся запись.

```
MERGE INTO HR.job_history jh USING
(SELECT employee_id, hire_date
 FROM HR.employees INNER JOIN HR.departments
 USING (department_id)
 WHERE departments.department_name = 'IT' AND
 employees.employee_id != departments.manager_id) sq
ON (jh.employee_id = sq.employee_id AND
 jh.start_date = sq.hire_date)
WHEN MATCHED THEN UPDATE
 SET jh.end_date = sq.hire_date + 6,
 jh.job_id = 'SH_CLERK',
 jh.department_id = (SELECT department_id
 FROM HR.departments
 WHERE department_name = 'Shipping')
WHEN NOT MATCHED THEN
 INSERT (employee_id, start_date, end_date,
 job_id, department_id)
 VALUES (sq.employee_id, sq.hire_date, sq.hire_date + 6,
 'SH_CLERK',
 (SELECT department_id FROM HR.departments
 WHERE department_name = 'Shipping'));
```

После выполнения операторов модификации необходимо зафиксировать или откатить результаты выполнения транзакции (см. пункт 2.2.3).

4.2 Импорт данных в схему пользователя

Довольно часто в таблицы схемы необходимо вносить большое количество данных, которые поставляются извне в виде файлов различных форматов и структуры. В SQL Developer предусмотрен импорт данных из отдельных файлов в форматах Excel, Text, CSV и DSV. Далее приведены порядок импорта новых данных в схему NEW_USER (см. пункты 2.2.1, 2.2.2) из файла формата Excel (.xlsx) и примеры операторов модификации данных, выполняющих внесение этих данных в существующие таблицы схемы.

Для импорта данных необходимо подключиться к учетной записи, для нашего примера это NEW_USER. Затем требуется вызвать правой кнопкой мыши контекстное меню на пункте «Tables» и выбрать в нем пункт

«Import Data...», после чего запустится «Data Import Wizard». В этом окне предлагается сразу указать файл, для которого можно настроить формат доступа к данным и просмотреть их содержимое (см. рисунок 20).

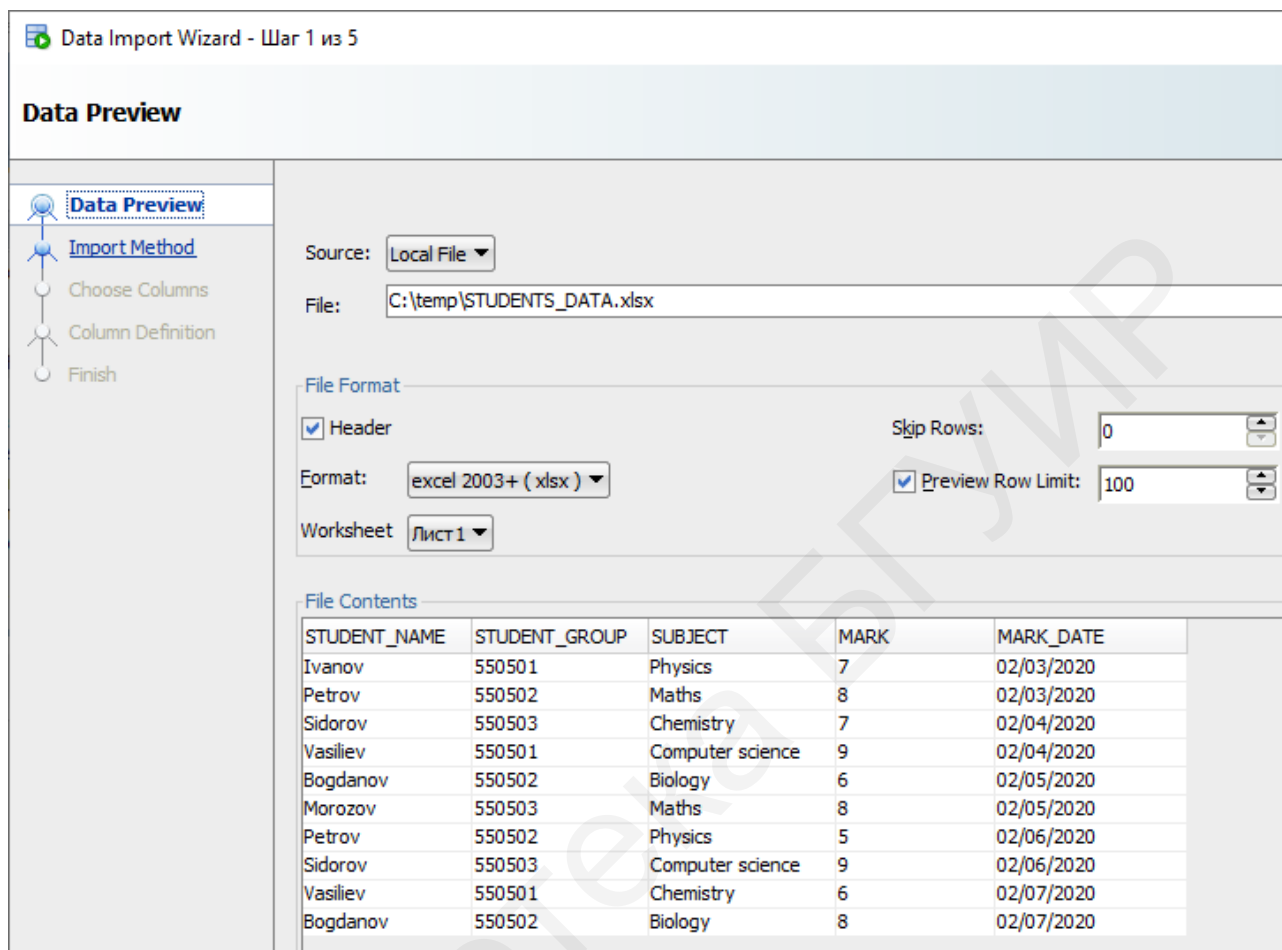


Рисунок 20 – Мастер импорта данных из файла

На следующем шаге мастера выбирается метод импорта и имя таблицы для сохранения данных, например `import_marks`. При выборе метода «Insert» мастер в результате сам создаст и заполнит данными указанную таблицу. При выборе метода «Insert Script» мастер сформирует только скрипт на языке SQL, который можно по необходимости скорректировать, дополнить командой `COMMIT` и выполнить в схеме пользователя. На последующих шагах работы мастера можно выбрать только необходимые для импорта столбцы и настроить их описание для результирующей таблицы, скорректировав их имена и типы данных. В результате импорта в схеме пользователя будет создана таблица с данными, которые можно скопировать в основные таблицы схемы и затем эту новую таблицу удалить.

Пример 63 содержит транзакцию, которая копирует импортированные данные из таблицы `import_marks` в основные таблицы схемы `NEW_USER` с учетом ссылочной целостности и с выполнением добавления ранее не созданных основных объектов (студентов, групп и предметов).

Пример 63. Копирование импортированных данных из таблицы import_marks в основные таблицы схемы NEW_USER с добавлением неизвестных объектов.

```
INSERT INTO NEW_USER.groups (group_id, name)
SELECT group_id_seq.NEXTVAL, student_group
FROM (SELECT DISTINCT student_group
      FROM import_marks)
WHERE student_group NOT IN (SELECT name FROM groups);

INSERT INTO NEW_USER.students (student_id, name, group_id)
SELECT student_id_seq.NEXTVAL, student_name, group_id
FROM (SELECT DISTINCT student_name, group_id
      FROM import_marks INNER JOIN groups
           ON (student_group = name))
WHERE (student_name, group_id) NOT IN
      (SELECT name, group_id FROM students);

INSERT INTO NEW_USER.subjects (subject_id, name)
SELECT subject_id_seq.NEXTVAL, subject
FROM (SELECT DISTINCT subject
      FROM import_marks)
WHERE subject NOT IN (SELECT name FROM subjects);

INSERT INTO NEW_USER.marks (student_id, subject_id,
                           mark, mark_date)
SELECT st.student_id, sj.subject_id, im.mark, im.mark_date
FROM import_marks im INNER JOIN
      groups gr ON (im.student_group = gr.name) INNER JOIN
      subjects sj ON (im.subject = sj.name) INNER JOIN
      students st ON (im.student_name = st.name AND
                    im.student_group = gr.name);

COMMIT;
```

Для быстрого создания таблиц и заполнения их на основе импортированных данных применяется оператор CREATE TABLE, построенный на основе подзапроса (см. пример 64). Список столбцов после названия таблицы служит для указания названий столбцов. Если такой список отсутствует, то названия столбцов таблицы будут совпадать с названиями столбцов таблицы, возвращаемой подзапросом.

Пример 64. Создание таблицы new_students в схеме NEW_USER на основе таблицы import_marks с переименованием столбцов.

```
CREATE TABLE new_students (student_name, group_name) AS
SELECT student_name, student_group
FROM import_marks
WHERE student_name NOT IN (SELECT name FROM students);
```

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Oracle Database XE [Электронный ресурс] / Oracle Documentation, 2020. – Режим доступа : <https://www.oracle.com/database/technologies/appdev/xe.html>.
2. Oracle Database Express Edition (XE) Downloads [Электронный ресурс] / Oracle Documentation, 2020. – Режим доступа : <https://www.oracle.com/database/technologies/xe-downloads.html>.
3. Oracle Database XE Prior Release Archive [Электронный ресурс] / Oracle Documentation, 2020. – Режим доступа : <https://www.oracle.com/database/technologies/xe-prior-releases.html>.
4. Installation Guide for Microsoft Windows [Электронный ресурс] / Oracle Documentation, 2020. – Режим доступа : <https://docs.oracle.com/en/database/oracle/oracle-database/18/ntdbi/index.html>.
5. SQL Developer [Электронный ресурс] / Oracle Documentation, 2020. – Режим доступа : <https://www.oracle.com/database/technologies/appdev/sql-developer.html>.
6. SQL Developer Data Modeler [Электронный ресурс] / Oracle Documentation, 2020. – Режим доступа : <https://www.oracle.com/database/technologies/appdev/datamodeler.html>.
7. Oracle VM VirtualBox [Электронный ресурс] / Oracle Documentation, 2020. – Режим доступа : <https://www.oracle.com/virtualization/virtualbox/>.
8. Pre-Built Developer VMs for Oracle VM VirtualBox [Электронный ресурс] / Oracle Documentation, 2020. – Режим доступа : <https://www.oracle.com/downloads/developer-vm/community-downloads.html>.
9. Developer Day - Hands-on Database Application Development [Электронный ресурс] / Oracle Documentation, 2020. – Режим доступа : <https://www.oracle.com/database/technologies/databaseappdev-vm.html>.
10. Коннолли, Т. Базы данных: проектирование, реализация и сопровождение. Теория и практика / Т. Коннолли, К. Бегг, А. Страчан. – Киев ; М. ; СПб. : ИД «Вильямс», 2014. – 1440 с.
11. Дейт, К. Дж. Введение в системы баз данных / К. Дж. Дейт. – Киев ; М. ; СПб. : ИД «Вильямс», 2006. – 1328 с.
12. Database Concepts [Электронный ресурс] / Oracle Documentation, 2020. – Режим доступа : <https://docs.oracle.com/en/database/oracle/oracle-database/18/cncpt/index.html>.
13. SQL Language Reference [Электронный ресурс] / Oracle Documentation, 2020. – Режим доступа : <https://docs.oracle.com/en/database/oracle/oracle-database/18/sqlrf/index.html>.
14. Database Administrator's Guide [Электронный ресурс] / Oracle Documentation, 2020. – Режим доступа : <https://docs.oracle.com/en/database/oracle/oracle-database/18/admin/index.html>.
15. Database Reference [Электронный ресурс] / Oracle Documentation, 2020. – Режим доступа : <https://docs.oracle.com/en/database/oracle/oracle-database/18/refrn/index.html>.

16. Multitenant Administrator's Guide [Электронный ресурс] / Oracle Documentation, 2020. – Режим доступа : <https://docs.oracle.com/en/database/oracle/oracle-database/18/multi/index.html>.
17. Security Guide [Электронный ресурс] / Oracle Documentation, 2020. – Режим доступа : <https://docs.oracle.com/en/database/oracle/oracle-database/18/dbseg/index.html>.
18. Дейт, К. Дж. SQL и реляционная теория. Как грамотно писать код на SQL / К. Дж. Дейт. – СПб. : Символ-Плюс, 2010. – 480 с.
19. Грабер, М. SQL для простых смертных / М. Грабер. – М. : Лори, 2014. – 383 с.
20. Casteel, J. Oracle 12c: SQL / J. Casteel. – Cendale Learning, 2015. – 604 p.
21. Куликов, С. С. Работа с MySQL, MS SQL Server и Oracle в примерах : практ. пособие / С. С. Куликов. – Минск : БОФФ, 2016. – 556 с.

Учебное издание

Калабухов Евгений Валерьевич

**РАБОТА С РЕЛЯЦИОННЫМИ БАЗАМИ ДАННЫХ
В СУБД ORACLE**

ПОСОБИЕ

Редактор *М. А. Зайцева*

Корректор *Е. Н. Батурчик*

Компьютерная правка, оригинал-макет *О. И. Толкач*

Подписано в печать 11.01.2021. Формат 60×84 1/16. Бумага офсетная. Гарнитура «Times».
Отпечатано на ризографе. Усл. печ. л. 4,3. Уч.- изд. л. 4,3. Тираж 40 экз. Заказ 177.

Издатель и полиграфическое исполнение: учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники».
Свидетельство о государственной регистрации издателя, изготовителя,
распространителя печатных изданий №1/238 от 24.03.2014,
№2/113 от 07.04.2014, №3/615 от 07.04.2014.
Ул. П. Бровки, 6, 220013, Минск