

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

МЕТОДИЧЕСКИЕ УКАЗАНИЯ
к лабораторным занятиям
по дисциплине «Базы данных»

для студентов направления подготовки
11.03.02 Инфокоммуникационные технологии и
системы связи,
Направленность (профиль)
«Интеллектуальная обработка данных в
инфокоммуникационных системах и сетях»

Ставрополь, 2026

СОДЕРЖАНИЕ

<u>ВВЕДЕНИЕ</u>	51
<u>ЛАБОРАТОРНАЯ РАБОТА №1</u>	52
<u>Управление таблицами и данными</u>	52
<u>ЛАБОРАТОРНАЯ РАБОТА №2</u>	61
<u>Более сложные манипуляции с данными</u>	61
<u>ЛАБОРАТОРНАЯ РАБОТА №3</u>	74
<u>Управление SQL*Plus</u>	75
<u>ЛАБОРАТОРНАЯ РАБОТА №4</u>	84
<u>Встроенные функции SQL</u>	84
<u>ЛАБОРАТОРНАЯ РАБОТА №5</u>	98
<u>Индексы и ограничения</u>	98
<u>ЛАБОРАТОРНАЯ РАБОТА №6</u>	109
<u>Связи между таблицами</u>	109
<u>ЛАБОРАТОРНАЯ РАБОТА №7</u>	119
<u>Написание подзапросов</u>	119
<u>ЛАБОРАТОРНАЯ РАБОТА №8</u>	124
<u>Перенос данных между таблицами</u>	124
<u>ЛАБОРАТОРНАЯ РАБОТА №9</u>	128
<u>Представления</u>	128
<u>ЛАБОРАТОРНАЯ РАБОТА №10</u>	133
<u>Последовательности</u>	133
<u>ЛАБОРАТОРНАЯ РАБОТА №11</u>	138
<u>Синонимы</u>	138
<u>ЛАБОРАТОРНАЯ РАБОТА №12</u>	141
<u>PL/SQL, хранимые процедуры, функции</u>	141
<u>ЛАБОРАТОРНАЯ РАБОТА №13</u>	162
<u>Пакеты PL/SQL и Триггеры</u>	162
<u>АППАРАТУРА И МАТЕРИАЛЫ</u>	174
<u>УКАЗАНИЯ ПО ТЕХНИКЕ БЕЗОПАСНОСТИ</u>	174
<u>СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ</u>	175

ВВЕДЕНИЕ

Цель и задачи освоения дисциплины (модуля)

Цель освоения дисциплины: формирование у студентов профессиональных компетенций, основанных на освоении современных систем управления базами данных, а также основ проектирования баз данных, удовлетворяющих требованиям профессиональных стандартов и организаций-работодателей.

Задачи освоения дисциплины:

Предоставить студентам в доступной для понимания форме основные сведения о:

- проектировании баз данных;
- моделях данных;
- операциях реляционной алгебры;
- нормализации отношений;
- обеспечении непротиворечивости и целостности данных;
- обеспечении защиты данных;
- классификации и сравнительной характеристике СУБД;
- проектировании реляционной базы данных;
- концепциях и разработке локальных и распределенных баз данных.
- принципах управления распределенной информации;
- распределенных хранилищах данных;
- перспективах управления распределенной информацией;
- распределенных СУБД и их возможностях.

Место дисциплины (модуля) в структуре образовательной программы

Дисциплина «Базы данных» относится к обязательной части дисциплин.

Перечень планируемых результатов обучения по дисциплине (модулю), соотнесённых с планируемыми результатами освоения образовательной программы

Код, формулировка компетенции	Код, формулировка индикатора	Планируемые результаты обучения по дисциплине (модулю), характеризующие этапы формирования компетенций, индикаторов
ОПК-3. Способен применять методы поиска, хранения, обработки, анализа и представления в требуемом формате информации из различных источников и баз данных, соблюдая при этом основные требования информационной безопасности	ИД-3 опк-3 решает задачи обработки данных с помощью средств вычислительной техники.	Решает задачи разработки баз данных с помощью средств вычислительной техники.

Код, формулировка компетенции	Код, формулировка индикатора	Планируемые результаты обучения по дисциплине (модулю), характеризующие этапы формирования компетенций, индикаторов
	ИД-5 ОПК-3 владеет методами и навыками обеспечения информационной безопасности.	Использует методы и навыки обеспечения информационной безопасности при реализации баз данных.
ОПК-4. Способен понимать принципы работы современных информационных технологий и использовать их для решения задач профессиональной деятельности	ИД-2 ОПК-4 проектирует решение конкретной задачи проекта, выбирая оптимальный способ ее решения, исходя из действующих правовых норм и имеющихся ресурсов и ограничений.	Применяет основы разработки и управления бизнес-процессами. Понимает и применяет методы концептуального проектирования.
	ИД-3 ОПК-4 понимает современные интерактивные программные комплексы и основные приемы обработки экспериментальных данных, в том числе с использованием стандартного программного обеспечения, пакетов программ общего и специального назначения.	Применяет методы и средства проектирования БД, структур данных, баз данных, программных интерфейсов.
	ИД-4 ОПК-4 использует возможности вычислительной техники и программного обеспечения для решения задач управления и алгоритмизации процессов обработки информации.	Осуществляет анализ предметной области, вырабатывает варианты разработки БД и проводит оценку и обоснование выработанных решений.

Код, формулировка компетенции	Код, формулировка индикатора	Планируемые результаты обучения по дисциплине (модулю), характеризующие этапы формирования компетенций, индикаторов
ОПК-5. Способен разрабатывать алгоритмы и компьютерные программы, пригодные для практического применения	ИД-1 опк-5 выбирает языки программирования и языки работы с базами данных, среды разработки информационных систем и технологий, исходя из имеющихся задач.	Осуществляет выбор СУБД, использует существующие типовые решения и шаблоны проектирования в процессе разработки БД.
	ИД-2 опк-5 читает коды программных продуктов, написанных на освоенных языках программирования, и вносит требуемые изменения.	Читает и разрабатывает программные коды на освоенных языках программирования, и вносит требуемые изменения.
	ИД-3 опк-5 разрабатывает оригинальные алгоритмы и компьютерные программы, пригодные для практического применения.	Разрабатывает модели компонентов информационных систем, включая модели баз данных, пригодные для практического применения.

PL/SQL – мощнейший процедурный язык корпорации Oracle, является основой приложений, разрабатываемых на технологиях Oracle на протяжении последних 15 лет. Изначально PL/SQL предназначался только для разработчиков.

Однако теперь он стал важнейшим инструментом администрирования баз данных, поскольку ответственность администраторов баз данных увеличилась, а границы между разработчиками и администраторами постепенно исчезают.

Данное учебное пособие позволяет приобрести навыки и умения разработки качественного программного PL/SQL-кода для

- последующей реализации основополагающей бизнес-логики на сервере Oracle с помощью хранимых PL/SQL-процедур и триггеров базы данных;
- дальнейшего формирования и обработки XML-документов внутри базы данных;
- связывания веб-страниц с базой данных Oracle;
- выполнения и автоматизации задач администрирования базы данных.

Учебное пособие объединяет теорию управления и организации баз данных и практические основы создания и управления данными системы баз данных.

В данном пособии предлагаются 13 лабораторных работ, отражающих все необходимые основы реализации и сопровождения баз данных средствами процедурного языка PL/SQL, а также приводится описание его возможностей.

Изучение PL/SQL способствует более эффективному профессиональному росту будущих специалистов в области проектирования, реализации и сопровождения баз данных. Предлагаемые контрольные вопросы помогут согласовать материалы лабораторных занятий с содержанием лекций по базам данных.

ЛАБОРАТОРНАЯ РАБОТА №1

Управление таблицами и данными

Цель и содержание

Цель работы: создание простых таблиц БД, добавление и вывод данных средствами языка PL/SQL

Теоретическое обоснование

Создание таблицы

Таблица базы данных устроена примерно так же, как и электронная таблица: она состоит из столбцов, и в нее можно помещать строки данных. Перед тем как добавлять строки, необходимо определить столбцы, которые будут составлять таблицу. Это делается с помощью команды CREATE TABLE.

Чтобы создать таблицу с помощью CREATE TABLE, нужно указать следующие данные

- имя новой таблицы (оно вводится после CREATE TABLE);
- имена и определения столбцов таблицы, разделенные запятыми.

```
CREATE TABLE имя_таблицы (имя_столбца тип_данных, ...);
```

Определение структуры таблицы

Известна команда, предназначенная для просмотра структуры таблицы. Она называется DESCRIBE (сокращенно – DESC) и имеет следующий синтаксис:

```
DESC имя_таблицы
```

Это одна из немногих команд, которые не требуется завершать точкой с запятой.

Вставка записей

Для добавления записей в таблицу используется команда INSERT.

```
INSERT INTO имя_таблицы VALUES (данные, ...);
```

Выбор записей

Для выборки всех записей из таблицы используется команда SELECT.

```
SELECT * FROM имя_таблицы;
```

Выбор определенных столбцов

С увеличением размеров таблиц иногда требуется просматривать лишь некоторые из столбцов. Для этого нужно просто перечислить требуемые столбцы в операторе SELECT, а не указывать их все с помощью символа «*».

```
SELECT имя_столбца, ... FROM имя_таблицы;
```

Переименование таблиц

Изменить имя существующей таблицы можно с помощью команды RENAME.

```
RENAME старое_имя_таблицы TO новое_имя_таблицы;
```

Изменение структуры таблицы

За время существования базы данных могут измениться бизнес-требования, которым она должна удовлетворять. Зачастую это служит причиной для изменения структуры таблиц, уже содержащихся в базе данных. К ним относятся добавление новых столбцов, изменение типа данных существующих.

Добавление столбцов

Добавлять к таблице столбцы можно в любой момент. Новые столбцы размещаются в конце табличной структуры после всех существующих столбцов. Используемый для этого синтаксис выглядит следующим образом:

```
ALTER TABLE имя_таблицы ADD имя_нового_столбца  
тип_данных;
```

Изменение типа данных столбца

Команда, изменяющая тип данных существующего столбца, имеет следующий синтаксис:

```
ALTER TABLE имя_таблицы MODIFY имя_столбца но-  
вый_тип_данных;
```

Удаление таблицы

Для удаления таблицы служит команда DROP TABLE. Используемый для этого синтаксис выглядит следующим образом:

```
DROP TABLE имя_таблицы;
```

Методика и порядок выполнения работы

Методика выполнения работы

1. Для создания таблицы вводим в строке приглашения SQL> следующий код

```
CREATE TABLE weather (  
    city                VARCHAR(80) ,  
    temp_lo             INT ,  
    temp_hi             INT ,  
    prcp                REAL) ;
```

Набрав эту команду, необходимо нажать клавишу ENTER. Тогда экран будет выглядеть так, как показано на рис. 1.1.

Таким способом SQL*Plus сообщает, что команда была успешно выполнена.



Рисунок 1.1 – Результат команды CREATE TABLE

2. Чтобы просмотреть структуру таблицы, необходимо ввести следующий код

```
DESC weather
```

Экран с результатами показан на рис. 1.2.

```

C:\oracle\app\oracle\product\10.2.0\server\BIN\sqlplus.exe
SQL>
SQL>
SQL>
SQL>
SQL>
SQL>
SQL> create table weather (
2   city varchar(80),
3   temp_lo int,
4   temp_hi int,
5   prcp real
6 );
Table created.
SQL> desc weather
Name                                     Null?      Type
-----
CITY                                     VARCHAR2(80)
TEMP_LO                                 NUMBER(38)
TEMP_HI                                 NUMBER(38)
PRCP                                     FLOAT(63)
SQL>

```

Рисунок 1.2 – Результат выполнения команды DESC

3. Для добавления несколько записей в таблицу введем код

```

INSERT INTO weather VALUES ('Stavropol', 20, 18, 50);
INSERT INTO weather VALUES ('Moscow', 20);
INSERT INTO weather VALUES ('Samara');
INSERT INTO weather VALUES ('S.peterburg');
INSERT INTO weather VALUES ('Krasnodar');

```

Набрав данные команды вы должны увидеть сообщение, показанное на рис. 1.3.

```

D:\OracleXE\app\oracle\product\10.2.0\server\BIN\sqlplus.exe
SQL> insert into weather(city,temp_lo,temp_hi,prcp) values ('Stavropol',20,18,50);
1 row created.
SQL> insert into weather(city,temp_lo) values ('Moscow',20);
1 row created.
SQL> insert into weather(city) values ('Samara');
1 row created.
SQL> insert into weather(city) values ('S.peterburg');
1 row created.
SQL> insert into weather(city) values ('Krasnodar');
1 row created.
SQL>
SQL>
SQL>

```

исунок 1.3 – Результат команды INSERT

4. Для вывода записей таблицы на экран добавим команду

```
SELECT * FROM weather;
```

В результате получим все введенные ранее записи, как показано на рис. 1.4.

CITY	TEMP_LO	TEMP_HI	PRCP
Stavropol	20	18	50
Moscow	20		
Samara			

CITY	TEMP_LO	TEMP_HI	PRCP
S.peterburg			
Krasnodar			

Рисунок 1.4 – Результат команды SELECT

5. Для просмотра данных столбца city введем следующую команду, выбирающую только первый столбец из ранее созданной таблицы

```
SELECT city FROM weather;
```

В результате вы увидите экран, показанный на рис. 1.5.

6. Переименуем только что созданную таблицу, введя следующую команду

```
RENAME weather TO wert;
```

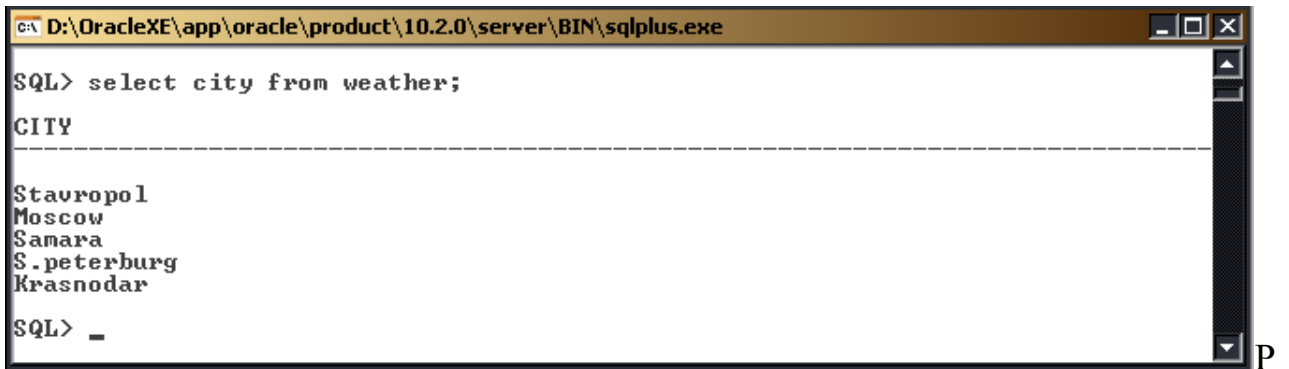


Рисунок 1.5 – Выбор заданных столбцов

7. Добавим еще один столбец в таблице wert и выведем ее структуру. Полученные результаты показаны на рис. 1.6.

```
ALTER TABLE wert ADD data VARCHAR(8);
```

```
DESC wert
```

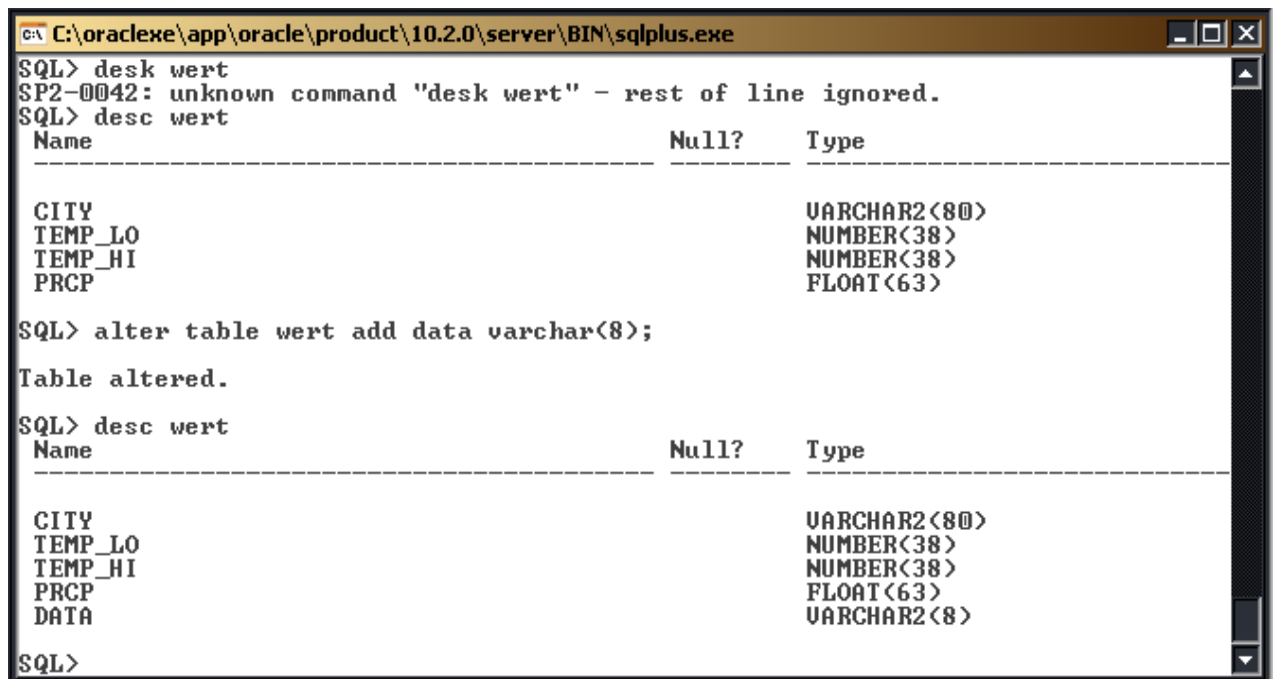
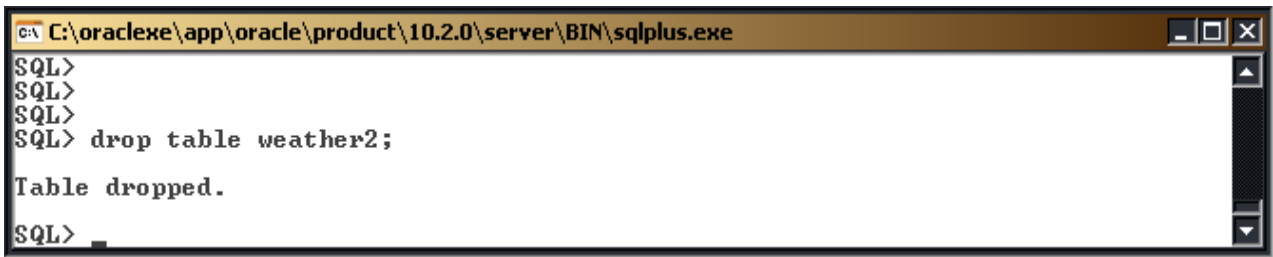


Рисунок 1.6 – Добавление столбца к существующей таблице

8. Теперь удалим таблицу wert, введя следующую команду

```
DROP TABLE wert;
```

На экране должен появиться ответ, показанный на рис. 1.7.



```

C:\oracle\app\oracle\product\10.2.0\server\BIN\sqlplus.exe
SQL>
SQL>
SQL>
SQL> drop table weather2;
Table dropped.
SQL>

```

Рисунок 1.7 – Результат выполнения команды DROP TABLE

9. Создадим еще одну таблицу

```

CREATE TABLE hase (
    product_name VARCHAR2(25),
    product_price  NUMBER(4,2),
    sales NUMBER(4,2));

```

10. Добавим данные в таблицу hase

```

INSERT INTO hase VALUES ('ProductName 1', 1, .08);
INSERT INTO hase VALUES ('ProductName 2', 2.5, .21);
INSERT INTO hase VALUES ('ProductName 3', 50.75, 4.19);
INSERT INTO hase VALUES ('ProductName 4', 99.99, 8.25);

```

11. Выполним выборку с применением математических операций.

```

SELECT product_name, product_price + sales FROM hase;
SELECT product_name, 100 - product_price FROM hase;
SELECT product_name, sales / product_price from hase;

```

При этом следует учесть, что знаменатель не может равняться нулю.

Результаты трех команд SELECT должны выглядеть примерно так, как показано на рис. 1.8.

```

C:\oracle\app\oracle\product\10.2.0\server\BIN\sqlplus.exe
SQL> select product_name, product_price+sales from hase;
PRODUCT_NAME          PRODUCT_PRICE+SALES
-----
ProductName 1          1,08
ProductName 2          2,71
ProductName 3          54,94
ProductName 4          108,24

SQL> select product_name, 100-product_price from hase;
PRODUCT_NAME          100-PRODUCT_PRICE
-----
ProductName 1          99
ProductName 2          97,5
ProductName 3          49,25
ProductName 4          ,01

SQL> select product_name, sales/product_price from hase;
PRODUCT_NAME          SALES/PRODUCT_PRICE
-----
ProductName 1          ,08
ProductName 2          ,084
ProductName 3          ,082561576
ProductName 4          ,082508251

SQL>

```

Рисунок 1.8 – Команда SELECT с различными математическими операциями

12. Узнаем, каковы будут цены из таблицы hase после их увеличения на 15%. Введите следующую команду

```
SELECT product_name, product_price * 1.15 FROM hase;
```

Результаты ее выполнения показаны на рис. 1.9.

```

C:\oracle\app\oracle\product\10.2.0\server\BIN\sqlplus.exe
SQL> select product_name, product_price*1.15 from hase;
PRODUCT_NAME          PRODUCT_PRICE*1.15
-----
ProductName 1          1,15
ProductName 2          2,875
ProductName 3          58,3625
ProductName 4          114,9885

SQL>

```

Рисунок 1.9 – Математические операции: увеличение значения на 15%

Порядок выполнения работы

1. Создайте любую таблицу, содержащую три столбца.
2. Выведите ее структуру на экран.
3. Добавьте не меньше пяти записей в таблицу.
4. Выведите все записи на экран.

5. Выведите значения одного из столбцов на экран.
6. Переименуйте таблицу.
7. Добавьте новый столбец и выведите структуру таблицы на экран.
8. Выполните выборку с применением математических операций.
9. Удалите таблицу.

Содержание отчета и его форма

Отчет должен содержать

- название лабораторной работы;
- цель работы;
- порядок выполнения работы;
- результаты выполнения;
- вывод о проделанной работе.

Контрольные вопросы и защита работы

Контрольные вопросы

1. Какая команда служит для создания таблицы?
2. Какая команда показывает структуру таблицы?
3. Какой командой добавляются записи в таблицу?
4. Какая команда служит для переименования таблицы?
5. Какой командой добавляется новый столбец в таблицу?
6. Какую позицию в таблице занимает столбец после добавления?
7. Какой командой осуществляю выборку данных из таблицы?
8. Что в «SELECT product_name, product_price * 1.15 FROM hase;» является выражением?
9. Какая команда удаляет таблицу из БД?

Защита работы

Защита работы заключается в

- выполнении заданий раздела «Методика и порядок выполнения работы»;
- ответах на контрольные вопросы;
- предоставлении отчета.

ЛАБОРАТОРНАЯ РАБОТА №2

Более сложные манипуляции с данными

Цель и содержание

Цель работы: приобрести навыки использования шаблонов для вывода данных, сортировки, модификации, удалению данных, и использованию транзакций.

Теоретическое обоснование

Ограничение диапазона выбираемых записей

Одной из наиболее распространенных операций, которые выполняют при выборе записей из таблицы, является получение определенного подмножества записей. Это подмножество может меняться в зависимости от поставленных вопросов (например: «Какие клиенты не получали от меня сообщений более двух недель?» или «Какие товары продавались партиями, содержащими более чем 100 штук за последние 30 дней?»). Подобная фильтрация записей выполняется путем добавления к оператору SELECT специальной конструкции WHERE. В ней указываются условия (conditions), которым должны удовлетворять отображаемые записи. Синтаксис выглядит следующим образом

```
SELECT столбцы FROM имя_таблицы WHERE условие (я) ;
```

Фильтрация записей по числовым значениям

Существует несколько способов фильтрации записей на основе значений в числовых столбцах. Можно выводить записи с определенным значением в столбце, записи со значениями, большими (или меньшими) заданной величины, лежащими в некотором диапазоне, а также записи, участвующие в арифметическом выражении и в условии неравенства.

Выбор записей по диапазону значений

Чтобы определить диапазон, нужно просто задать его нижний и верхний пределы. Для этого вам придется освоить новую технику, а именно: научиться указывать, что для прохождения записи через фильтр должны быть одновре-

менно выполнены два разных условия, для этого достаточно соединить два условия словом «AND». Использование AND между двумя условиями – это классический способ определения диапазона допустимых значений. Нам предлагается альтернативный способ получения того же результата, менее традиционный, но более наглядный: конструкцию BETWEEN.

Исключение записей

Если требуется *исключить* записи со значениями из определенного диапазона можно просто поменять местами знаки «больше, чем» и «меньше, чем» и поставить OR вместо AND для соединения двух условий, соблюдая приоритет логических операций. Для диапазона значений можно использовать и конструкцию BETWEEN, предварив ее модификатором «NOT».

Использование шаблонов

При поиске в текстовых столбцах часто возникает потребность отыскать небольшой фрагмент текста в любом месте столбца – например, для поиска всех записей с названиями товаров, содержащими слово «Chrome» (такими, как «Chrome full»). Это делается с помощью *шаблонов* (wildcards), представляющих произвольную часть текста.

После слова «Chrome» должен стоять знак процента (%). Это и есть шаблон, который, будучи указан в конце текстовой строки, означает: «за данной строкой может следовать что угодно, это все равно будет рассматриваться как совпадение». Шаблон % представляет любое количество текста – иначе говоря, одним символом % можно заменить сколько угодно других символов. Существует также шаблон, заменяющий только один символ. Этим шаблоном является знак подчеркивания (_).

Выбор записей по null-значениям

Некоторые записи содержат null-значения в столбцах, перечисленных в конструкции WHERE. Null-значение не удовлетворяет никакому критерию, за исключением того, который специально предназначен для проверки на null. Чтобы выполнить проверку на null, следует поместить в конструкцию WHERE параметр IS NULL.

Параметры IS NULL и IS NOT NULL применимы также в числовых и текстовых выражениях. Фактически, они работают со столбцом любого типа.

Изменение порядка записей

Чаще всего записи помещаются в таблицу далеко не в том порядке, в котором хотелось бы их просматривать. Например, данные о покупках вставляются в хронологическом порядке, но при последующем просмотре может потребоваться их упорядочение по товару или магазину. Данные о служащих компании вводятся в порядке приема этих людей на работу, но при просмотре списка вам наверняка захочется, чтобы он был отсортирован по имени и/или по отделу. Для получения таких результатов необходимо знать, как управлять порядком отображения выбранных записей.

Интересно отметить, что при этом не потребуется изменять фактическое расположение записей в таблице. Вместо этого можно просто изменять порядок отображения записей. При выполнении такого запроса выводится отсортированные записи на экран. Это позволяет просматривать записи в любом нужном порядке, без необходимости постоянно заменять таблицы их заново отсортированными версиями.

Сортировка по отдельным столбцам

Для изменения последовательности вывода записей достаточно добавить к команде SELECT конструкцию ORDER BY. В ней указывается один или несколько столбцов, по которым будет сортировать записи. Синтаксис конструкции ORDER BY следующий:

```
SELECT * FROM имя_таблицы ORDER BY столбец_сортировки;
```

Сортировка по нескольким столбцам

Для управления последовательностью вывода сортируемых столбцов можно указать в конструкции ORDER BY команды SELECT второй столбец сортировки.

```
SELECT * FROM product_n ORDER BY hand, product_price;
```

Обратите внимание, для определения двух столбцов сортировки следует просто разделить их имена запятой.

Отображение только уникальных значений

Бывают ситуации, когда необходимо знать, какие значения содержит столбец, но при этом желательно видеть только по одному экземпляру каждого значения. Например, если вас интересует, какие продавцы выполняли продажи в течение определенного периода времени, нет никакого смысла повторно выводить идентификатор продавца, по одному разу для каждой продажи. Достаточно посмотреть, кто входит в эту группу, а кто – нет. Следовательно, необходимо вывести по **ОДНОЙ** строке для каждого продавца, чье имя встречается как минимум в одной из записей о продажах в пределах указанного интервала времени. На этот случай предусмотрен модификатор **DISTINCT**.

Модификация данных в таблице

Для изменения данных в таблице используется команда **UPDATE**, ее синтаксис выглядит следующим образом

```
UPDATE имя_таблицы SET имя_столбца = новое_значение
WHERE условие;
```

Удаление записей из таблицы

Для удаления данных из таблицы используется команда **DELETE** которая имеет следующий синтаксис

```
DELETE FROM имя_таблицы WHERE условие;
```

Удаляя записи, можно указывать в условном выражении любое значение столбца (или столбцов).

Удаление всех строк

Как последний вариант можно удалить все строки таблицы. Для этого применяются два способа: использовать команду **DELETE**, не указывая условие **WHERE**, или использовать команду **TRUNCATE** (Усечь).

Команда, удаляющая все записи таблицы, имеет следующий синтаксис

```
DELETE FROM имя_таблицы;
```

Эта команда легко воспринимается при чтении кода, но у нее есть существенный недостаток: хотя она указывает на необходимость удаления всех за-

писей без исключения, СУБД все равно будет сначала считывать каждую запись, как и при наличии условия WHERE. Это может оказаться очень трудоемкой процедурой, напрасно отнимающей время и ресурсы сервера. Поэтому, если нужно удалить из таблицы все записи, более эффективным методом будет использование команды TRUNCATE.

Основное достоинство команды TRUNCATE – скорость. Когда СУБД выполняет эту команду, он не просматривает существующие записи, а просто выбрасывает их. Другой выигрыш от использования этой команды состоит в том, что она автоматически освобождает табличное пространство, которое занимали усеченные записи.

Команда TRUNCATE имеет следующий синтаксис

```
TRUNCATE TABLE имя_таблицы;
```

Управление транзакциями

Отмена транзакций

Когда вставляются, обновляются или удаляются данные, СУБД не выполняет эти изменения немедленно. Однако на самом деле изменения хранятся во временной области памяти и будут применены к таблице только в ответ на одну из нескольких специальных команд.

Отмена в Oracle выполняется с помощью команды ROLLBACK (Откат). Производя откат одной и более транзакций, вы сообщаете СУБД, что они не должны применяться к базе данных.

Возможности команды ROLLBACK не ограничиваются только одним уровнем отмены. В сочетании с другой командой, SAVEPOINT, она позволяет возвращаться в любую из предварительно установленных точек. Команда SAVEPOINT имеет следующий синтаксис:

```
SAVEPOINT имя_точки_сохранения;
```

При выполнении большого пакета команд можно устанавливать точки сохранения после каждой логической группы команд, и если следующая группа

по какой-то причине даст неудовлетворительный результат, то всегда можно вернуться к последней точке и применить лишь те изменения, которые были выполнены до нее.

Для имен точек сохранения используются соглашения, похожие на соглашения для имен таблиц и столбцов: максимальная длина – 30 символов, а первым символом должна быть буква.

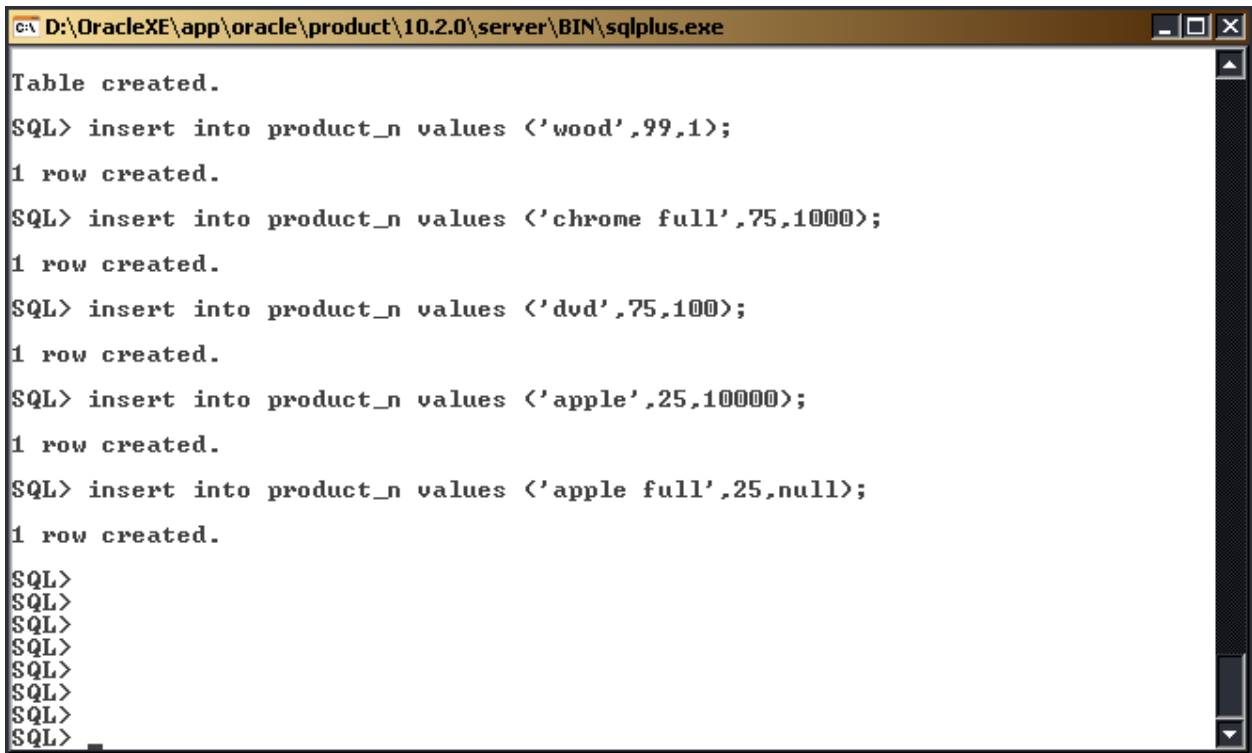
Команда COMMIT (Завершить). Приводит к записи всех ваших изменений в таблицу базы данных (что делает невозможным откат), все точки сохранения, установленные к моменту завершения транзакции, сбрасываются. После завершения изменений в таблице все точки сохранения будут сброшены. Явный ввод команды COMMIT – это лишь один из способов завершения изменений в базе данных. Некоторые команды выполняют завершение предыдущих команд, не дожидаясь ваших указаний, т.е. неявно. Говоря конкретнее, любые команды DDL (например, CREATE TABLE или DROP TABLE) неявно завершают изменение всех не сохраненных данных перед выполнением своих функций. Выход из СУБД (или просто закрытие SQL*Plus) также приводит к автоматическому завершению ваших изменений.

Методика и порядок выполнения работы

Методика выполнения работы

1. Создадим таблицу product_n и поместим в нее записи:

```
CREATE TABLE product_n (
    product_name VARCHAR2(25),
    product_price NUMBER(4,2),
    hand NUMBER(5,0));
INSERT INTO product_n VALUES ('Wood', 99, 1);
INSERT INTO product_n VALUES ('Chrome full', 75, 1000);
INSERT INTO product_n VALUES ('DVD', 50, 100);
INSERT INTO product_n VALUES ('Apple', 25, 10000);
INSERT INTO product_n VALUES ('Apple full', 25, NULL);
```



```

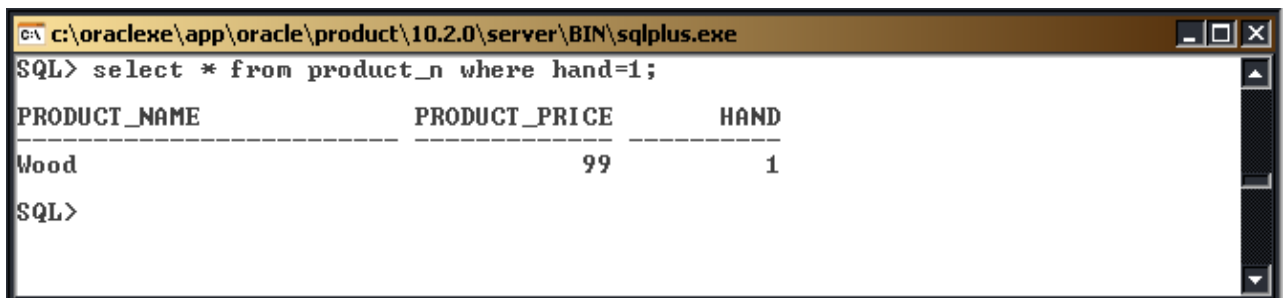
C:\D:\OracleXE\app\oracle\product\10.2.0\server\BIN\sqlplus.exe
Table created.
SQL> insert into product_n values (<'wood',99,1>);
1 row created.
SQL> insert into product_n values (<'chrome full',75,1000>);
1 row created.
SQL> insert into product_n values (<'dvd',75,100>);
1 row created.
SQL> insert into product_n values (<'apple',25,100000>);
1 row created.
SQL> insert into product_n values (<'apple full',25,null>);
1 row created.
SQL>
SQL>
SQL>
SQL>
SQL>
SQL>
SQL>
SQL>
SQL>

```

Рисунок 2.1 – Добавление в таблицу product_n записей

2. Выберем записи по одиночному значению (результат выборки на рис. 2.2)

```
SELECT * FROM product_n WHERE hand = 1;
```



```

C:\c:\oraclexe\app\oracle\product\10.2.0\server\BIN\sqlplus.exe
SQL> select * from product_n where hand=1;

```

PRODUCT_NAME	PRODUCT_PRICE	HAND
Wood	99	1

```

SQL>

```

Рисунок 2.2 – Результат выполнения команды SELECT

3. Следующий шаг будет состоять в выборе записей со значениями, которые больше или меньше определенной величины (рис. 2.3).

```
SELECT * FROM product_n WHERE hand < 500;
```

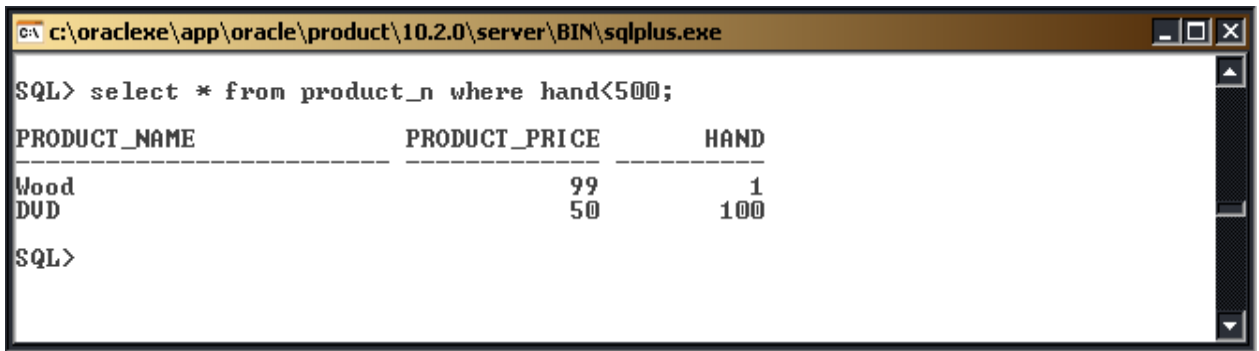


Рисунок 2.3 – Выбор записи со значениями, совпадающими с определенным числом

4. Используем оператор AND для вывода диапазона значений.

```
SELECT * FROM product_n WHERE
product_price >= 50 AND product_price <= 100;
```

5. Используем оператор BETWEEN для вывода диапазона значений.

```
SELECT * FROM product_n WHERE
product_price BETWEEN 50 AND 100;
```

6. Исключим вывод записей из диапазона значений с помощью оператора OR и AND

```
SELECT * FROM product_n WHERE
product_price < 50 OR product_price > 100;
```

7. Исключим вывод записей из диапазона значений с помощью оператора NOT и BETWEEN

```
SELECT * FROM product_n WHERE product_price NOT BETWEEN
50 AND 100;
```

8. Для исключения только одного значения используем в условии оператор «<>».

```
SELECT * FROM product_n WHERE product_price <> 50;
```

9. Выберем записей по группе допустимых значений с использованием оператора IN

```
SELECT * FROM product_n WHERE product_name IN ('DVD',
'Apple');
```

10. Найдем все записи таблицы product_n, в которых название товара

начинается с «Chrome» (рис. 2.4).

```
SELECT * FROM product_n WHERE product_name LIKE
        'Chrome%';
```



Рисунок 2.4 – Использование шаблона для поиска текста

11. Найдём все записи, в которых название товара содержит букву «W», за которой идет любой 1 символ.

```
SELECT * FROM product_n WHERE product_name LIKE 'W_';
```

12. Найдём все записи содержащие NULL значения.

```
SELECT * FROM product_n WHERE hand IS NULL;
```

13. Для поиска записей, содержащих данные в определенных столбцах, воспользуемся оператором IS NOT NULL:

```
SELECT * FROM product_n WHERE hand IS NOT NULL;
```

14. Отсортируем все данные в таблице `product_n` по их стоимости (рис. 2.5).

```
SELECT * FROM product_n ORDER BY product_price;
```

The screenshot shows a SQL*Plus window with the title bar 'C:\oracle\app\oracle\product\10.2.0\server\BIN\sqlplus.exe'. The command prompt shows the query: `SQL>`. The result is displayed as a table with three columns: `PRODUCT_NAME`, `PRODUCT_PRICE`, and `HAND`. The data rows are sorted by price in ascending order.

PRODUCT_NAME	PRODUCT_PRICE	HAND
Apple	25	10000
Apple full	25	
DVD	50	100
Chrome full	75	1000
Wood	99	1

Рисунок 2.5 – Сортировка записей по заданному столбцу

15. Создадим еще таблицу и заполним ее записями, чтобы потом извлечь из нее уникальные значения.

```
CREATE TABLE hase2 (
```

```

product_name VARCHAR2(25),
ty NUMBER(4,2),
person VARCHAR2(3));
INSERT INTO hase2 VALUES ('Wood', 1, 'CA');
INSERT INTO hase2 VALUES ('Char', 75, 'BB');
INSERT INTO hase2 VALUES ('Apple', 2, 'GA');
INSERT INTO hase2 VALUES ('Spool', 8, 'GA');
INSERT INTO hase2 VALUES ('Res', 20, 'LB');
INSERT INTO hase2 VALUES ('Wood', 11, 'BB');
INSERT INTO hase2 VALUES ('Res', 30, 'CA');

```

16. Найдем уникальные значения с помощью модификатора DISTINCT (рис. 2.6).

```

SELECT product_name FROM hase2 ORDER BY product_name;
SELECT DISTINCT product_name FROM hase2 ORDER BY product_name;

```

17. Переименуем в таблице hase2 все товары «Char» в «Widgets».

```

UPDATE hase2 SET product_name = 'Widgets'
WHERE product_name = 'Char';

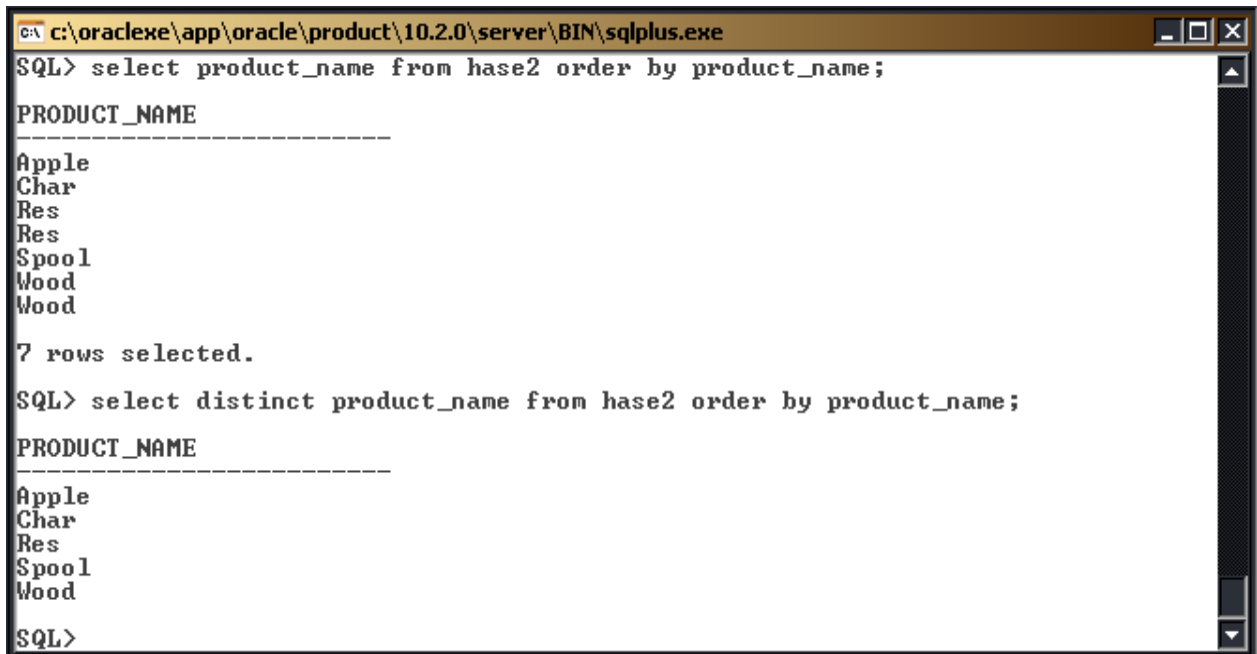
```

18. Удалим все записи, где упоминается «Wood».

```

DELETE FROM hase2 WHERE product_name = 'Wood';

```



```

c:\oracle\app\oracle\product\10.2.0\server\BIN\sqlplus.exe
SQL> select product_name from hase2 order by product_name;
PRODUCT_NAME
-----
Apple
Char
Res
Res
Spool
Wood
Wood
7 rows selected.
SQL> select distinct product_name from hase2 order by product_name;
PRODUCT_NAME
-----
Apple
Char
Res
Spool
Wood
SQL>

```

Рисунок 2.6 – Выбор уникальных значений таблицы

19. Добавим записи в таблицу hase2 и отменим добавление с помощью команды ROLLBACK (рис. 2.7).

```

INSERT INTO hase2 VALUES ('Wood', 1, 'CA');
INSERT INTO hase2 VALUES ('Char', 75, 'BB');
INSERT INTO hase2 VALUES ('Apple', 2, 'GA');
INSERT INTO hase2 VALUES ('Spool', 8, 'GA');
INSERT INTO hase2 VALUES ('Res', 20, 'LB');
INSERT INTO hase2 VALUES ('Wood', 11, 'BB');
INSERT INTO hase2 VALUES ('Res', 30, 'CA');
ROLLBACK; SELECT * FROM hase2;

```

20. Создадим точку сохранения d и произведем откат всех операций до этой точки (рис. 2.7).

```

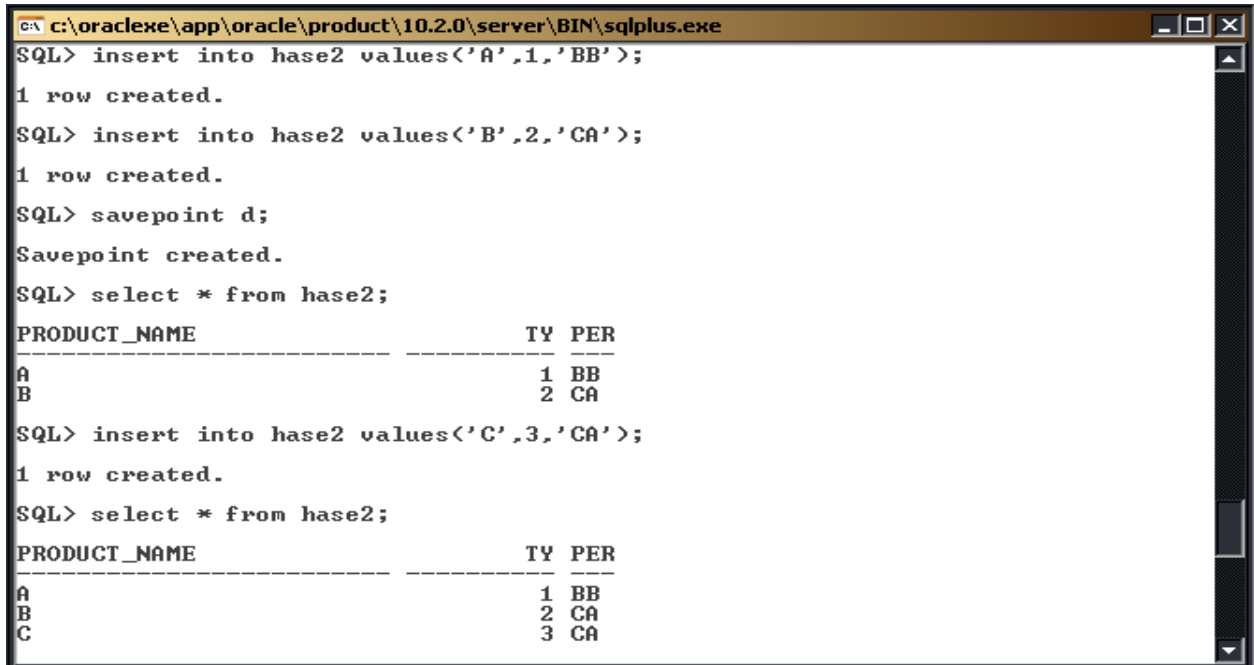
DELETE FROM hase2;
INSERT INTO hase2 VALUES ('A', 1, 'BB');
INSERT INTO hase2 VALUES ('B', 2, 'CA');
SAVEPOINT d;
SELECT * FROM hase2;
INSERT INTO hase2 VALUES ('C', 3, 'CA');
SELECT * FROM hase2; ROLLBACK TO d;

```

```

SELECT * FROM hase2;
INSERT INTO hase2 VALUES ('C', 3, 'CA');
COMMIT;
ROLLBACK TO d;
SELECT * FROM hase2;

```



SQL> insert into hase2 values('A',1,'BB');

1 row created.

SQL> insert into hase2 values('B',2,'CA');

1 row created.

SQL> savepoint d;

Savepoint created.

SQL> select * from hase2;

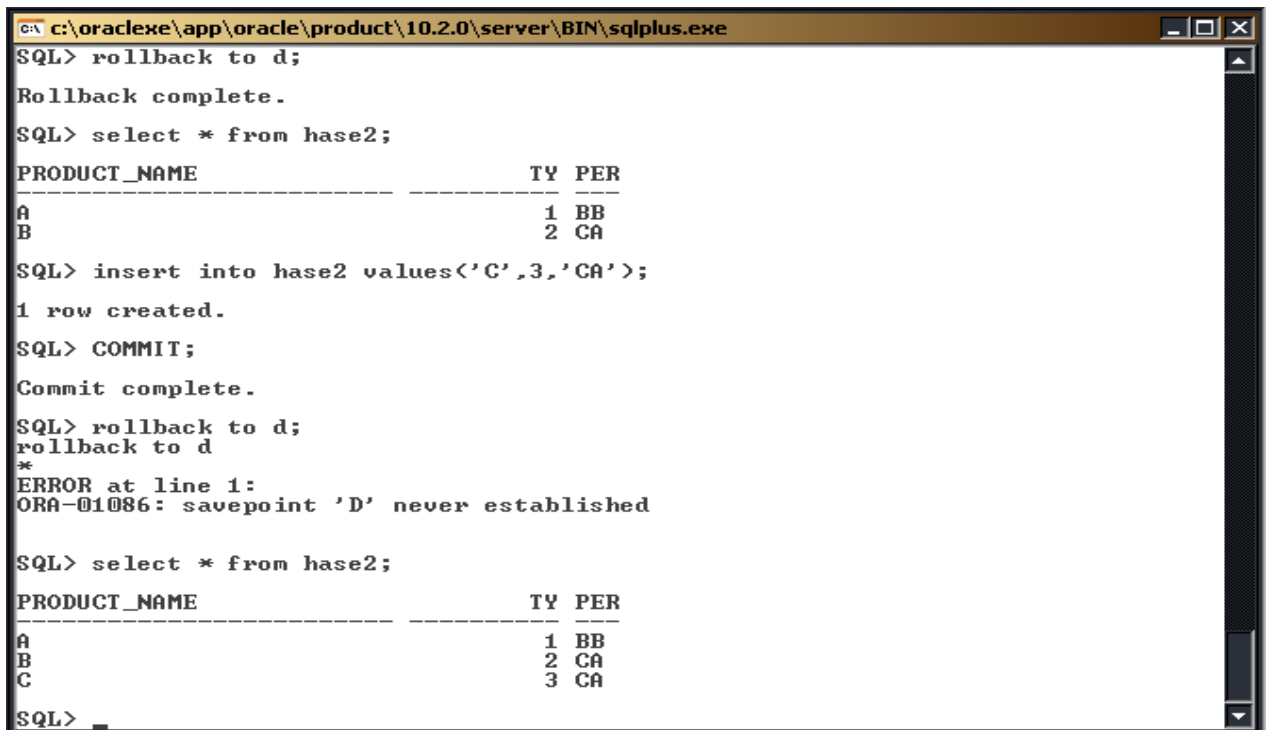
PRODUCT_NAME	TY	PER
A	1	BB
B	2	CA

SQL> insert into hase2 values('C',3,'CA');

1 row created.

SQL> select * from hase2;

PRODUCT_NAME	TY	PER
A	1	BB
B	2	CA
C	3	CA



SQL> rollback to d;

Rollback complete.

SQL> select * from hase2;

PRODUCT_NAME	TY	PER
A	1	BB
B	2	CA

SQL> insert into hase2 values('C',3,'CA');

1 row created.

SQL> COMMIT;

Commit complete.

SQL> rollback to d;

rollback to d

ERROR at line 1:

ORA-01086: savepoint 'D' never established

SQL> select * from hase2;

PRODUCT_NAME	TY	PER
A	1	BB
B	2	CA
C	3	CA

SQL>

Рисунок 2.7 – Результат выполнения команд SAVEPOINT, ROLLBACK, COMMIT

Порядок выполнения работы

1. Создайте любую таблицу и поместите в нее несколько записей.
2. Выберите все записи по одиночному значению.
3. Выберите все записи со значениями, которые больше или меньше определенной величины.
4. Воспользуйтесь оператором AND для вывода диапазона значений.
5. Воспользуйтесь оператором BETWEEN для вывода диапазона значений.
6. Исключите вывод записей из диапазона значений с помощью оператора OR и AND.
7. Исключите вывод записей из диапазона значений с помощью оператора NOT и BETWEEN.
8. Для исключения только одного значения воспользуйтесь в условии оператором «<>».
9. Выведите все записи по группе допустимых значений с использованием оператора IN
10. Найдите записи в таблице с помощью шаблонов.
11. Найдите все записи, содержащие NULL значения.
12. Для поиска записей, содержащих данные в определенных столбцах, воспользуйтесь оператором IS NOT NULL:
13. Отсортируйте все данные по какому-либо столбцу.
14. Создайте еще одну таблицу и заполните ее записями, чтобы потом извлечь из нее уникальные значения.
15. Замените значение записи любого столбца на другое.
16. Удалите все записи, содержащие определенные записи.
17. Отмените добавление с помощью команды ROLLBACK.
18. Создайте точку сохранения и произведите откат всех операций до этой точки.

Содержание отчета и его форма

Отчет должен содержать

- название лабораторной работы;
- цель работы;
- порядок выполнения работы;
- результаты выполнения;
- вывод.

Контрольные вопросы и защита работы

Контрольные вопросы

1. Какие операторы служат для вывода диапазона значений?
2. Какие операторы используются для вывода записей из диапазона значений.
3. Для чего служит оператор «<>»?
4. Для чего нужен оператор IN?
5. Для чего нужен оператор %?
6. С помощью какого оператора можно найти NULL значения?
7. Для чего необходима комбинация операторов IS NOT NULL?
8. Команда для сортировки данных.
9. Команда для выборки уникальных значений.
10. Команда для обновления записей в таблице.
11. Команда для удаления записей в таблице.
12. Команда ROLLBACK.
13. Для чего нужны точки сохранения.

Защита работы

Защита работы заключается в

- выполнении заданий раздела «Методика и порядок выполнения работы»;
- ответах на контрольные вопросы;
- предоставлении отчета.

ЛАБОРАТОРНАЯ РАБОТА №3

Управление SQL*Plus

Цель и содержание

Цель работы: приобрести навыки форматирования выходных данных SQL*Plus, использования файлов сценария.

Теоретическое обоснование

Использование текстового редактора

После ввода в следующем приглашении SQL> команды EDIT (или ее сокращенного варианта, ED) SQL*Plus откроет текстовый редактор, используемый на вашем компьютере по умолчанию, и автоматически загрузит в него последнюю SQL-команду. Вы сможете отредактировать команду, «сохранить» ее в SQL*Plus, а затем выполнить отредактированный вариант (рис. 3.1).

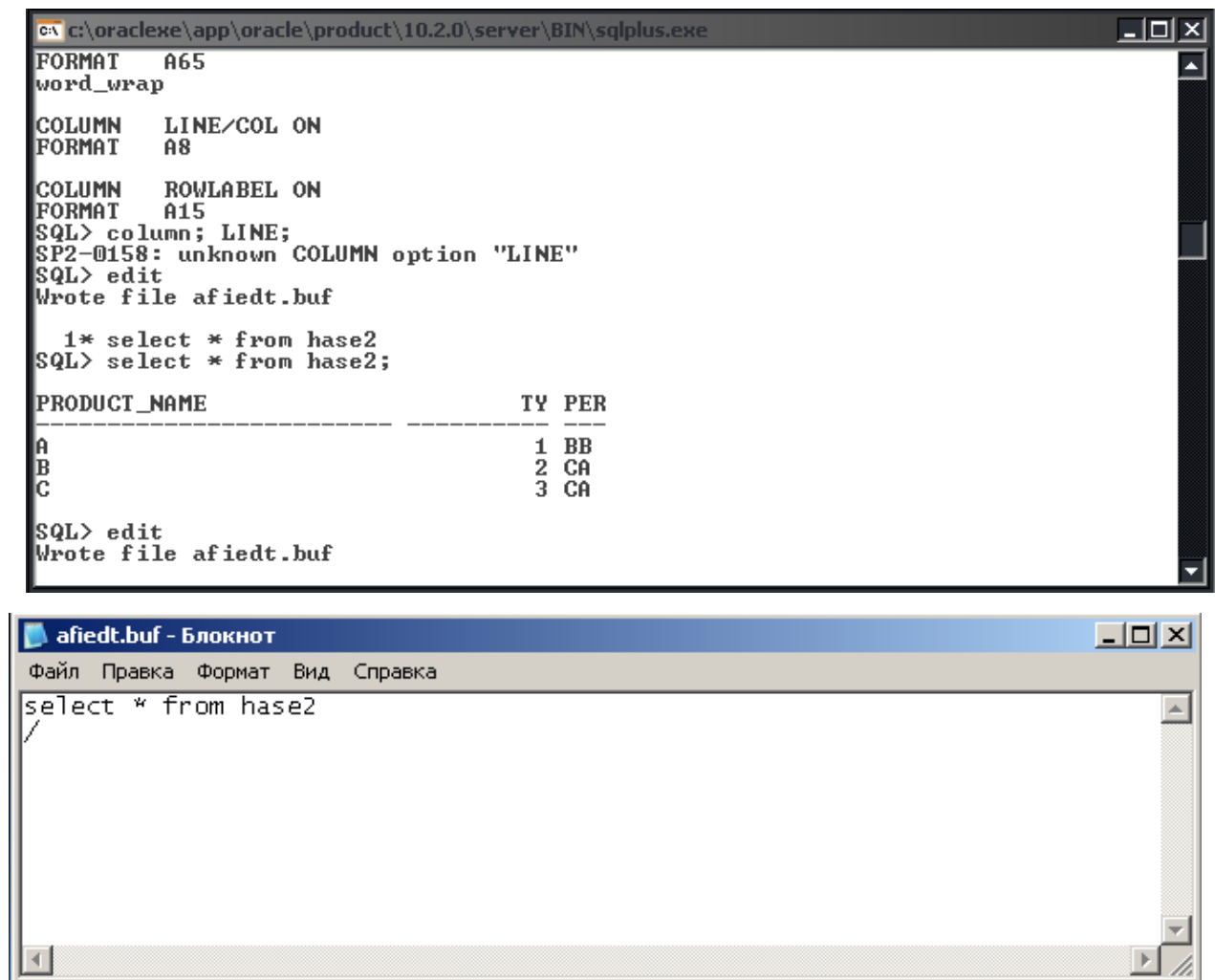


Рисунок 3.1 – Использование текстового редактора по умолчанию

После закрытия редактора отредактированная команда автоматически запишется в SQL*Plus. Чтобы ее выполнить, введите один прямой слэш (/) и

нажмите клавишу ENTER.

Форматирование выходных данных SQL*Plus

Вероятно, вы уже заметили, что SQL*Plus не выравнивает количество десятичных знаков в числах, обрезает заголовки столбцов по своему усмотрению и настойчиво показывает все содержимое большого текстового столбца в одной строке, независимо от того, насколько широким будет этот столбец на экране. Чтобы улучшить внешний вид выходных данных SQL*Plus существует команда COLUMN. Она не взаимодействует с базой данных. Ее единственная задача — повлиять на способ отображения информации вашей копией SQL*Plus. По этой причине команды COLUMN не требуется завершать точкой с запятой, как стандартные команды SQL. Кроме того, они действуют только во время сеанса работы с SQL*Plus. Если выйти из SQL*Plus, в следующий раз придется вводить все эти команды заново.

Форматирование чисел

Обычно с числами требуется осуществлять три действия:

- Выравнивать количество десятичных знаков
- Помещать разделитель между сотнями, тысячами и т.д.
- Добавлять знак денежной единицы

Выравнивание количества десятичных знаков

Команда COLUMN, выравнивающая количество десятичных знаков, имеет следующий синтаксис:

COLUMN *имя_столбца* FORMAT *код_формата*

Вместо аргумента *имя столбца* подставляется имя того столбца, который вы хотите отформатировать. Обратите внимание на то, что не упоминается, какой таблице принадлежит столбец. Команда COLUMN воздействует на все столбцы с указанным именем, независимо оттого, в какой таблице они находятся. Впрочем, если два столбца имеют одинаковые имена, они с большой вероятностью содержат похожие данные, и форматирование, примененное к одному из них, обычно имеет смысл и для другого.

Аргумент *код_формата* заменяется на представление числа. Это пред-

ставление содержит по одной цифре «9» для каждого десятичного знака, который требуется для ваших чисел, а также символ «.» для обозначения десятичного разделителя.

Добавление разделителя групп разрядов

Разделитель групп разрядов (group separator) – это символ, разделяющий сотни, тысячи и т.д. в пределах числа. В столбце `hand` таблицы `product_n` содержатся значения, которые будут смотреться лучше при наличии запятой, отделяющей сотни от тысяч.

Таблица 3.1 – Коды числовых форматов

Элемент	Пример	Описание
1	2	3
\$	\$9999	Помещает знак доллара перед значением
,(запятая)	9,999	Помещает запятую в указанной позиции
. (точка)	99.99	Помещает десятичную точку в указанной позиции
MI	9999MI	Отображает знак минуса (-) после любого отрицательного числа
S	S9999	Помещает в указанной позиции знак плюса (+) для положительных чисел и знак минуса (-) для отрицательных
PR	9999PR	Окружает отрицательные числа угловыми скобками (O)
D	99D99	Отображает в указанной позиции десятичный разделитель, принятый в стране
G	9G999	Отображает в указанной позиции разделитель групп разрядов, принятый в вашей стране
C	C999	Отображает в указанной позиции знак денежной единицы ISO
L	L999	Отображает в указанной позиции знак местной денежной единицы
RN	RN	Отображает числа римскими цифрами верхнего или нижнего регистра (только для целых чисел от 1 до 3999)
0	0999	Отображает один и более начальных нулей
0	9990	Отображает пустые значения в виде нулей

Форматирование заголовков столбцов

Все рассмотренные выше способы позволяют улучшать внешний вид отображаемых записей, тогда как заголовки столбцов над этими записями остаются в беспорядке. Помимо того, что имена столбцов состоят только из за-

главных букв, в некоторых случаях они превышают ширину самих столбцов и поэтому обрезаются. Об устранении этих проблем позаботится специальная разновидность команды COLUMN. Вот ее синтаксис

```
COLUMN имя_столбца HEADING 'текст_заголовка' JUSTIFY LEFT
      или
COLUMN имя_столбца HEADING 'текст_заголовка' JUSTIFY CENTER
      или
COLUMN имя_столбца HEADING 'текст_заголовка' JUSTIFY RIGHT
```

Кроме управления содержимым заголовков, мы получаем возможность использовать смесь символов верхнего и нижнего регистров и разбивать заголовки на несколько строк. Вертикальная черта (|) в аргументе *текст_заголовка* обозначает перевод строки. При этом указать, как должен быть выровнен заголовок: по левому краю, по правому краю, или по центру.

Также можно все опции команды COLUMN использовать одновременно.

Для отключения форматирования, заданного командой COLUMN, используется следующий синтаксис

```
COLUMN имя_столбца OFF
```

Буферизация выходных данных на диске

Буферизация (spooling) – это процесс записи информации в дисковый файл. Иногда это удобно делать прямо из SQL*Plus, чтобы сохранить серию команд с результатами их выполнения или объемные выходные данные одной команды.

Команда SPOOL имеет следующий синтаксис

```
SPOOL имя_буферного_файла
```

При желании можно включить в имя буферного файла расширение (например, .sqlmm .prn). Если расширение не указано, имя автоматически дополняется расширением .lst. Кроме того, имя буферного файла может содержать *путь* (path), т.е. имя дискового накопителя и каталога, где должен быть

сохранен файл. Если путь не указан, буферный файл сохраняется в подкаталоге BIN базового каталога.

Чтобы увидеть, как выполняется буферизация, введем перечисленные ниже команды. Обратите внимание, что после команд SPOOL не нужно ставить точку с запятой, поскольку они являются внутренними командами SQL*Plus и не влияют на работу сервера.

Файлы сценариев SQL

Для бизнес-сред характерно наличие операций, выполняющихся одинаковым (или почти одинаковым) образом по многу раз. Повторный набор одних и тех же команд может быстро утомить. Вместо этого мы можем сохранять часто используемые команды в дисковом файле. Такой подход имеет три основных преимущества: экономит время, устраняя необходимость повторного набора команд; позволяет завершать процедуру в более короткие сроки, поскольку команды считываются с диска намного быстрее, чем вы их набираете: гарантирует, что при каждом выполнении команды будут иметь один и тот же выверенный синтаксис.

Создание файла сценария

Файл сценария (script file) представляет собой обычный текстовый файл. Создать его можно в любом текстовом редакторе или процессоре. При использовании текстового процессора необходимо сохранить файл командой Save As в формате «text only», чтобы он не содержал кодов форматирования. Можно воспользоваться командой EDIT программы SQL*Plus, чтобы запустить стандартный текстовый редактор операционной системы и создать в нем новый файл сценария.

Запуск сценария

Чтобы запустить сценарий в SQL*Plus следует поставить перед именем файла знак «at» (@). Например:

```
@c:\test
```

В команду не потребовалось включать расширение .sql. Если расширение не указано, предполагается, что файл имеет расширение .sql.

Использование переменных в файлах сценариев

Временами возникает потребность в сценарии, который мог бы выполнять разные действия в зависимости от ситуации. Этого можно достичь за счет использования *переменных* (variables). Переменная заменяет часть команды, позволяя вводить эту часть при выполнении сценария и тем самым влиять на его работу. Противоположностью переменным является информация, явно указанная в файле сценария. Информация такого типа называется *жестко закодированной* (hard-coded), поскольку ее нельзя изменять во время выполнения сценария. Переменные можно встраивать в сценарии SQL двумя способами: использовать переменные подстановки или команду ACCEPT.

Переменные подстановки

Использование *переменной подстановки* (substitution variable) – самый простой способ встроить переменную в сценарий.

Команды SETVERIFY OFF и SETVERIFY ON помогают улучшить восприятие сценария. Без них SQL*Plus будет показывать старое и новое значение переменной подстановки перед выполнением команды SELECT, что может привести в замешательство любого другого пользователя, запускающего сценарий.

Переменные подстановки можно с тем же успехом использовать для текста и дат. При этом в SQL текст и даты требуется заключать в одиночные кавычки.

Команда ACCEPT

Приглашение, которое SQL*Plus выдает пользователям, встречая переменную подстановки, выглядит не слишком привлекательно. Альтернативой является команда ACCEPT, которая позволяет определять приглашение произвольного вида. Эта команда имеет следующий синтаксис

```
ACCEPT имя_переменной prompt 'текст_приглашения'
```

Методика и порядок выполнения работы

Методика выполнения работы

1. Добавим к нашей таблице product_n запись, требующую значительного

форматирования. Для этого введите следующий код:

```
INSERT INTO product_n VALUES ('Bar', 9.95, 1234);
```

2. Используем команду COLUMN для выравнивания количества десятичных знаков (рис. 3.2).

```
SELECT * FROM product_n;
COLUMN product_price FORMAT 9999.99
SELECT * FROM product_n;
```

The screenshot shows a SQL*Plus window with the following content:

```

C:\oracle\app\oracle\product\10.2.0\server\BIN\sqlplus.exe
SQL> select * from product_n;
PRODUCT_NAME          PRODUCT_PRICE      HAND
-----
Wood                   99                1
Bar                    9.95             1234
DVD                    50               100
Apple                  25              10000
Chrome full            75              1000
Apple full             25
6 rows selected.

SQL> COLUMN product_price Format 9999.99
SQL> select * from product_n;
PRODUCT_NAME          PRODUCT_PRICE      HAND
-----
Wood                   99.00             1
Bar                    9.95             1234
DVD                    50.00             100
Apple                  25.00             10000
Chrome full            75.00             1000
Apple full             25.00
6 rows selected.

SQL> _

```

Рисунок 3.2 – Результат использование команды COLUMN

3. Для добавления разделителя групп разрядов воспользуемся тем же синтаксисом COLUMN, который использовался для выравнивания количества десятичных знаков, изменив код формата

```
COLUMN hand FORMAT 99,999
```

Значения столбца hand содержат запятые в соответствующих местах (рис. 3.3).

```

c:\oracle\app\oracle\product\10.2.0\server\BIN\sqlplus.exe
SQL> select * from product_n;

PRODUCT_NAME          PRODUCT_PRICE    HAND
-----
Wood                   99,00           1
Bar                    9,95          1234
DVD                   50,00           100
Apple                 25,00         10000
Chrome full          75,00           1000
Apple full           25,00
6 rows selected.

SQL> COLUMN hand Format 99,999
SQL> select * from product_n;

PRODUCT_NAME          PRODUCT_PRICE    HAND
-----
Wood                   99,00           1
Bar                    9,95          1,234
DVD                   50,00           100
Apple                 25,00         10,000
Chrome full          75,00           1,000
Apple full           25,00
6 rows selected.

SQL>

```

Рисунок 3.3 – Результат применения разделителя групп разрядов

4. Для добавления знака денежной единицы, поместим знак денежной единицы (например знак \$) перед каждым значением product_price

```

COLUMN product_price FORMAT $99.99
SELECT * FROM product_n;

```

5. Расставим заголовки столбцов по центру

```

SELECT * FROM product_n;

COLUMN product_name HEADING 'Product' JUSTIFY CENTER

```

```

SELECT * FROM product_n;

```

6. Создадим файл буферизации выходных данных на диске.

```

SPOOL c:\test.prn
SELECT * FROM product_n;
SPOOL OFF

```

После выполнения этих команд в файле test.prn будет все, что выводилось на экран SQL*Plus на протяжении сеанса работы.

7. Используем переменные подстановки для значения price. Для этого создадим файл сценария с именем test2.sql и поместим в него следующие команды

```

SET VERIFY OFF

```

```
SELECT product_name, product_price FROM product_n
WHERE product_price >= &price;

SET VERIFY ON
```

8. . Используем переменные подстановки с помощью команды ACCEPT.

Создадим сценарий test3.sql и поместим в него следующие команды (рис.3.4)

```
SET VERIFY OFF
SET ECHO OFF

ACCEPT name PROMPT 'Enter name? (text)'

SELECT product_name, product_price
FROM product_n WHERE product_name = '&name';

SET VERIFY ON
SET ECHO ON
```

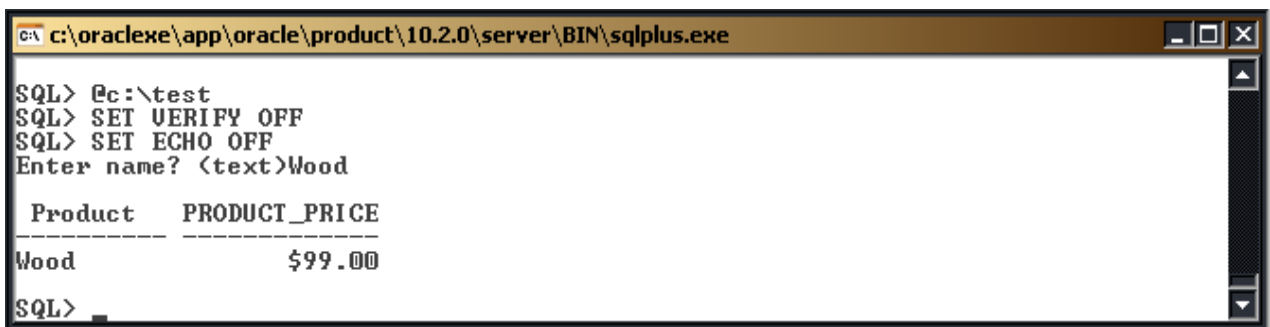


Рисунок 3.4 – Результат выполнения сценария test3.sql

Порядок выполнения работы

1. Используйте команду COLUMN для выравнивания количества десятичных знаков в вашей таблице.
2. Используйте команду COLUMN для добавления разделителя групп разрядов в вашей таблице.
4. Добавьте столбец в вашу таблицу, значения которого соответствуют формату денежной единицы.
5. Расставьте заголовки столбцов по правому краю.
6. Создайте файл для буферизации выходных данных на диск.
7. Воспользуйтесь переменной подстановки для любого значения в вашей таблице.

8. Используйте команду АССЕРТ в вашей таблице.

Содержание отчета и его форма

Отчет должен содержать

- название лабораторной работы;
- цель работы;
- порядок выполнения работы;
- результаты выполнения;
- вывод о полученных навыках форматирования выходных данных SQL*Plus и использования файлов сценария.

Контрольные вопросы и защита работы

Контрольные вопросы

1. Синтаксис команды COLUMN для чисел.
2. Для чего нужны файлы сценариев SQL?
4. Синтаксис команды COLUMN для форматирования заголовков столбцов.
5. Команда для создания файла буферизации выходных данных.
7. Для чего нужны переменные подстановки?
8. Какие отличия переменных подстановки для чисел от текста?
9. Синтаксис команды АССЕРТ.

Защита работы

Защита работы заключается в

- выполнении заданий раздела «Методика и порядок выполнения работы»;
- ответах на контрольные вопросы;
- предоставлении отчета.

ЛАБОРАТОРНАЯ РАБОТА №4

Встроенные функции SQL

Цель и содержание

Цель работы: приобрести навыки и умения использовать различные встроенные функции, системные переменные.

Теоретическое обоснование

Функция – это нечто вроде мини-команды, которую можно помещать в более крупный SQL-оператор. Функция принимает некоторые данные и возвращает их в преобразованном виде, в зависимости от того, для какой цели она разрабатывалась. В частности, функции могут изменять внешнее представление данных, выполнять их статистическую обработку или изменять содержание (например, преобразовывать один набор кодов в другой).

Все функции разделены на две категории: однострочные и групповые. Однострочные функции выполняют операции, которые могут повлиять на каждую строку таблицы. В отличие от них групповые функции предназначены для получения информации о некоторых подмножествах данных, выбранных любым необходимым для вас образом.

Системные переменные

Системные переменные создаются и содержат информацию о среде, в которой функционирует база данных. Системные переменные, описанные далее, позволяют определять системные дату и время, идентификатор пользователя, выполняющего SQL-оператор, и имя компьютера, с применением которого пользователь вводит команды.

SYSDATE

Функция SYSDATE возвращает текущие дату и время. Точнее, она возвращает дату и время, которые являются текущими с точки зрения сервера Oracle, поэтому если сервер окажется в другом часовом поясе, то возвращаемая информация будет относиться именно к этому часовому поясу.

Функция SYSDATE применяется также при просмотре записей с датами, которые определенным образом связаны с сегодняшним днем.

USER и USERENV

Обе эти функции полезны, когда требуется вести аудит действий над таблицей, т. е. следить за тем, кто вставляет, обновляет и удаляет записи. Функция USER возвращает идентификатор пользователя, который выдал команду, содержащую эту функцию.

Функция USERENV может возвращать множество различных сведений о вычислительной среде, в которой была выдана содержащая ее команда.

Числовые функции

Числовые функции оперируют с числовыми значениями, изменяя их в соответствии с потребностями.

ROUND

Функция ROUND округляет числа с любой заданной точностью. Она имеет следующий синтаксис

ROUND (входное_значение, число_знаков_после_десятичной_
точки)

Для использования функции ROUND (как и любой другой функции, модифицирующей значение) ей необходимо передать значение, подлежащее модификации. Поскольку это делается в операторе SELECT, следует передавать функции имя столбца, содержащего модифицируемые значения.

Если указать отрицательное число, то функция ROUND округлит число до десятичной точки, т. е. числа будут округляться до ближайших десятков, сотен, тысяч и т. д.

TRUNC

Функция TRUNC усекает число, понижая его точность. Различие между усечением и округлением проявляется, когда за последним из оставшихся десятичных разрядов идет значение 5 и выше.

Текстовые функции

Текстовые функции, называемые *символьными функциями* (character functions), оперируют с текстовыми строками. Чаще всего с текстовыми строками требуются следующие действия

- изменение регистра символов (на верхний, нижний или смешанный);
- разбиение длинных строк на несколько более коротких подстрок;
- избавление текста, поступающего из внешнего источника, от избыточ-

ных в конце пробелов.

UPPER, LOWER и INITCAP

Эти функции меняют регистр переданного им текста.

Из всех трех символьных функций, меняющих регистр, чаще всего используется функция UPPER. Она востребована в операторах SELECT, когда не ясно, какими буквами был набран текст в столбце. В таких случаях достаточно подставить имя столбца в функцию UPPER, а искомый текст набрать в верхнем регистре.

LENGTH

Временами требуется определять длину данных, хранимых в столбце таблицы. Такую возможность предоставляет функция LENGTH.

SUBSTR

В компьютерном мире текст называется *строкой* (string), поэтому фрагмент текста называется *подстрокой* (substring). Функция SUBSTR «вырезает» из строки подстроку. Чтобы «разрезать» строку на подстроки, необходимо указать, в какой позиции должна начинаться подстрока и какой длины она должна быть.

Синтаксис команды SUBSTR

SUBSTR (*исходный_текст*, *позиция_начального_символа*,
количество_символов) ;

Значение *исходный_текст* обычно представляет собой имя столбца, подлежащего разбору. *Позиция_начального_символа* – это символ, с которого будет начинаться подстрока, а *количество_символов* – общее количество символов, которое она должна содержать.

INSTR

Функция SUBSTR обычно используется в тех случаях, когда точно известна длина каждой подстроки. Однако, требуется разбирать строки, в которых подстроки имеют переменную длину. Это означает, что неизвестна не только длина первой подстроки, но и начальная позиция второй.

Единственный способ разбора такой строки состоит в нахождении позиции символа (или символов), разделяющего элементы строки. Именно для этого

и предназначена функция INSTR.

Функция INSTR осуществляет поиск указанного текста и возвращает число, обозначающее начальную позицию данного текста в строке. Используя это число для определения длины первой подстроки и начальной позиции следующей подстроки, можно уверенно «резать» длинные строки, независимо от того, как они устроены.

Синтаксис функции INSTR выглядит следующим образом

INSTR (*исходный_текст*, *текст_для_поиска*,
позиция_начального_символа) ;

Как правило, *исходный_текст* представляет собой имя столбца, содержащего длинную строку, которую необходимо разобрать. *Текст_для_поиска* – это текст, который необходимо найти, а *позиция_начального_символа* определяет номер символа исходного текста, с которого будет начинаться поиск (для поиска с самого начала текста, требуется указать значение 1).

Функция множественного выбора *DECODE*

Одним из различий между языком SQL и языком программирования PL/SQL, является то, что SQL не позволяет управлять последовательностью выполнения команд – в нем нет циклов, операторов «goto», «if-then-else». Сценарий SQL выполняется сверху вниз, строка за строкой, без ветвлений. В SQL отсутствуют средства принятия решений, но есть одна функция, которая в чем-то аналогична оператору if-then-else: DECODE. Эта функция транслирует одно множество данных в другое, используя определенные значения «до» и «после». Охарактеризовать ее работу можно следующим образом: если входное значение равно «А», на выходе будет «В», а если входное значение равно «С», функция вернет «D». Значения «А», «В», «С» и «D» определяются пользователем. Это может оказать неоценимую помощь при трансляции данных из одной системы баз данных в другую, поскольку разные системы обычно используют разные множества кодов для представления схожей информации.

Функция DECODE имеет следующий синтаксис

DECODE (*источник_входных_данных*,

```

входное_значение_1, выходное_значение_1,
входное_значение_2, выходное_значение_2,
... ,
последнее_входное_значение, последнее_выходное_значение,
[выход-
ное_значение_по_умолчанию_при_отсутствии_совпадений] );

```

Указав *источник_входных_данных* (обычно это имя столбца некоторой таблицы), определяются пары входных/выходных значений. Слева располагается одно из значений, которые будут поступать из источника данных, а справа – то, во что требуется транслировать это значение. Можно определить сколько угодно таких пар (именно об этом говорит многоточие между строками).

Определив пары *входных/выходных* значений, можно отдельно указать конечный результат, которому не соответствует какое-либо определенное входное значение. Он будет возвращен в том случае, если функция DECODE встретит значение, отсутствующее в списке (по аналогии с блоком else оператора if-then-else). Эта часть функции необязательна (поэтому и заключена в квадратные скобки), но обычно ее стоит указывать для того, чтобы знать, не встретились ли среди входных данных какие-нибудь другие значения.

Вставка комментариев в SQL-сценарии

Существует два типа комментариев. Комментарии первого типа занимают одну строку и хорошо подходят в качестве небольших памяток или для временного блокирования части кода, который нет необходимости выполнять, но и нежелательно удалять. Комментарии второго типа могут содержать сколько угодно строк, но не столь очевидны, поэтому не бросаются в глаза при чтении файла. И тот, и другой тип находят свое применение, и во многих файлах сценариев они встречаются одновременно.

Чтобы создать однострочный комментарий, необходимо поставить в начале строки два дефиса. Все, что последует за дефисами, СУБД будет игнорировать.

Если требуется сгруппировать несколько строк комментариев, более подходящим может оказаться другой подход, при котором в начале секции комментария помещается «/*», а в конце – «*/». Все строки между этими парами символов игнорируются.

Групповые функции

SUM

Функция SUM суммирует значения и возвращает итог.

COUNT

Функция COUNT, подсчитывает записи. Указывая «*» вместо имен столбцов, СУБД считать всю таблицу. Это не так важно для маленьких учебных таблиц, но представляет собой серьезную проблему в системах с сотнями тысяч, а тем более с миллиардами записей. Полное сканирование таблицы намного замедляет выдачу результата и способствует тому, что СУБД отводит компьютерные ресурсы от главных задач, для выполнения которых предназначалась база данных, что тормозит работу всей компании.

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

MIN

Функция MIN возвращает наименьшее из значений множества записей, удовлетворяющих предикату запроса.

MAX

Функция MAX возвращает наибольшее из значений множества записей, удовлетворяющих предикату запроса.

Группирование данных с помощью конструкции GROUP BY

Для создания групп служит конструкции GROUP BY в операторе SELECT. После конструкции GROUP BY указывается столбец, по значениям которого будет выполняться группирование. Обычно это первый столбец в

списке SELECT.

Включение и исключение групп с помощью конструкции HAVING

Конструкция WHERE позволяет фильтровать записи, возвращаемые оператором SELECT. При группировании записей конструкция WHERE работает точно также: фильтрует отдельные записи, тем самым исключая их из вычислений, выполняемых групповыми функциями.

Однако после создания групп возникает новая задача – фильтрация самих групп на основе групповой информации. Предположим, что некоторой компании не хватает складских площадей и она намерена сократить запас товаров на складе. Для поддержки этого мероприятия требуется составить список дешевых товаров – например, тех, у которых стоимость составляет менее 75\$. Здесь и применяется конструкция HAVING. Она фильтрует группы на основе *групповых значений*. В отличие от конструкции WHERE, фильтрующей записи до группирования, конструкция HAVING фильтрует уже сформированные группы.

Методика и порядок выполнения работы

Методика выполнения работы

1. Используем системную переменную SYSDATE. Введем следующую команду

```
SELECT  SYSDATE  FROM  DUAL;
```

В ответ на экране будет показана текущая дата (рис. 4.1).

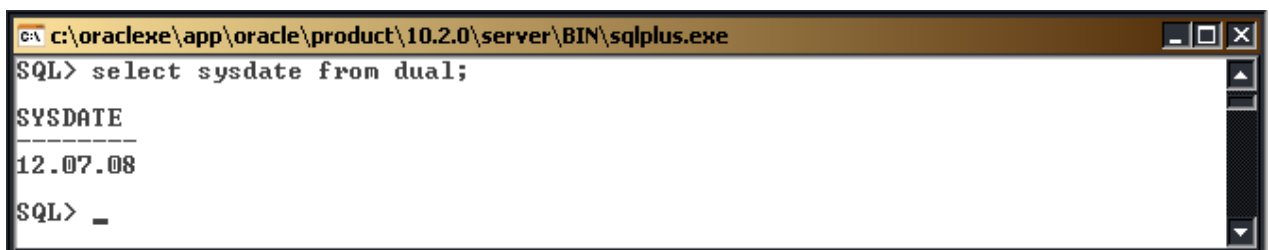


Рисунок 4.1 – Результат выполнения функции SYSDATE

2. Укажем SYSDATE в операторе INSERT, чтобы вставить текущую дату в любое поле cur.

```
CREATE TABLE hase3(name VARCHAR2(8), cur DATE);
INSERT INTO hase3 VALUES ('Small', SYSDATE);
```

3. Используем системную переменную SYSDATE для отображения всех данных за последние 2 дня.

```
SELECT * FROM hase3 WHERE cur BETWEEN (SYSDATE-2) AND SYS-
DATE;
```

4. Воспользуемся системной переменной USER, чтобы увидеть имя пользователя, под которым вошли в систему.

```
SELECT USER FROM DUAL;
```

5. Применим системную переменную USERENV для того, чтобы увидеть имя компьютера, на котором работает пользователь.

```
SELECT USERENV('TERMINAL') FROM DUAL;
```

6. Используем функцию ROUND для округления числа (рис. 4.2).

```
SELECT ROUND(1.19345, 1) FROM DUAL;
```

7. Используем функцию TRUNC для понижения точности числа (усечение).

```
SELECT TRUNC (1.19345, 1) FROM DUAL;
```

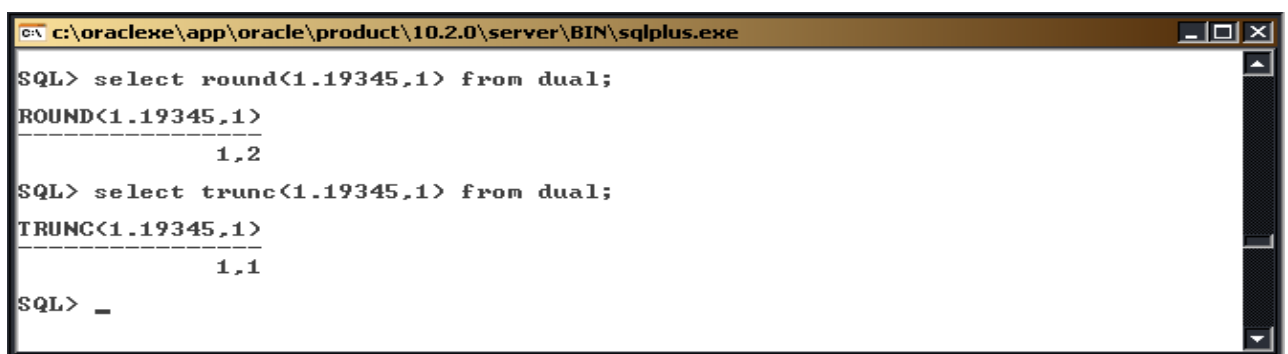


Рисунок 4.2 – Результат выполнения функций ROUND и TRUNC

8. Изменим регистр букв с помощью UPPER, LOWER и INITCAP (рис. 4.3).

```
SELECT UPPER(product_name) FROM product_n;
SELECT LOWER(product_name) FROM product_n;
```

```
SELECT INITCAP (product_name) FROM product_n;
```

```

C:\oracle\app\oracle\product\10.2.0\server\BIN\sqlplus.exe
SQL> select upper<product_name> from product_n;
UPPER<PRODUCT_NAME>
-----
WOOD
DVD
APPLE
CHROME FULL
APPLE FULL

SQL> select lower<product_name> from product_n;
LOWER<PRODUCT_NAME>
-----
wood
dvd
apple
chrome full
apple full

SQL> select initcap<product_name> from product_n;
INITCAP<PRODUCT_NAME>
-----
Wood
Dvd
Apple
Chrome Full
Apple Full

SQL> _

```

Рисунок 4.3 – Результаты выполнения функций UPPER, LOWER и INITCAP

9. Применим функцию LENGTH для вывода самых длинных названий товаров в таблице.

```
SELECT product_name, LENGTH(product_name) NAME_LENGTH
FROM product_n WHERE LENGTH(product_name) > 5 ORDER BY product_name;
```

```

C:\oracle\app\oracle\product\10.2.0\server\BIN\sqlplus.exe
SQL> select product_name, length<product_name> NAME_LENGTH from product_n where
length<product_name>>15 order by product_name;
no rows selected

SQL> select product_name, length<product_name> NAME_LENGTH from product_n where
length<product_name>>5 order by product_name;
PRODUCT_NAME          NAME_LENGTH
-----
Apple full              10
Chrome full             11

SQL>

```

Рисунок 4.4 – Результат выполнения функции LENGTH

10. Получи подстроку с помощью функции SUBSTR (рис. 4.5).

```
SELECT SUBSTR(product_name,1,3) FROM product_n;
```

```

C:\oracle\app\oracle\product\10.2.0\server\BIN\sqlplus.exe
SQL> select substr<product_name,1,3> from product_n;

SUB
---
Woo
DUD
App
Chr
App
SQL> _

```

Рисунок 4.5 – Результат выполнения функции SUBSTR

11. Произведем поиск вхождения подстроки в строке с помощью функции INSTR. Как показано на рис. 4.6, функция INSTR возвращает число, обозначающее положение подстроки «full» в каждой записи.

```
SELECT INSTR(product_name, 'full',1) FROM product_n;
```

```

C:\oracle\app\oracle\product\10.2.0\server\BIN\sqlplus.exe
SQL> select instr<product_name, 'full',1> from product_n;

INSTR<PRODUCT_NAME,'FULL',1>
-----
0
0
0
8
7
SQL>

```

Рисунок 4.6 – Результат выполнения функции INSTR

12. Рассмотрим применение функции DECODE для множественного выбора (рис. 4.7).

```

SELECT DECODE (SUB-
STR(product_name,1,3), 'Woo', 'OK', 'ERROR')
FROM product_n;

```

```

C:\oracle\app\oracle\product\10.2.0\server\BIN\sqlplus.exe
SQL> select decode<substr<product_name,1,3>,'Woo','OK','ERROR'> from product_n;

DECODE
-----
OK
ERROR
ERROR
ERROR
ERROR
SQL> _

```

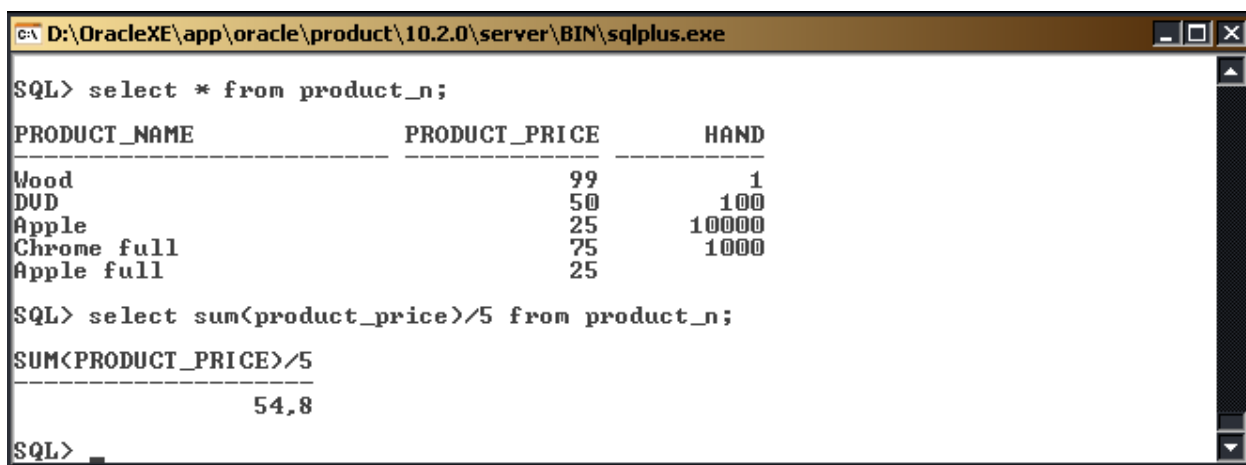
Рисунок 4.7 – Результат выполнения функции DECODE

13. Осуществим вставку комментариев в SQL-сценарии.

```
SELECT * FROM product_n;
-- Эта строка игнорируется. СУБД не будет пытаться ее
      ВЫПОЛНИТЬ
SELECT * FROM hase3;
/*Этот сценарий демонстрирует использование
многострочных комментариев.*/
SELECT * FROM product_n;
```

14. Произведем подсчет средней стоимости товаров в таблице product_n с помощью SUM (рис. 4.8).

```
SELECT * FROM product_n;
SELECT SUM(product_price)/5 FROM product_n;
```



SQL> select * from product_n;

PRODUCT_NAME	PRODUCT_PRICE	HAND
Wood	99	1
DVD	50	100
Apple	25	10000
Chrome full	75	1000
Apple full	25	

SQL> select sum<product_price>/5 from product_n;

SUM<PRODUCT_PRICE>/5
54,8

SQL> _

Рисунок 4.8 – Результат выполнения функции SUM

15. Подсчитаем количество товара с помощью функции COUNT.

```
SELECT COUNT(product_name) FROM product_n;
```

16. Узнаем, сколько стоит самый дешевый товар из таблицы product_n с помощью функции MIN.

```
SELECT MIN(product_price) FROM product_n;
```

17. Чтобы узнать максимальную цену товара в таблице product_n, воспользуемся функцией MAX.

```
SELECT MAX(product_price) FROM product_n;
```

18. Подсчитаем стоимость товара по группам (рис. 4.9).

```
INSERT INTO product_n VALUES ('DVD',66,100);

SELECT product_name, SUM(product_price)

FROM product_n GROUP BY product_name;
```

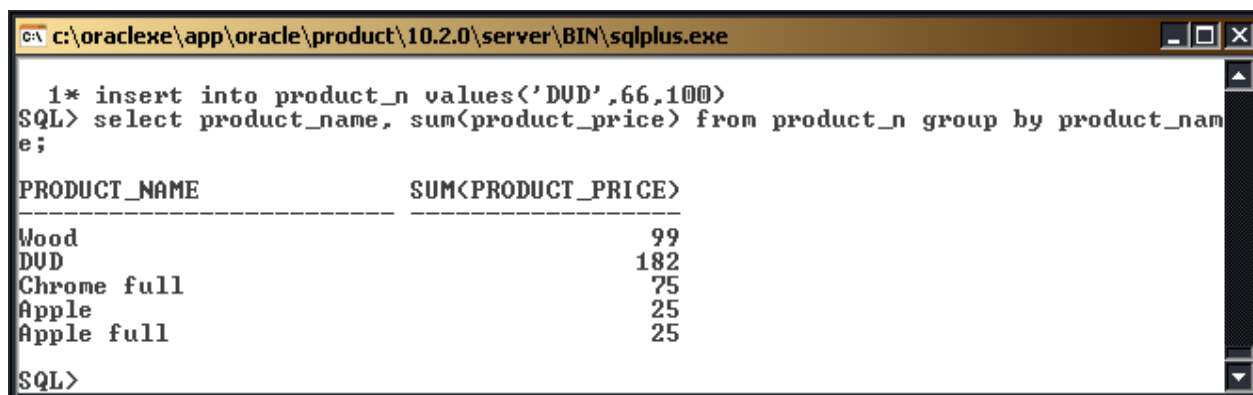


Рисунок 4.9 – Группирование данных с помощью конструкции GROUP BY

19. Воспользуемся конструкцией HAVING для нахождения самых дешевых товаров.

```
SELECT product_name, SUM(product_price)

FROM product_n GROUP BY product_name

HAVING SUM(product_price) < 75;
```

Порядок выполнения работы

1. Используйте системную переменную SYSDATE для добавления текущей даты.
2. Используйте системную переменную SYSDATE, чтобы увидеть все данные за последние 4 дня.
3. Воспользуйтесь системной переменной USER и USERENV, чтобы определить, кто и с какого компьютера обращался к базе данных.
4. Используйте функцию ROUND для округления числа.
5. Используйте функцию TRUNC для усечения числа до 2 разрядов.
6. Измените регистр букв с помощью функций UPPER и LOWER.

7. С помощью функции LENGTH узнайте самые длинные записи в таблице.
8. Воспользуйтесь функцией SUBSTR для вывода первых 3 символов строки.
9. Выведите начальную позицию подстроки в строке с помощью функции INSTR.
10. Используйте функцию DECODE для любого множественного выбора.
11. Вставьте однострочный комментарий в ваш SQL-сценарий.
12. Подсчитайте сумму всех записей столбца с помощью функции SUM.
13. Подсчитайте количество записей в вашей таблице с помощью функции COUNT.
14. Найдите самое маленькое число в вашей таблице.
15. Найдите самое большое число в вашей таблице.
18. Объедините все записи любого столбца в группы.
19. Воспользуйтесь конструкцией HAVING чтобы отфильтровать группы.

Содержание отчета и его форма

Отчет должен содержать

- название лабораторной работы;
- цель работы;
- порядок выполнения работы;
- результаты выполнения;
- вывод о результатах использования различных встроенных функций и системных переменных.

Контрольные вопросы и защита работы

Контрольные вопросы

1. Что представляют собой системные переменные?
2. Для чего предназначена системная переменная SYSDATE?
3. Системные переменные USER и USERENV.

4. Функция ROUND и TRUNC.
6. Функции UPPER и LOWER.
7. Функция LENGTH.
8. Функция SUBSTR.
9. Функция INSTR..
10. Функция DECODE.
11. Комментарии в SQL-сценариях.
12. Функция SUM.
13. Функция COUNT.
14. Функции MAX и MIN.
15. Оператор GROUP BY..
16. Для чего нужна конструкция HAVING?

Защита работы

Защита работы заключается в

- выполнении заданий раздела «Методика и порядок выполнения работы»;
- ответах на контрольные вопросы и предоставлении отчета.

ЛАБОРАТОРНАЯ РАБОТА №5

Индексы и ограничения

Цель и содержание

Цель работы: приобрести навыки создания и использования индексов БД и ограничений.

Теоретическое обоснование

Несомненно, вы знакомы с индексными указателями (индексами), размещаемыми на последних страницах книг. По ним можно увидеть слова, обозначающие каждую тему или понятие, за которыми следует номер страницы (или несколько номеров). Пролистав индекс и найдя нужную тему, вы можете сразу перейти на страницы, имеющие к этой теме отношение. Индексы удобны тем, что позволяют находить в книге определенную информацию, не просматривая

все страницы подряд.

Индексы в базах данных

Тот же принцип индексирования можно применить к таблице базы данных. Когда таблица содержит большое число записей, ее полный просмотр с целью поиска определенных записей может занять у СУБД (как и у любой другой СУБД) много времени – сравните это с перелистыванием всей книги в поиске страниц, где обсуждается определенная тема. СУБД позволяет создавать внутритабличную структуру на основе значений одного или нескольких столбцов таблицы, называемую индексом. Индексы обеспечивают не только быстрый доступ к данным, но и логический порядок данных в таблицах. Тщательная организация индексов – самый эффективный способ повысить эффективность работы с таблицами баз данных. Поэтому функции индекса во многом схожи с функциями индекса (алфавитного указателя) книги: по индексу можно найти конкретное значение, а затем, по указателю, строку с этим значением. При создании индекса для таблицы БД получает указание просканировать таблицу и выявить отличие друг от друга значения в столбце или группе столбцов. Пользуясь полученной информацией, создает список страниц данных, размещаемый на индексной странице для индексируемого значения. Впоследствии это позволит искать значение в индексе, а не просматривать строку за строкой.

Индексы применяются в первую очередь для ускорения доступа к данным. Они не являются обязательным элементом, однако искать данные в таблицах без индексов намного труднее, чем искать данные по столбцу, индексированному надлежащим образом. Для определения самого эффективного способа исполнения запроса существует программный компонент, называемый оптимизатором запросов. Его основой являются индексы. Однако необходимо учесть, что слишком много индексов также не очень хорошо. Построение индекса требует много времени и пространства хранения информации. Индексы затрудняют модификацию таблиц и их данных. Из-за того, что при изменении таблицы воздействию могут подвергнуться и ее индексы, операторы IN-

SERT, UPDATE и DELETE выполняются медленнее. Создание индексов представляет собой балансирование между повышением и снижением производительности системы.

Таким образом, использование индексов – это компромисс. Они замедляют ввод данных, но ускоряют их считывание. Следовательно, индексирование таблиц нежелательно в тех приложениях, где ввод данных должен происходить как можно быстрее. Например, в системе, обслуживающей сеть розничных магазинов, от кассовых терминалов требуется максимальная скорость выполнения транзакций. В этом случае индексирование таблицы транзакций будет ошибкой, поскольку приведет к замедлению ввода данных. С другой стороны, в той же самой компании могут работать специалисты, которым нужно выполнять запросы, анализирующие транзакции, а эти запросы сильно выиграют от наличия правильно индексированных таблиц. Во многих системах данные о транзакциях каждую ночь автоматически копируются из рабочих таблиц во вспомогательные индексированные таблицы, которые на следующий день используются для анализа. Вставка данных в индексированную таблицу занимает намного больше времени, чем вставка в таблицу транзакций, не имеющую индексов, но это никого не волнует, поскольку магазины закрыты, покупателей нет, а вся работа выполняется компьютерами.

Чем больше таблица, тем больший выигрыш можно получить от ее индексирования. Например, все созданные в предыдущих главах таблицы настолько малы, что любые операции с ними выполняются практически мгновенно. Как следствие, индексирование этих таблиц не даст заметной выгоды. По мере роста таблиц выигрыш увеличивается.

Создание индекса

Для создания индекса используется команда CREATE INDEX ее синтаксис:

```
CREATE INDEX имя_индекса ON имя_таблицы (имя_столбца) ;
```

Если нужно, чтобы индекс содержал более одного столбца, используется следующий синтаксис

```
CREATE INDEX имя_индекса ON имя_таблицы (
```

имя_первого_столбца, имя_второго_столбца, ...);

Составной индекс – это просто индекс по двум и более столбцам таблицы. Он подходит в любом случае, когда некоторые столбцы таблицы с большой вероятностью будут использоваться в конструкции WHERE совместно – например, имя и фамилия, или город, почтовый индекс. Чтобы СУБД использовал составной индекс, в конструкцию WHERE должны быть включены либо *все* его столбцы, либо *первый* столбец. Столбцы можно помещать в индекс в определенном порядке, не обязательно так, как они расположены в таблице, поэтому вы можете сделать первым столбцом тот, который вероятнее всего будет использоваться в конструкции WHERE. Столбцы составного индекса не обязаны быть соседними в исходной таблице. Максимальное число столбцов, которые можно включать в стандартный индекс, равно 32.

Удаление индекса

Команда удаления индекса имеет следующий синтаксис

```
DROP INDEX имя_индекса;
```

Различные типы индексов

Индексирование – это, по существу, способ компактного хранения информации о местонахождении записей, организованной на основе содержимого этих записей. Содержимое базы данных может сильно варьироваться от системы к системе, и для разных типов содержимого оптимальными оказываются разные виды организации.

Индексы B-дерева*

Как следует из названия структуры *B*-дерева*, ее блоки организованы в виде древовидного графа (tree-like graph).

При создании индекса *B*-дерева* СУБД анализирует значения в индексируемом столбце (столбцах) и определяет, как разделить таблицу на *блоки-листья* (leafblocks) с равным числом записей. Затем он создает уровни *блоков-ветвей* (branch blocks), обеспечивающих поиск записей в нижележащих блоках-листьях.

Достоинство индекса *B*-дерева* в том, что он позволяет быстро

идентифицировать записи, не считывая их. За счет минимизации объема считываемых данных, а следовательно, и объема выполняемой работы, может возвращать результаты гораздо быстрее. Предположим, например, что таблица содержит миллиард записей, и нужно просмотреть запись с определенным идентификационным номером. Таблица не обязательно будет отсортирована по идентификационным номерам, поэтому не исключено, что должен будет считывать все записи, пока не найдет нужную. Но если для таблицы определен индекс *B*-дерева*, то Oracle сможет найти запись быстрее. На каждом шаге исключается половина записей таблицы, поэтому может сократить объем работы до разумных величин. Поскольку сила индекса *B*-дерева* заключена в делении данных на множества и подмножества, этот тип индекса оптимален в тех случаях, когда индексируемый столбец содержит широкий диапазон значений – скажем, имена или даты. Для столбца с узким диапазоном значений (например, «пол») лучше всего использовать битовый индекс.

Битовые индексы

Коль скоро структура индекса *B*-дерева* оптимальна для индексирования столбца со многими уникальными значениями, логично предположить, что для столбца с малым числом уникальных значений может лучше подойти другая структура индекса. Например, столбец «пол», скорее всего, будет содержать одно из трех значений: «М» («мужской»), «F» («женский») или «U» («Unknown» «неизвестный»). Помещение столь малого количества уникальных значений в структуру *B*-дерева* не имеет смысла, поскольку подход «делить на подгруппы шаг за шагом», делающий *B*-дерево* столь удобным для поиска многовариантных значений, не даст большого выигрыша. В такой ситуации лучше использовать битовый индекс.

Для тех запросов SELECT, в которых конструкция WHERE содержит столбец с низкой кардинальностью, предварительное создание битового индекса может значительно сократить время получения ответов. Рост скорости обусловлен двумя факторами: во-первых, битовые индексы могут быть довольно

компактными (поскольку для хранения бит требуется лишь малая часть того места, которое отводится другим типам данных в стандартном индексе), а во-вторых, значения «1» или «0» могут анализироваться компьютером очень быстро.

Команда создания битового индекса практически идентична команде создания стандартного индекса; добавляется только слово «BITMAP», как показано в следующем примере

```
CREATE BITMAP INDEX имя_индекса ON
      имя_таблицы (имя_столбца) ;
```

Хеш-таблицы

Хеш-таблица (hash tables) – это структура данных, реализующая интерфейс ассоциативного массива, а именно, она позволяет хранить пары (ключ, значение) и выполнять три операции: операцию добавления новой пары, операцию поиска и операцию удаления пары по ключу.

Каждой подобной структуре ставится в соответствие некоторая *хеш-функция* (hash function) , принимающая в качестве параметра значение ключа поиска (*хеш-ключа* – hash key) и вычисляющая число в интервале от 0 до B-1, где B – количество *сегментов* (buckets). Элементы *массива сегментов* (bucket array) проиндексированы от 0 до B-1 и содержат заголовки связанных списков (их количество равно B), по одному на каждый элемент-сегмент массива. Если запись обладает значением K ключа поиска, она присоединяется к списку сегмента с номером h(K), где h – *хеш-функция*.

Ограничение

Ограничение (constraint) – это одно и более условий, которым должна удовлетворять введенная пользователем запись, чтобы вставить ее в таблицу. Ограничения хранятся как часть определения таблицы и после создания применяются автоматически. Когда введенная кем-либо команда INSERT или UPDATE нарушает ограничение, прерывает ее выполнение, производит откат и выдает сообщение об ошибке.

NOT NULL

Этот тип ограничения служит для того чтобы в столбцах не допускались

null-значения при вставке или обновлении записей. Команда изменения статуса существующего столбца на NOT NULL имеет следующий синтаксис:

```
ALTER TABLE имя_таблицы MODIFY (имя_столбца NOT NULL);
```

UNIQUE

Такой синтаксис используется для создания ограничения, гарантирующего уникальность значений

```
ALTER TABLE имя_таблицы ADD CONSTRAINT имя_ограничения
UNIQUE (имена_столбцов);
```

Когда создается ограничение уникальности, неявно *создает уникальный индекс* (unique index). Этот индекс содержит столбцы, указанные в команде ALTER TABLE ... ADD CONSTRAINT, а его имя совпадает с именем ограничения. Уникальный индекс помогает идентифицировать повторяющиеся записи. Создавая этот индекс, считывает все существующие записи таблицы и представляет их в индексе. По этой причине создание ограничения уникальности будет успешным только в том случае, если оно не нарушается текущими записями таблицы. Если таблица содержит записи с повторяющимися значениями в указанных вами столбцах, то в ответ на команду ALTER TABLE ... ADD CONSTRAINT будет выдано сообщение об ошибке с объяснением проблемы.

CHECK

Контрольное ограничение (check constraint) позволяет определить, каким условиям должны удовлетворять входные данные, чтобы разрешил их ввод. Контрольное ограничение можно определять для каждого столбца таблицы. Например, вы можете объявить, что значения в столбце с ценами должны быть положительными числами, и одновременно потребовать, чтобы значения в столбце с датами лежали в определенном диапазоне. Контрольные ограничения входят в число наиболее мощных инструментов, обеспечивающих чистоту хранимых данных.

Команда создания контрольного ограничения на столбец существующей таблицы имеет следующий синтаксис:

```
ALTER TABLE имя_таблицы ADD CONSTRAINT [имя_ограничения]
```

CHECK (имя_столбца условие) ;

Параметр *имя_таблицы*, разумеется, идентифицирует таблицу, столбцы которой должны контролироваться. Необязательный параметр *имя_ограничения* позволяет вам самостоятельно присвоить имя ограничению. Если опустить этот параметр, выберет имя автоматически, причем оно не будет нести какой-либо полезной информации. Лучше присваивать имя вручную, чтобы при нарушении ограничения в сообщении об ошибке стояло именно заданное вами имя – предположительно, более информативное, чем присвоенное по умолчанию. Параметр *имя_столбца* идентифицирует столбец, к которому относится ограничение. Параметр *условие* определяет, каким требованиям должны удовлетворять данные, чтобы попытка их вставки в таблицу была успешной. Все вместе эти параметры выглядят точно так же, как и условие в конструкции WHERE оператора SELECT.

Разрешение и запрещение существующих ограничений

Бывают ситуации, когда желательно временно запретить ограничения, а впоследствии снова их разрешить. Например, при загрузке данных из внешнего источника вы можете заранее знать, что некоторые из них не удовлетворяют определенному ограничению, но предпочесть устранение ошибок средствами. В подобных случаях следует использовать команду ALTER TABLE, чтобы временно запретить ограничение, не удаляя его. Затем вы сможете загрузить данные с нежелательными значениями, исправить их и снова разрешить ограничение.

Синтаксис этой команды выглядит следующим образом

```
ALTER TABLE имя_таблицы DISABLE CONSTRAINT
имя_ограничения;
```

Команда разрешения ограничения имеет похожий синтаксис

```
ALTER TABLE имя_таблицы ENABLE CONSTRAINT
имя_ограничения;
```

Изменение и удаление существующих ограничений

Синтаксис команды, разрешающей столбцу принимать null-значения, имеет следующий вид

```
ALTER TABLE имя_таблицы MODIFY (имя_столбца NULL);
```

Если нужно, чтобы столбец перестал принимать null-значения, используется следующий синтаксис

```
ALTER TABLE имя_таблицы MODIFY (имя_столбца NOT NULL);
```

Чтобы удалить ограничение, используется следующий синтаксис

```
ALTER TABLE имя_таблицы DROP CONSTRAINT имя_ограничения;
```

Методика и порядок выполнения работы

Методика выполнения работы

1. Создадим рабочую таблицу person и добавим несколько записей, с помощью приведенного ниже кода:

```
CREATE TABLE person(person_code VARCHAR2(3),
first_name VARCHAR2(15),last_name VARCHAR2(20),hiredate
DATE);
```

```
INSERT INTO person
```

```
VALUES ('CCA', 'Charlene', 'Atlas', '01-02-08');
```

```
INSERT INTO person
```

```
VALUES ('CCA', 'Gary', 'Anderson', '15-02-07');
```

```
INSERT INTO person
```

```
VALUES ('CBB', 'Bobby', 'Barkenhagen', '28-02-07');
```

```
INSERT INTO person
```

```
VALUES ('CLB', 'Laren', 'Baxter', '01-04-08');
```

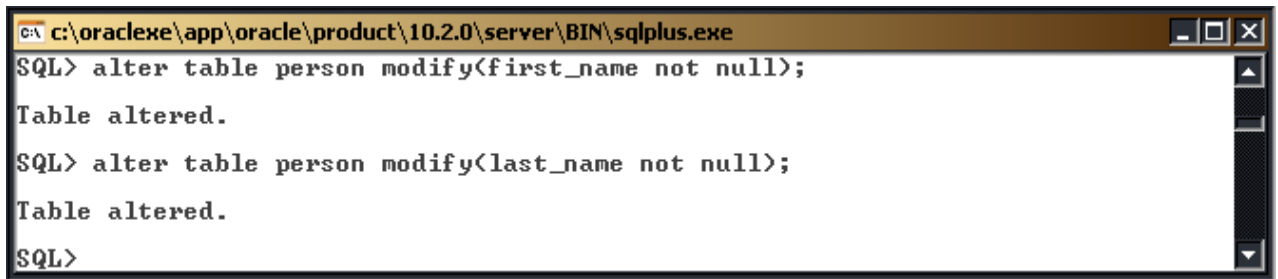
2. Создадим индекс к нашей таблице person показанный на рис. 5.2.

```
CREATE INDEX person_code_index ON person (person_code);
```

3. Применим ограничение NOT NULL к таблице person. Таблица person содержит столбцы first_name и last_name; логично предположить, что без имени и фамилии информация о человеке будет неполной. Чтобы сделать эти столбцы обязательными.

```
ALTER TABLE person MODIFY (first_name NOT NULL);
```

```
ALTER TABLE person MODIFY (last_name NOT NULL);
```



```

C:\oracle\app\oracle\product\10.2.0\server\BIN\sqlplus.exe
SQL> alter table person modify<first_name not null>;
Table altered.
SQL> alter table person modify<last_name not null>;
Table altered.
SQL>

```

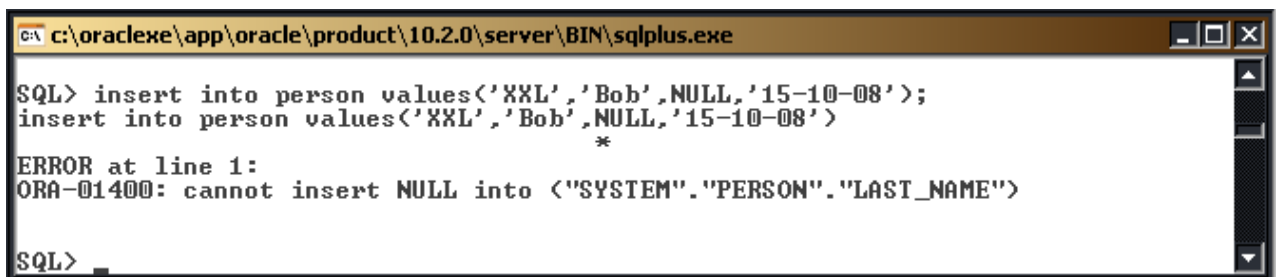
Рисунок 5.2 – Результат выполнения команды ALTER TABLE

4. Теперь протестируем ограничение с помощью приведенного ниже кода (рис. 5.3).

```

INSERT INTO person
VALUES ('XXL', 'Bob', NULL, '15-10-08');

```



```

C:\oracle\app\oracle\product\10.2.0\server\BIN\sqlplus.exe
SQL> insert into person values('XXL','Bob',NULL,'15-10-08');
insert into person values('XXL','Bob',NULL,'15-10-08')
*
ERROR at line 1:
ORA-01400: cannot insert NULL into <"SYSTEM"."PERSON"."LAST_NAME">
SQL>

```

Рисунок 5.3 – Попытка добавить NULL значение

5. Добавим и протестируем ограничение UNIQUE к нашей таблице

```

ALTER TABLE person ADD CONSTRAINT person_un
UNIQUE (first_name, last_name, hiredate);
INSERT INTO person
VALUES ('XXL', 'Bob', 'Bob', '15-10-08');
INSERT INTO person
VALUES ('LLL', 'Bob', 'Bob', '15-10-08');

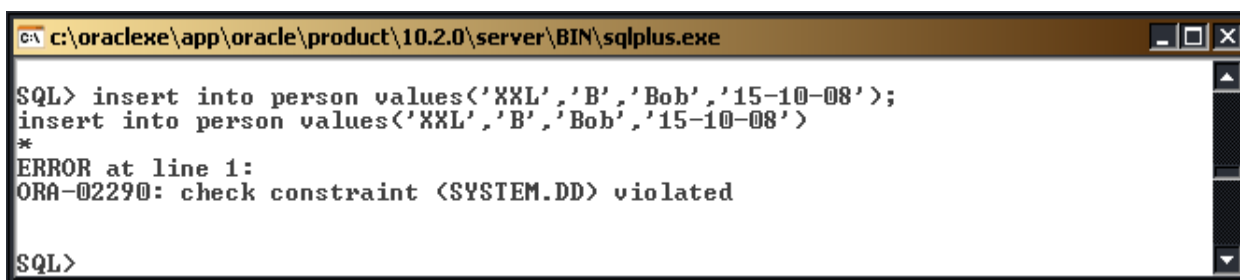
```

6. Добавим и протестируем ограничение CHECK к таблице person (рис. 5.4).

```

ALTER TABLE person ADD CONSTRAINT dd CHECK(last_name !=
'Bob');
INSERT INTO person VALUES('XXL', 'B', 'Bob', '15-10-08');

```



```

C:\oracle\app\oracle\product\10.2.0\server\BIN\sqlplus.exe
SQL> insert into person values('XXL','B','Bob','15-10-08');
insert into person values('XXL','B','Bob','15-10-08')
*
ERROR at line 1:
ORA-02290: check constraint (SYSTEM.DD) violated

SQL>

```

Рисунок 5.4 – Результат использования ограничения CHECK

7. Удалим ограничение dd.

```
ALTER TABLE person DROP CONSTRAINT dd;
```

Порядок выполнения работы

1. Создайте рабочую таблицу и добавьте несколько записей.
2. Создайте индекс к вашей таблице.
3. Примените и протестируйте ограничение NOT NULL к вашей таблице.
5. Добавьте и протестируйте ограничение UNIQUE.
7. Добавьте и протестируйте ограничение CHECK.
8. Удалите созданные ограничения.

Содержание отчета и его форма

Отчет должен содержать

- название лабораторной работы;
- цель работы;
- порядок выполнения работы;
- результаты выполнения;
- вывод о результатах создания и использования индексов

БД и ограничений.

Контрольные вопросы и защита работы

Контрольные вопросы

1. Что такое индекс?
2. Команда создания индекса.
3. Что такое ограничения?

4. Ограничение NOT NULL, UNIQUE и CHECK.

5. Синтаксис команд для изменения и удаления существующих ограничений.

Защита работы

Защита работы заключается в

- выполнении заданий раздела «Методика и порядок выполнения работы»;
- ответах на контрольные вопросы;
- предоставлении отчета.

ЛАБОРАТОРНАЯ РАБОТА №6

Связи между таблицами

Цель и содержание

Цель работы: приобрести навыки создания связанных таблиц, написания операторов SELECT, отображающих данные из нескольких таблиц.

Теоретическое обоснование

Все данные представлены в виде набора простых таблиц (двумерных массивов), разбитых на строки и столбцы, на пересечении которых расположены данные. У каждого столбца есть имя, уникальное в пределах таблицы, причем все значения в одном столбце – однородны, т. е. имеют один тип. Каждая строка имеет одно или несколько полей, набор значений в которых уникален в пределах таблицы. Этот набор называется *первичным ключом* (primary key) и

служит для идентификации строки. Этот принцип не допускает, в частности, хранение в таблице совершенно одинаковых строк. Имя таблицы, имя столбца и первичный ключ однозначно определяют хранимый элемент данных. Строки в реляционной базе данных не упорядочены. Упорядочивание производится в момент формирования ответа на запрос. Запросы к базе данных возвращают результат в виде таблиц, которые также могут выступать как объект для новых запросов.

Так, например, простая таблица – справочник стран. Назовем ее COUNTRIES (табл. 6.1).

Таблица 6.1 – Справочник стран

Справочник стран COUNTRIES	
ID	NAME
1	Россия
2	Франция
3	Марокко
4	Япония

В таблице COUNTRIES всего два столбца:

- ID – код страны;
- NAME – ее название.

Столбец ID служит *первичным ключом* таблицы, а столбец NAME содержит требуемую информацию, которую далее и будем стремиться извлекать запросами. Все данные столбца ID – целочисленные, столбца NAME – содержат текстовую информацию.

Для того, чтобы база данных стала реляционной, одних *данных* недостаточно. Необходимы также и *связи* (relations). Для связи между таблицами служит так называемый *внешний ключ* (foreign key). Название довольно точно выражает его суть. Если в таблице А есть столбец для хранения первичного ключа таблицы В, то такой столбец и называется внешним ключом. Первичные и

внешние ключи устанавливают связи между таблицами, превращая набор таблиц в цельную конструкцию – реляционную базу данных.

Например. Допустим, что создана еще одна таблица – справочник товаров GOODS (табл. 6.2).

Таблица 6.2 – Справочник товаров

Товарный справочник GOODS				
ID	NAME	PRICE	UNIT	COUNTRY
1	Яблоки	50.00	кг	Россия
2	Груши	60.40	кг	Франция
3	Апельсины	40.00	кг	Марокко
4	Макароны	21.00	шт	Франция
5	Кефир	25.30	шт	Россия
6	Молоко	30.50	шт	Россия

Ее колонки: ID – первичный ключ, NAME – название товара, PRICE – его цена, UNIT – краткое название единицы измерения, COUNTRY – название страны-производителя.

Правильно ли построена такая таблица? Казалось бы, всем упоминавшимся выше принципам она удовлетворяет: уникальные имена столбцов с однородными данными, строки с уникальным первичным ключом. Тем не менее, построена она непрофессионально. Требуется нормализация таблиц. Суть в том, чтобы избежать избыточности в хранении данных. Этого можно достигнуть путем выделения их в отдельные таблицы.

Рассмотрим таблицу GOODS. Представьте, что в дальнейшем придется изменить название какой-нибудь страны. Такое случается часто. Бирма когда-то меняла свое название на Мьянму, Польша – на Польскую Республику. Хочется ли нам менять огромное количество строк во всех таблицах, где эти страны упоминаются? Представим также, что возникнет необходимость отобрать запросом весь штучный товар. Можно ли быть уверенным в том, что оператор всюду ввел эту аббревиатуру правильно и одинаково? Скорее всего, окажется, что в таблице встречаются все мыслимые вариации: «шт», «Шт», «шт.», «штук» и «штуки».

Выходом из данной ситуации будет выделение из нее двух других таблиц: справочника стран (COUNTRIES) и справочника единиц измерений (UNITS) (рис. 6.3).

Таблица 6.3 – Справочник единиц измерений (UNITS)

Справочник единиц измерения UNITS		
ID	NAME	SHORT_NAME
1	Штуки	шт
2	Килограммы	кг

Справочник товаров GOODS будет теперь выглядеть совершенно по-другому (табл. 6.4).

Таблица 6.4 – Справочник товаров GOODS

Товарный справочник GOODS после нормализации				
ID	NAME	PRICE	UNIT_ID	COUNTRY_ID
1	Яблоки	50.00	2	1
2	Груши	60.40	2	2
3	Апельсины	40.00	2	3
4	Макароны	21.00	1	2
5	Кефир	25.30	1	1
6	Молоко	30.50	1	1

Вместо столбцов с названиями единиц измерения и стран появились столбцы UNIT_ID и COUNTRY_ID с кодами, отсылающими нас к другим таблицам. Это и есть внешние ключи. Что означает значение 2 в столбце UNIT_ID? Оно означает, что интересующая нас информация по единице измерения находится той строке таблицы UNITS, где ID = 2. Достаточно посмотреть в этот справочник, чтобы убедиться, что называется эта единица полностью «штуки», а кратко – «шт».

Создание первичного ключа

Команда, позволяющая определить, какие столбцы существующей таблицы будут использоваться в качестве первичного ключа, имеет следующий синтаксис

```
ALTER TABLE имя_таблицы ADD PRIMARY KEY
(имя_столбца_1, имя_столбца_2, ...);
```

Когда вы создаете первичный ключ, автоматически индексирует его столбцы. Это немного замедляет ввод данных, но ускоряет выполнение лю-

бых запросов, у которых в конструкции WHERE на первом месте указаны столбцы первичного ключа.

Создание ограничения внешнего ключа

Первичные и внешние ключи – это физические компоненты, без которых невозможно установить связь между таблицами. Однако сами по себе они не обеспечивают реляционной целостности. Даже если столбцы первичного и внешнего ключей имеют одинаковые имена и типы данных, не считает их связанными, пока вы не объявите об этом. Последний шаг имеет решающее значение: необходимо определить ограничение на подчиненную таблицу, которое будет использоваться для сверки значений, вводимых в столбец внешнего ключа, со значениями первичного ключа главной таблицы. Без такого ограничения пользователь сможет ввести во внешний ключ подчиненной таблицы значения, не существующие в главной таблице.

Команда создания ограничения внешнего ключа имеет следующий синтаксис

```
ALTER TABLE имя_подчиненной_таблицы
ADD CONSTRAINT имя_ограничения
FOREIGN KEY (имена_столбцов_подчиненной_таблицы)
REFERENCES имя_главной_таблицы;
```

Написание операторов SELECT, отображающих данные из нескольких таблиц

Зная, как создавать связи между таблицами, необходимо научиться соединять содержащуюся в них информацию.

Чтобы вернуть данным из таблицы GOODS их вид до нормализации, т. е. получить наглядную таблицу, в которой во всех колонках расположены не номера, а нормальные текстовые названия, используется оператор SELECT. Оператор SELECT, извлекающий данные из двух таблиц, имеет следующий синтаксис

```
SELECT      имя_таблицы_1.имя_столбца,
            имя_таблицы_2.имя_столбца
```

```
FROM      имя_таблицы_1, имя_таблицы_2
WHERE      имя_главной_таблицы.первичный_ключ =
           имя_подчиненной_таблицы.внешний_ключ;
```

Описывая в операторе SELECT связь «главный/подчиненный», указываем как требуется связывать информацию из главной таблицы с информацией из подчиненной таблицы.

Написание команды, которая комбинирует данные из более чем одной таблицы в единый список, называется созданием *соединения* (join). Самое важное, что следует помнить о соединении, заключается в следующем: *необходимо включать в оператор SELECT конструкцию WHERE, описывающую, как связаны друг с другом первичный ключ главной таблицы и внешний ключ подчиненной таблицы*. Если этого не сделать, соединит каждую запись главной таблицы с каждой записью подчиненной таблицы, результатом чего может стать очень длинный список. Подобная ошибка может резко замедлить работу базы данных на время обработки неправильно написанного запроса, поэтому ее не следует допускать.

Внешние соединения

У последнего запроса есть два недостатка. Во-первых, значения кодов страны или единицы измерения, указанные в основной таблице, могут отсутствовать в справочниках. В этом случае соответствующие товары просто исчезнут из результатов запроса. Во-вторых, в силу определенных причин такой запрос может выполняться не вполне оптимально, иначе говоря – медленно.

Методика и порядок выполнения работы

Методика выполнения работы

1. Создадим три новых таблицы

```
CREATE TABLE goods (
    idgoods INT, name VARCHAR(100) NOT NULL UNIQUE,
    price NUMBER(10,2) NOT NULL);
CREATE TABLE units ( idunits INT,
    name VARCHAR(100) NOT NULL UNIQUE,
```

```

short_name VARCHAR(100) NOT NULL UNIQUE);
CREATE TABLE countries (idcountries INT,
name VARCHAR(100) NOT NULL UNIQUE);

```

2. Добавим первичный ключ по столбцу idgoods.

```
ALTER TABLE goods ADD PRIMARY KEY (idgoods);
```

3. Тем же способом создадим первичные ключи в двух других таблицах

```

ALTER TABLE units ADD PRIMARY KEY (idunits);
ALTER TABLE countries ADD PRIMARY KEY (idcountries);

```

4. В качестве внешних ключей добавим в таблицу goods два столбца

```

ALTER TABLE goods ADD units_idunits INT;
ALTER TABLE goods ADD countries_idcountries INT;

```

5. Протестируем все три таблицы на возможность вставки неверных значений (рис. 6.1).

```

INSERT INTO goods VALUES (1, 'ЯБЛОКИ', 1.44, 1, 1);
SELECT * FROM goods;
SELECT * FROM units;
SELECT * FROM countries;

```

The screenshot shows a terminal window titled "D:\OracleXE\app\oracle\product\10.2.0\server\BIN\sqlplus.exe". The SQL prompt shows the following commands and results:

```

SQL> INSERT into goods values (1,'Apple',1.44,1,1);
1 row created.
SQL> select * from goods;

```

IDGOODS	NAME	PRICE	UNITS_IDUNITS	COUNTRIES_IDCOUNTRIES
1	Apple	1.44	1	1

```

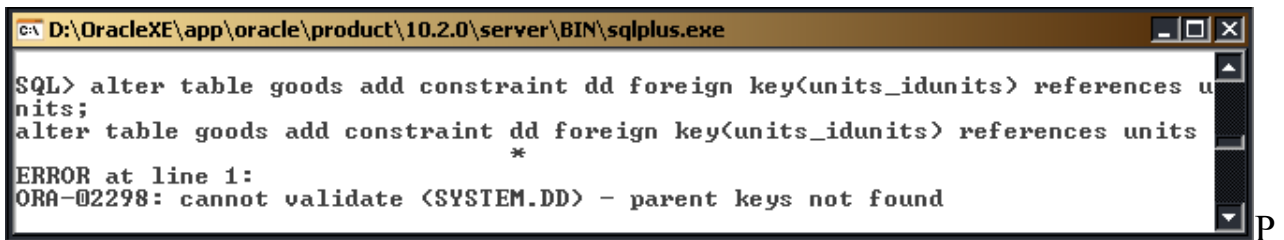
SQL> select * from units;
no rows selected
SQL> select * from countries;
no rows selected
SQL>

```

Рисунок 6.1 – Вывод данных таблиц

6. Создадим ограничение внешнего ключа в таблице units (рис. 6.2)

```
ALTER TABLE goods ADD CONSTRAINT dd
FOREIGN KEY(units_idunits) REFERENCES units;
```

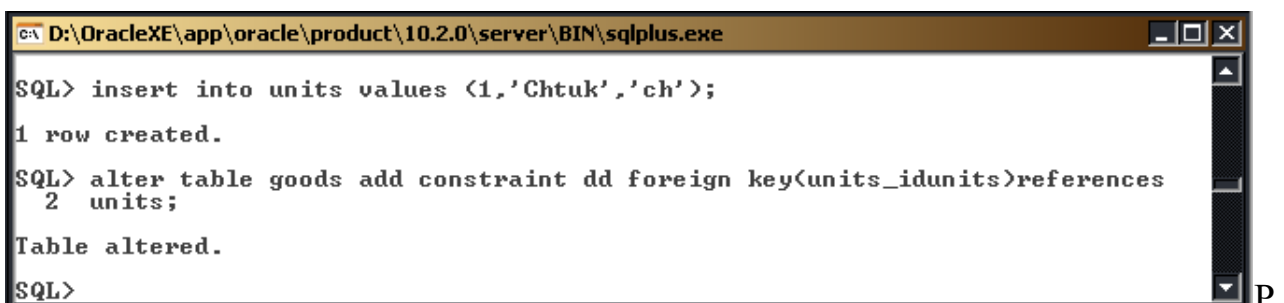


исунок 6.2 – Результат применения ограничения для таблицы goods

Исправим некорректные данные в таблицах и снова создадим ограничение (рис. 6.3)

```
INSERT INTO units VALUES (1, 'Chtuk', 'ch');

ALTER TABLE goods ADD CONSTRAINT dd
FOREIGN KEY(units_idunits) REFERENCES units;
```



исунок 6.3 – Результат добавления ограничения для внешнего ключа

7. Протестируем новое ограничение, попробуйте ввести следующую запись (рис. 6.4)

```
INSERT INTO goods VALUES (2, 'APPLE', 1.44, 12, 12)
```

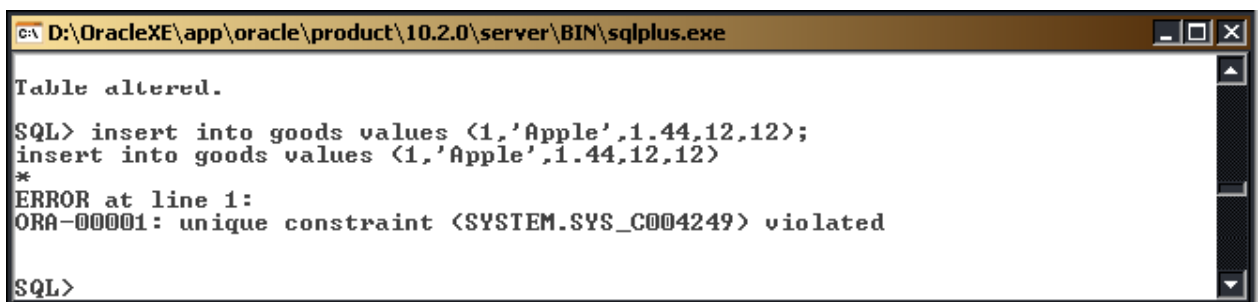


Рисунок 6.4 – Попытка добавить запись со ссылкой на таблицу units

8. Создадим ограничение внешнего ключа для таблицы countries, проверяющее таблицу goods перед вставкой значений в столбец countries_idcountries.

```
INSERT INTO countries VALUES (1, 'Russia');
```

```
ALTER TABLE goods ADD CONSTRAINT ddd
FOREIGN KEY(countries_idcountries) REFERENCES countries;
```

9. Напишем оператор SELECT, отображающих данные из нескольких таблиц (рис. 6.5)

```
SELECT goods.idcountries, goods.name, goods.price,
       units.name, countries.name FROM units, countries,
       goods WHERE units.idunits = goods.units_idunits AND
               countries.idcountries =
               goods.countries_idcountries;
```

The screenshot shows a window titled "D:\OracleXE\app\oracle\product\10.2.0\server\BIN\sqlplus.exe". The command prompt shows the following sequence of commands and output:

```
SQL> edit
Wrote file afiedt.buf

1* alter table goods modify short_name varchar(10)
SQL> SELECT goods.idgoods, goods.name, goods.price
2  , units.name, countries.name
3  FROM units, countries, goods
4  WHERE units.idunits=goods.units_idunits
5  AND countries.idcountries=goods.countries_idcountries;
```

The output is displayed as a table with three columns: IDGOODS, NAME, and PRICE. The data row shows IDGOODS 1, NAME Apple, and PRICE 1.44. The table is followed by a blank line and then the word "Russia" which is not part of the table output.

IDGOODS	NAME	PRICE
1	Apple	1.44

SQL>

исунок 6.5 – Оператор SELECT, извлекающий данные из трех таблиц

10. Воспользуемся альтернативным способом присоединения вспомогательных таблиц, используя ключевое слово JOIN.

```
SELECT goods.idgoods, goods.name, goods.price,
       units.name, countries.name FROM goods INNER JOIN units ON
       units.idunits = goods.units_idunits INNER OUTER JOIN
       countries ON countries.idcountries =
       goods.countries_idcountries;
```

Порядок выполнения работы

1. Создайте три новых таблицы.
2. Добавьте первичные ключи к созданным таблицам.

3. Протестируйте все три таблицы на возможность вставки неверных значений.
4. Добавьте необходимые внешние ключи.
5. Попробуйте создать ограничение внешнего ключа в главной таблице.
6. Протестируйте новое ограничение.
7. Создайте ограничение внешнего ключа для таблиц, проверяющее главную таблицу.
8. Напишем оператор SELECT, отображающих данные из нескольких таблиц
9. Воспользуемся альтернативным способом присоединения вспомогательных таблиц, используя ключевое слово JOIN.

Содержание отчета и его форма

Отчет должен содержать

- название лабораторной работы;
- цель работы;
- порядок выполнения работы;
- результаты выполнения;
- вывод.

Контрольные вопросы и защита работы

Контрольные вопросы

1. Что представляет собой связь?
2. Для чего применяется первичный ключ?
3. С какой целью создается внешний ключ?
4. Могут ли значения первичного и внешнего ключа быть различными?
3. Синтаксис для создания ограничения внешнего ключа.
4. Синтаксис оператора SELECT, для отображения данных из нескольких таблиц.
5. Что обозначает ключевое слово JOIN.

Защита работы

Защита работы заключается в

- выполнении заданий раздела «Методика и порядок выполнения работы»;
- ответах на контрольные вопросы;
- предоставлении отчета.

ЛАБОРАТОРНАЯ РАБОТА №7

Написание подзапросов

Цель и содержание

Цель работы: приобрести навыки создания и использования подзапросов.

Теоретическое обоснование

Подзапрос – это обычный запрос SELECT, вложенный в оператор SELECT, UPDATE или DELETE. Он используется в качестве источника данных для раздела FROM или WHERE родительского оператора.

Подзапрос может содержать внутри себя другие подзапросы. В документации утверждается, что число уровней вложения не ограничено. «Не ограничено» обычно следует интерпретировать как «зависит от количества доступных ресурсов компьютера, но в любом случае это наверняка больше, чем вам когда-либо потребуется».

Типичные проблемы, решаемые с помощью подзапросов

Подзапрос применяется в тех случаях, когда для выполнения одного запроса требуется предварительно выполнить другой. Например, чтобы определить, какие товары продаются лучше, чем в среднем, нужно сначала найти это

«среднее». Чтобы выяснить, сколько денег заработал старший продавец, нужно знать, кто является старшим продавцом. При необходимости установить, какие товары продаются хуже, чем в прошлом году, требуется знать, как они продавались год назад. Во всех этих ситуациях необходимая информация может быть получена с помощью подзапроса.

Ключевой характеристикой только что написанных подзапросов является то, что они возвращают единственное значение. Например, когда вы пишете подзапрос для определения цены Apple, может быть возвращено только одно значение, поскольку для хранения этой цены в таблице товаров отведен только один столбец. В этом случае в конструкции WHERE родительского оператора можно ставить знак равенства. Подзапрос такого типа называется *однострочным подзапросом* (single-row sub-query), поскольку он может возвращать только одну строку результатов. При этом, родительский оператор может возвращать любое число строк на основе одного результата подзапроса. Подзапрос и родительский оператор могут ссылаться на совершенно разные таблицы.

Многострочные подзапросы

Многострочный подзапрос (multirowsubquery) – это подзапрос, который может возвращать более одной строки результатов. Для таких подзапросов нельзя выполнять сравнение с помощью знака равенства; необходимо использовать функцию IN. Подзапросы можно использовать также в операторах UPDATE и DELETE.

Методика и порядок выполнения работы

Методика выполнения работы

1. Создадим оператор SELECT, возвращающий все записи о товарах с ценой, равной цене Apple. Необходимо не только подставить значение 25 в конструкцию WHERE, а определить – какова цена Apple.

Введем следующую команду и сравним результаты с теми, что показаны на рис. 7.1

```
SELECT * FROM product_n WHERE product_price = (
    SELECT product_price FROM product_n
```

```
WHERE product_name = 'Apple'
);
```

```

C:\oracle\app\oracle\product\10.2.0\server\BIN\sqlplus.exe
SQL> edit
Wrote file afiedt.buf

 1 SELECT *
 2 FROM   product_n WHERE   product_price= (
 3 SELECT product_price
 4 FROM   product_n
 5 WHERE  product_name  =  'Apple')
SQL> /

```

PRODUCT_NAME	PRODUCT_PRICE	HAND	LSDATE
Apple	25	10000	
Apple full	25		

```

SQL>

```

Рисунок 7.1 – Вывод всех записи о товарах с ценой, равной цене Apple

Следующий пример однострочного подзапроса, допустим, требуется получить список самых дорогих товаров. Для этого можно создать подзапрос, определяющий среднюю цену товара (рис. 7.2)

```
SELECT * FROM product_n WHERE product_price >
(SELECT SUM(product_price)/COUNT(*) FROM product_n);
```

При этом необходимо учесть, что количество товара не может равняться нулю.

```

C:\oracle\app\oracle\product\10.2.0\server\BIN\sqlplus.exe
SQL> select * from product_n where product_price >(select sum(product_price)/count(*) from product_n);

```

PRODUCT_NAME	PRODUCT_PRICE	HAND	LSDATE
Wood	99	1	
Chrome full	75	1000	

```

SQL>

```

Рисунок 7.2 – Список самых дорогих товаров

2. Создадим еще одну таблицу и добавим некоторые данные:

```
CREATE TABLE ch (product_name VARCHAR2(10));
INSERT INTO ch VALUES ('Apple');
INSERT INTO ch VALUES ('Wood');
```

Допустим, необходимо узнать, какие товары не продаются. Для этого по-

лучим с помощью подзапроса список всех названий товаров из таблицы ch, а затем передадим его родительскому оператору, чтобы исключить записи об этих товарах из выходных данных. Введем следующий код и сравним результаты с показанными на рис. 7.3:

```
SELECT * FROM ch;

SELECT * FROM product_n
WHERE product_name NOT IN
      (SELECT DISTINCT product_NAME FROM ch)
ORDER BY product_name;
```

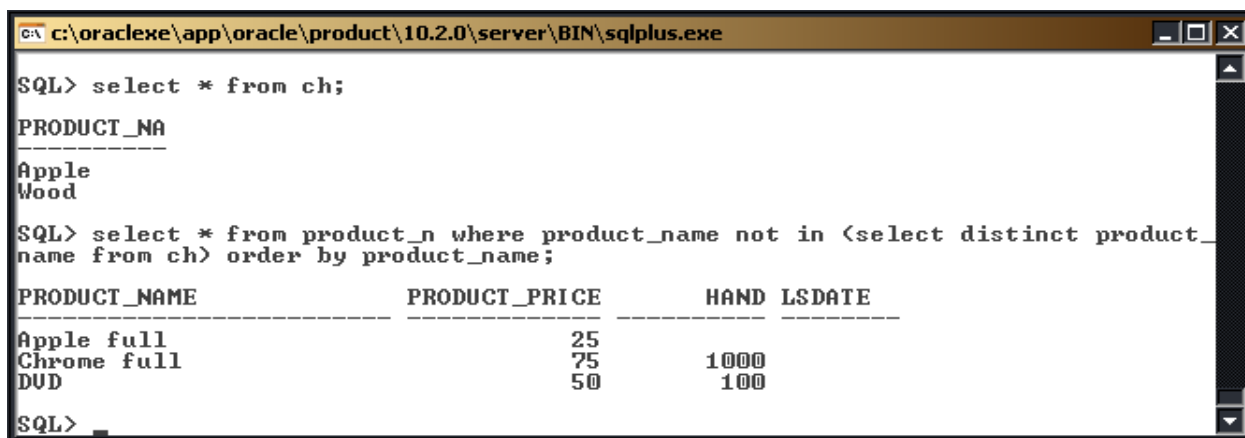


Рисунок 7.3 – Результат выполнения многострочного подзапроса

3. Теперь предположим, что дано указание снизить на 10% цены всех товаров, не пользующихся спросом. Это можно сделать единственной командой UPDATE, поместив в ее конструкцию WHERE подзапрос, определяющий, какие товары не продавались (рис. 7.4)

```
SELECT * FROM product_n;

UPDATE product_n
SET product_price = product_price * .9
WHERE product_name NOT IN (
      SELECT DISTINCT product_name FROM ch);

SELECT * FROM product_n;
```

```

C:\oracle\app\oracle\product\10.2.0\server\BIN\sqlplus.exe
SQL> select * from product_n;

PRODUCT_NAME          PRODUCT_PRICE      HAND  LSDATE
-----
Wood                   99                 1
DUD                    50                100
Apple                  25               10000
Chrome full            75                1000
Apple full             25

SQL> update product_n set product_price=product_price * .9 where product_name not
in (select distinct product_name from ch);

3 rows updated.

SQL> select * from product_n;

PRODUCT_NAME          PRODUCT_PRICE      HAND  LSDATE
-----
Wood                   99                 1
DUD                    45                100
Apple                  25               10000
Chrome full            67,5              1000
Apple full             22,5
SQL>

```

Рисунок 7.4 – Результат выполнения многострочного подзапроса в UPDATE

Порядок выполнения работы

1. Создайте оператор SELECT использующий однострочный подзапрос.
2. Создайте два оператора SELECT использующих многострочные подзапросы.

Содержание отчета и его форма

Отчет должен содержать

- название лабораторной работы;
- цель работы;
- порядок выполнения работы;
- результаты выполнения;
- вывод о результатах создания и использования подза-

просов.

Контрольные вопросы и защита работы

Контрольные вопросы

1. Что такое подзапрос?
2. Какие проблемы можно решить с помощью подзапросов?
3. Что такое многострочный подзапрос?
4. Функция IN.
5. В каких операторах можно использовать подзапросы?

Защита работы

Защита работы заключается в

- выполнении заданий раздела «Методика и порядок выполнения работы»;
- ответах на контрольные вопросы;
- предоставлении отчета.

ЛАБОРАТОРНАЯ РАБОТА №8

Перенос данных между таблицами

Цель и содержание

Цель работы: приобрести навыки переноса данных из одной таблицы в другую, а также объединения нескольких таблиц в одну.

Теоретическое обоснование

Типичной задачей, возникающей при работе с SQL, является перенос данных из существующей системы в новую. Иногда существующая система просто заменяется новой. В других случаях необходимо отображать и переносить в свою систему данные, полученные из внешнего источника. Зачастую исходные данные должны модифицироваться до занесения в новые таблицы, что может потребовать использования таких функций, как UPPER, LOWER, SUBSTR, INSTR, DECODE.

Реляционные базы данных обеспечивают наиболее эффективное хранение информации, но ее извлечение из реляционных таблиц может занимать довольно много времени, поскольку таблицы требуется соединять. В некоторых приложениях имеет смысл скопировать реляционные данные из многих таблиц в одну таблицу плоского файла, где все соединения уже выполнены. Таблица

плоского файла занимает больше места, чем исходные реляционные таблицы, но доступ к ней может осуществляться быстрее, поскольку отпадает необходимость в соединениях.

Перенос данных с помощью INSERT

Распространенным способом переноса данных является использование команды INSERT с подзапросом, извлекающим вставляемые данные из другой таблицы. Ее синтаксис выглядит следующим образом:

```
INSERT INTO имя_таблицы (оператор SELECT);
```

Оператор SELECT – это любая команда SELECT, которая выдает нужные вам данные со структурой, соответствующей структуре таблицы назначения. Ниже приведен пример использования этого синтаксиса.

Создание новой таблицы на основе уже существующей

В рассмотренном выше методе копирования данных из одной таблицы в другую предполагалось, что таблица назначения уже существует. Это вполне подходит для ежедневного добавления записей в таблицу назначения, но вместе с тем есть и более простой способ создания такой таблицы. Необходимо использовать разновидность команды CREATE TABLE со следующим синтаксисом:

```
CREATE TABLE имя_новой_таблицы AS оператор SELECT;
```

В данном случае *оператор SELECT* будет тем же самым оператором SELECT, с помощью которого заполняли первую таблицу назначения.

Методика и порядок выполнения работы

Методика выполнения работы

1. Для переноса данных с помощью INSERT создадим таблицу назначения для демонстрации выше описанной техники, введите следующую команду:

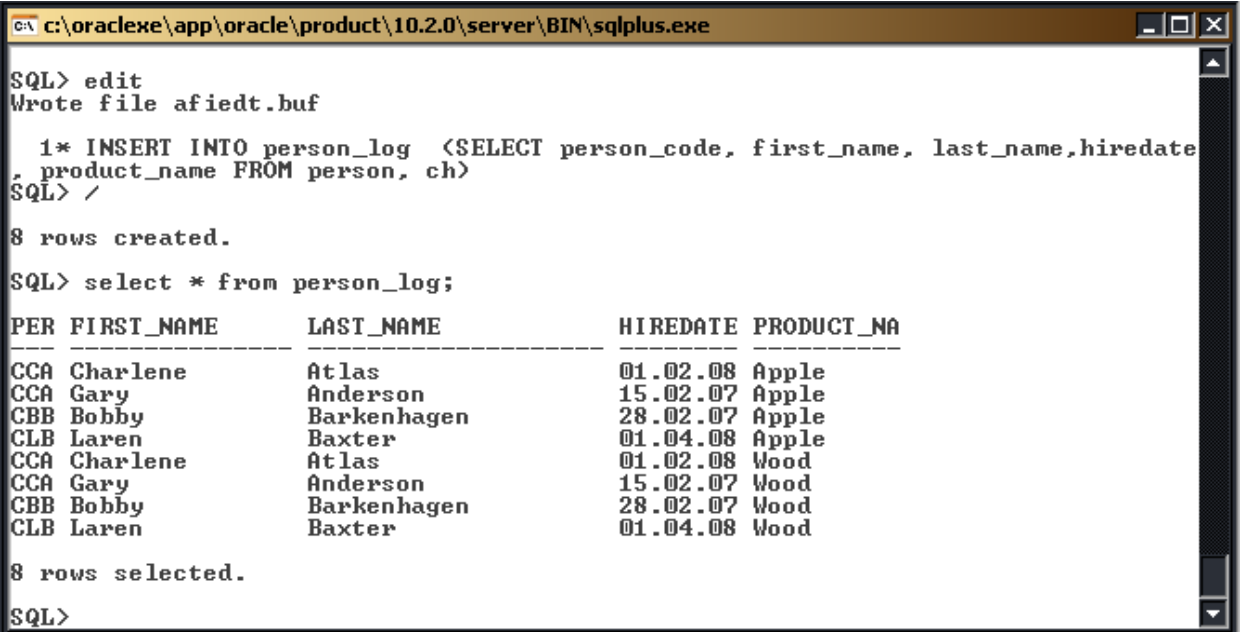
```
CREATE TABLE person_log (
    person_code VARCHAR2(3),
    first_name VARCHAR2(15),
    last_name VARCHAR2(20),
    hiredate DATE,
```

```
product_name VARCHAR2(10));
```

Таблица `person_log` – это «плоское» представление наиболее важной информации о каждой покупке: даты продажи, названия товара, а также полного имени продавца.

2. Теперь, когда есть таблица плоского файла для хранения записей, подлежащих анализу, заполним ее данными. Это будет сделано при помощи команды `INSERT`, соединяющей записи из таблиц `PERSON`, `PRODUCT_N` (рис. 8.1).

```
INSERT INTO person_log (SELECT person_code, first_name,
    last_name, hiredate, product_name FROM person, ch);
```



```

SQL> edit
Wrote file afiedt.buf

  1* INSERT INTO person_log (SELECT person_code, first_name, last_name, hiredate
, product_name FROM person, ch)
SQL> /

8 rows created.

SQL> select * from person_log;

PER FIRST_NAME      LAST_NAME      HIREDATE  PRODUCT_NA
-----
CCA Charlene        Atlas          01.02.08  Apple
CCA Gary            Anderson       15.02.07  Apple
CBB Bobby           Barkenhagen    28.02.07  Apple
CLB Laren           Baxter         01.04.08  Apple
CCA Charlene        Atlas          01.02.08  Wood
CCA Gary            Anderson       15.02.07  Wood
CBB Bobby           Barkenhagen    28.02.07  Wood
CLB Laren           Baxter         01.04.08  Wood

8 rows selected.

SQL>

```

Рисунок 8.1 – Результат переноса данных из двух таблиц

3. Для создания новой таблицы на основе уже существующей введем показанный ниже код, чтобы создать вторую таблицу назначения (рис. 8.2).

```
CREATE TABLE person_log2 AS SELECT person_code,
    first_name, last_name, hiredate, product_name FROM per-
        son, ch;
```

```

C:\oracle\app\oracle\product\10.2.0\server\BIN\sqlplus.exe
SQL> edit
Wrote file afiedt.buf

1* CREATE TABLE person_log2 AS SELECT person_code, first_name, last_name, hire
date, product_name FROM person, ch
SQL> /
Table created.
SQL> select * from person_log2;
PER FIRST_NAME      LAST_NAME      HIREDATE  PRODUCT_NA
-----
CCA Charlene        Atlas          01.02.08  Apple
CCA Gary            Anderson      15.02.07  Apple
CBB Bobby           Barkenhagen   28.02.07  Apple
CLB Laren            Baxter        01.04.08  Apple
CCA Charlene        Atlas          01.02.08  Wood
CCA Gary            Anderson      15.02.07  Wood
CBB Bobby           Barkenhagen   28.02.07  Wood
CLB Laren            Baxter        01.04.08  Wood
8 rows selected.
SQL>

```

Рисунок 8.2 – Перенос данных с помощью CREATE TABLE

Порядок выполнения работы

1. Создайте таблицу назначения.
2. При помощи команды INSERT, соедините записи из двух таблиц.
3. Создайте новую таблицу на основе существующей.

Содержание отчета и его форма

Отчет должен содержать

- название лабораторной работы;
- цель работы;
- порядок выполнения работы;
- результаты выполнения;
- вывод о результате переноса данных из одной таблицы

в другую, а также объединения нескольких таблиц в одну.

Контрольные вопросы и защита работы

Контрольные вопросы

1. Перенос данных с помощью команды INSERT.
2. Создание новой таблицы на основе существующей.

Защита работы

Защита работы заключается в

- выполнении заданий раздела «Методика и порядок выполнения работы»;
- ответах на контрольные вопросы; предоставлении отчета.

ЛАБОРАТОРНАЯ РАБОТА №9

Представления

Цель и содержание

Цель работы: приобрести навыки создания и использования представлений.

Теоретическое обоснование

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

Представления широко применяются для соединения данных из двух и

более таблиц и выдачи их пользователям в виде одного легко читаемого списка. Упрощая процесс выборки записей до такой степени, когда пользователям не нужно знать, как соединяются таблицы, вы делаете данные доступными для большего числа людей.

С помощью представлений удобно поддерживать безопасность, поскольку они позволяют ограничивать диапазон строк и столбцов, возвращаемых пользователям. Если есть необходимость в том, чтобы пользователи видели столбец с зарплатой из таблицы личных данных, не следует включать его в определение представления. Для пользователей представления этот столбец не будет существовать. То же самое справедливо и для строк: добавив в представление конструкцию WHERE, получим отфильтрованные записи определенным образом.

Наконец, представления могут сделать работу с таблицами более удобной. Поскольку представление – это просто хранимый запрос, вам предоставлена возможность менять как имена столбцов, так и порядок их отображения.

Создание представления

Для создания представления нужно указать только имя представления и оператор SELECT, который будет выполняться при обращении к представлению. Вот соответствующий синтаксис:

```
CREATE OR REPLACE VIEW имя_представления AS оператор SE-
LECT
```

Обратите внимание, что в данной конструкции присутствует новый элемент: OR REPLACE. Он позволяет создавать новое представление даже тогда, когда представление с указанным именем уже существует. При этом, существующее представление перезаписывается.

Удаление представлений

Удалить представление можно также, как и таблицу (но ЭТО действие менее разрушительно, поскольку представление не содержит никаких данных;

самое худшее, к чему может привести случайное удаление представления – это к необходимости создавать его заново). Команда, удаляющая представление, имеет следующий синтаксис:

```
DROP VIEW имя_представления;
```

Изменение определения представления

СУБД не позволяет изменять существующее представление. Единственный способ изменить представление – это удалить его и создать заново. По этой причине все команды создания представлений следует хранить в файлах сценариев. Решив изменить представление, достаточно будет открыть файл сценария, изменить содержащуюся в нем команду CREATE VIEW и запустить сценарий еще раз.

Анализ первых *N* записей

Узнав, насколько легко с помощью SQL можно просмотреть первые 1, 10 или 100 записей, удовлетворяющих любым заданным критериям отбора и сортировки, вы предпочтете вводить соответствующие команды вручную, а не создавать инкапсулирующее их представление.

Каждой записи, возвращаемой в результате обработки любого запроса, динамически присваивается порядковый номер. Первая (или единственная) возвращенная запись получит номер независимо от своего положения в таблице. На эти номера можно ссылаться в конструкции WHERE. Если в операторе SELECT выполняется сортировка результатов, можно добиться того, что СУБД покажет 5, 50 или 500 наиболее важных записей, дополнив оператор конструкции WHERE, ограничивающей количество выводимых строк.

Синтаксис, позволяющий это реализовать, выглядит следующим образом:

```
SELECT имя_столбца_1 [, имя_столбца_2...] FROM имя_таблицы  
WHERE ROWNUM <= нужное_число_записей ORDER BY стол-  
бец_сортировки;
```

Методика и порядок выполнения работы

Методика выполнения работы

1. Чтобы увидеть, как работает представление, введем следующие команды и сравним результаты с показанными на рис. 9.1.

```
SELECT * FROM product_n;

CREATE OR REPLACE VIEW pr_n AS

SELECT * FROM product_n WHERE product_name =

'DVD';

SELECT * FROM pr_n;
```

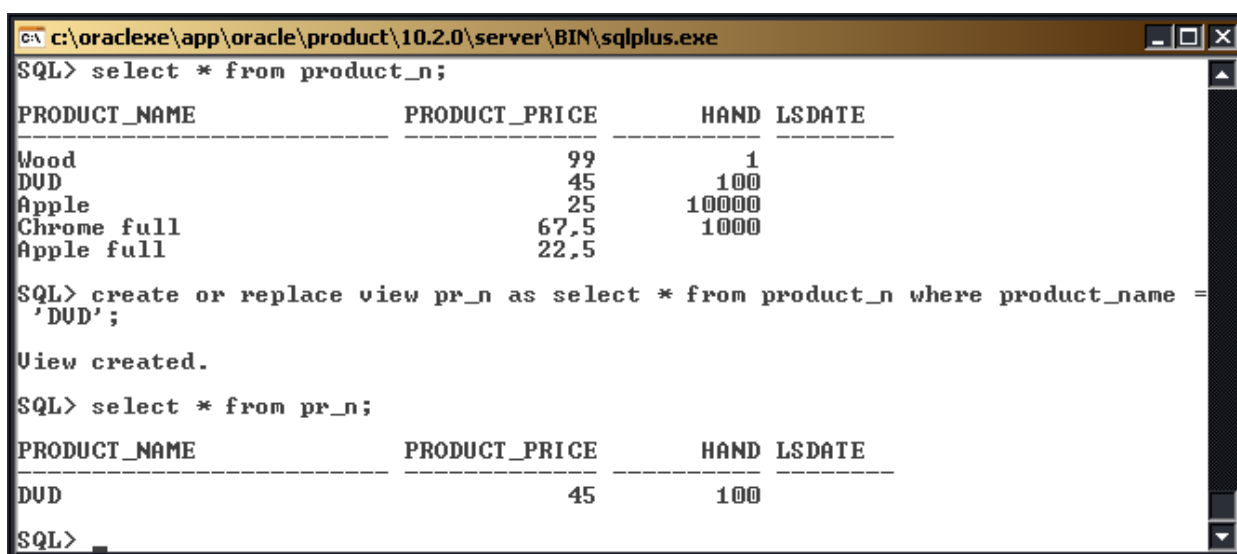


Рисунок 9.1 – Создание представления и его выполнение

2. Удалим только что созданное представление (рис. 9.2).

```
DROP VIEW pr_n;
```



Рисунок 9.2 – Удаление представления pr_n

3. Для вывода первых двух записей введем код

```
SELECT * FROM product_n WHERE ROWNUM<=2 ORDER BY product_name;
```

```

C:\oracle\app\oracle\product\10.2.0\server\BIN\sqlplus.exe
SQL> select * from product_n;
PRODUCT_NAME          PRODUCT_PRICE      HAND  LDATE
-----
Wood                   99                1
DVD                    45               100
Apple                  25              10000
Chrome full            67,5              1000
Apple full             22,5
SQL> select * from product_n where ROWNUM<=2 order by product_name;
PRODUCT_NAME          PRODUCT_PRICE      HAND  LDATE
-----
DVD                    45               100
Wood                   99                1
SQL>

```

Рисунок 9.3 – Вывод первых двух записей с помощью ROWNUM

Порядок выполнения работы

1. Создайте представление к любой таблице.
2. Удалите созданное представление.
3. Выведите несколько первых записей из любой таблицы.

Содержание отчета и его форма

Отчет должен содержать

- название лабораторной работы;
- цель работы;
- порядок выполнения работы;
- результаты выполнения;
- вывод о результате создания и использования представлений.

Контрольные вопросы и защита работы

Контрольные вопросы

1. Что такое представления?
2. Как создаются представления.
3. Как удалить представления.
4. Возможно ли осуществить вывод первых N записей.

Защита работы

Защита работы заключается в

- выполнении заданий раздела «Методика и порядок вы-

полнения работы»;

- ответах на контрольные вопросы;
- предоставлении отчета.

ЛАБОРАТОРНАЯ РАБОТА №10

Последовательности

Цель и содержание

Цель работы: приобрести навыки создания, модифицирования и использования последовательности.

Теоретическое обоснование

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

Создание последовательности

Для создания последовательности используется следующий синтаксис:

```
CREATE SEQUENCE имя_последовательности;
```

Данная команда создает последовательность, которая начинается с 1 и увеличивается на 1 при каждом обращении. Тем не менее при определении по-

следовательности можно использовать много дополнительных параметров.

Приведенный ниже синтаксис позволяет увидеть основные из них:

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

Параметр INCREMENT BY позволяет создавать последовательности с инкрементом, отличным от 1. Значение этого параметра может содержать до 28 цифр. Если указать отрицательное число, то значение последовательности будет уменьшаться при каждом обращении.

Параметр START WITH позволяет создать последовательность, начальное значение которой отлично от 1. Это может пригодиться при создании последовательности для таблицы, уже содержащей записи, чтобы начать последовательность с числа, следующего за наибольшим существующим идентификатором записи.

Параметры MAXVALUE и MINVALUE позволяют ограничить интервал чисел, генерируемых последовательностью. Если использовать их в сочетании с параметром CYCLE, заданное множество значений будет циклически повторяться.

Использование последовательности

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

Несмотря на то, что последовательности обычно создаются для одной

таблицы, в СУБД нет ограничения, которое бы этого требовало. Последовательность является независимым объектом. Она может использоваться в одной таблице, во многих таблицах или ни в одной из таблиц.

Модификация существующей последовательности

Созданную последовательность можно модифицировать различными способами. В частности, можно изменить значение инкремента, скорректировать или удалить минимальное и максимальное значения, разрешить или запретить циклический повтор по достижении граничного значения.

Синтаксис, позволяющий выполнять эти изменения в существующей последовательности, очень похож на синтаксис, используемый для ее создания. Он выглядит следующим образом:

```
ALTER SEQUENCE имя_последовательности
[INCREMENT BY значение_инкремента]
[MAXVALUE наибольшее_значение \NOMAXVALUE]
[MINVALUE наименьшее_значение \NOMINVALUE]
[CYCLE \NOCYCLE] ;
```

Методика и порядок выполнения работы

Методика выполнения работы

1. Создадим последовательность.

```
CREATE SEQUENCE test_seq;
```

2. Протестируем созданную последовательность, введем следующие команды и сравним результаты с показанными на рис. 10.1

```
SELECT test_seq.NEXTVAL FROM DUAL;
SELECT test_seq.NEXTVAL FROM DUAL;
SELECT test_seq.NEXTVAL FROM DUAL;
```



Рисунок 10.1 – Результат выполнения последовательности

3. Теперь заполним столбцы таблицы из последовательности. Это делается путем включения ссылки на последовательность в оператор INSERT, как в приведенных ниже командах. Введем их и сравним результаты с показанными на рис. 10.2.

```

CREATE TABLE test_n (id NUMBER(18,0), text VARCHAR2(10));
INSERT INTO test_n VALUES (test_seq.NEXTVAL, 'Rec1');
INSERT INTO test_n VALUES (test_seq.NEXTVAL, 'Rec2');
INSERT INTO test_n VALUES (test_seq.NEXTVAL, 'Rec3');
SELECT * FROM test_n;

```

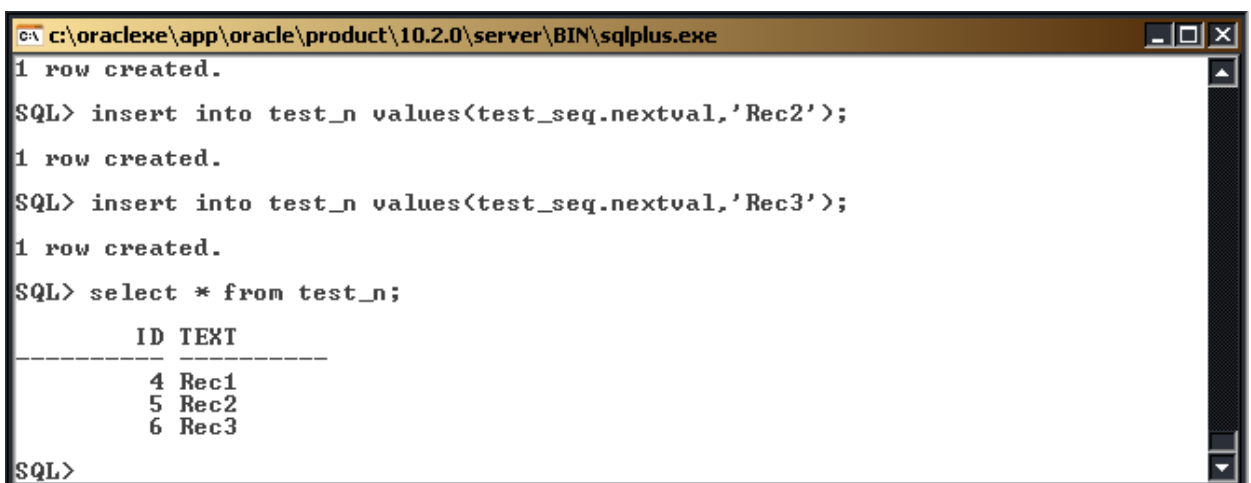


Рисунок 10.2 – Вывод данных после включения ссылки на последовательность в операторе INSERT

4. Модифицируем последовательность, для этого введем следующие ко-

манды и сравним их результаты с показанными на рис. 10.3:

```
ALTER SEQUENCE test_seq MAXVALUE 10;

SELECT test_seq.NEXTVAL FROM DUAL;

SELECT test_seq.NEXTVAL FROM DUAL;

SELECT test_seq.NEXTVAL FROM DUAL;

SELECT test_seq.NEXTVAL FROM DUAL;

SELECT test_seq.NEXTVAL FROM DUAL;
```

```
c:\oracle\app\oracle\product\10.2.0\server\BIN\sqlplus.exe
1* ALTER SEQUENCE test_seq MAXVALUE 10
SQL> /
Sequence altered.
SQL> select test_seq.NEXTVAL from dual;
   NEXTVAL
-----
        7
SQL> select test_seq.NEXTVAL from dual;
   NEXTVAL
-----
        8
SQL> select test_seq.NEXTVAL from dual;
   NEXTVAL
-----
        9
SQL> select test_seq.NEXTVAL from dual;
   NEXTVAL
-----
       10
SQL> select test_seq.NEXTVAL from dual;
select test_seq.NEXTVAL from dual
*
ERROR at line 1:
ORA-08004: sequence TEST_SEQ.NEXTVAL exceeds MAXVALUE and cannot be
instantiated
SQL>
```

Рисунок 10.3 – Результат выполнения модифицированной последовательности

Порядок выполнения работы

1. Создайте последовательность.
2. Заполните столбцы таблицы из последовательности.
3. Модифицируйте последовательность с помощью MAXVALUE.

Содержание отчета и его форма

Отчет должен содержать

- название лабораторной работы;

- цель работы;
- порядок выполнения работы;
- результаты выполнения;
- вывод о результатах создания, модифицирования и использования последовательности.

Контрольные вопросы и защита работы

Контрольные вопросы

1. Что такое последовательность?
2. Синтаксис команды для создания последовательности.
3. Параметры CURRVAL и NEXTVAL.
4. Параметр INCREMENT BY.

Защита работы

Защита работы заключается в

- выполнении заданий раздела «Методика и порядок выполнения работы»;
- ответах на контрольные вопросы;
- предоставлении отчета.

ЛАБОРАТОРНАЯ РАБОТА №11

Синонимы

Цель и содержание

Цель работы: приобрести навыки создания и использования синонимов.

Теоретическое обоснование

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

С какой целью создается синоним для какого-либо объекта? Главным образом для удобства: если часто ссылаться на таблицу с длинным именем, то возможно оценить использование короткого имени без переименования таблицы и изменения кода, который на нее ссылается.

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

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

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

Создание синонима

Команда создания синонима имеет следующий синтаксис:

```
CREATE [PUBLIC] SYNONYM имя_синонима FOR имя_объекта;
```

Если требуется сделать таблицу доступной другим пользователям, создается синоним с тем же именем, что и у таблицы. Пример команды такого типа

```
CREATE PUBLIC SYNONYM prod FOR product_n;
```

Модификация существующего синонима

Ввиду чрезвычайной простоты синонимов, СУБД не предоставляет никаких средств для их изменения. При необходимости можно просто удалить старый синоним и создать новый. Команда удаления синонима имеет следующий синтаксис:

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

Методика и порядок выполнения работы

Методика выполнения работы

1. Создадим и протестируем синоним, для этого введем приведенные ниже команды. На рис. 11.1 показано, какие результаты сможем увидеть.

```
SELECT * FROM product_n;

CREATE SYNONYM prod FOR product_n;

SELECT * FROM prod;
```

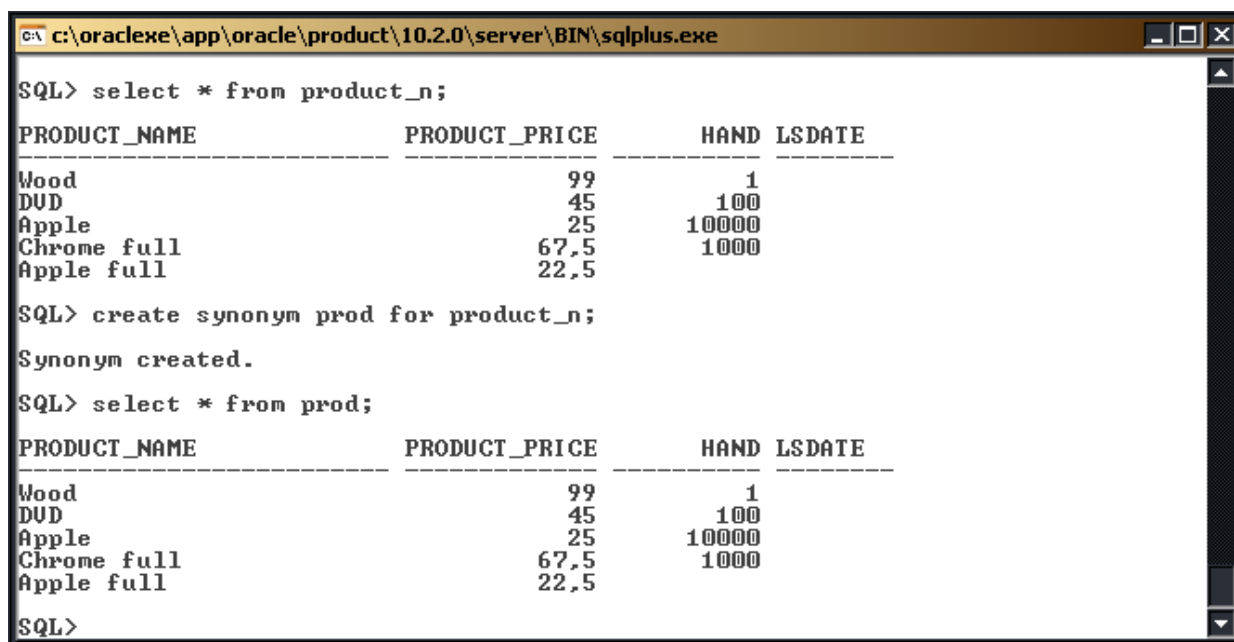


Рисунок 11.1 – Создание и выполнение синонима

2. Удалим первый из созданных выше синонимов (PROD), введем следующую команду:

```
DROP SYNONYM prod;
```

Для удаления общего синонима введем такую команду:

```
DROP PUBLIC SYNONYM prod;
```

Порядок выполнения работы

1. Создайте и протестируйте синоним к любой таблице.
2. Удалите созданные синонимы.

Содержание отчета и его форма

Отчет должен содержать

- название лабораторной работы;
- цель работы;

- порядок выполнения работы;
- результаты выполнения;
- вывод результате создания и использования синонимов.

Контрольные вопросы и защита работы

Контрольные вопросы

1. Что такое синонимы?
2. Команда создания синонима.
3. Команда удаления синонима.

Защита работы

Защита работы заключается в

- выполнении заданий раздела «Методика и порядок выполнения работы»;
- ответах на контрольные вопросы;
- предоставлении отчета.

ЛАБОРАТОРНАЯ РАБОТА №12

PL/SQL, хранимые процедуры, функции

Цель и содержание

Цель работы: приобрести навыки разработки программ на языке PL/SQL, создания и использования хранимых процедур и функций.

Теоретическое обоснование

SQL создавался как язык запросов к базам данных (Structured Query Language, язык структурированных запросов), его развитие шло по пути оптимизации под единственную задачу – декларативного управления данными. Различные поставщики программного обеспечения баз данных выработали общие стандарты SQL, но не определились, как предоставить пользователям более сложные SQL-ориентированные программные средства. В результате каждый

поставщик СУБД предлагает оригинальные или частично стандартизованные продукты. Корпорация называет свое решение PL/SQL. Можно считать это сокращением от «Programming Language for SQL».

PL/SQL предоставляет средства, позволяющие выполнять сложную обработку информации. Так, *хранимые процедуры* (stored procedures) PL/SQL можно использовать для определения эффективности работы служащих и последующего принятия решения о выплате премий. *Эlegantные функции PL/SQL* позволяют рассчитывать налоговые вычеты для служащих и т. д.

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

Хранимые процедуры

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

Хранимая процедура может выполнять SQL-операторы и манипулировать данными в таблицах. Ее можно вызывать из другой хранимой процедуры PL/SQL, хранимой функции или триггера, а также непосредственно из строки приглашения SQL*Plus. Процедура состоит из двух основных частей: спецификации и тела. *Спецификация процедуры* (procedure specification) включает в себя имя процедуры и описание ее входных и выходных данных. Эти входные и вы-

ходные данные называются *формальными параметрами* (formal parameters) или *формальными аргументами* (formal arguments). Если при вызове процедуры указываются параметры командной строки или другие входные данные, эти значения называются *фактическими* (actual) параметрами или фактическими аргументами. *Тело процедуры* (procedure body) представляет собой блок PL/SQL-кода.

Хранимые функции

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

Хранимые процедуры в сравнении с SQL-сценариями

SQL-сценарии размещаются на жестком диске компьютера, тогда как хранимые процедуры – в базе данных. SQL-сценарий содержит серию команд SQL, выполняющихся строго последовательно. Хранимая процедура, напротив, может содержать команды передачи управления, позволяющие циклически выполнять некоторую секцию кода, переходить на другую секцию при выполнении определенных условий и реагировать на ошибки указанным вами способом.

Структура блока PL/SQL

Весь код PL/SQL, выполняющий фактическую работу, состоит из *базовых блоков*. Базовый блок (basic block) PL/SQL состоит из четырех секций: *секции заголовка* (header section), *необязательной секции объявлений* (declaration section), *выполняемой секции* (execution section) и *необязательной секции исключений* (exception section).

Анонимный блок (anonomousblock) – это блок PL/SQL без секции заголовка, иначе говоря, секции имени, поэтому он и называется анонимным. Анонимные блоки могут выполняться из SQL* Plus и использоваться в функциях, процедурах и триггерах PL/SQL. Процедуры, функции и триггеры также состоят из базовых блоков. Это означает, что базовый блок можно помещать в другой ба-

зовый блок.

Секция заголовка

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

```
«just_a_label» DECLARE
```

Метки блоков облегчают чтение кода. В процедуре, содержащей вложенные блоки (блоки внутри других блоков), можно ссылаться на элемент определенного блока, предваряя имя элемента именем блока (например, *метка_блока.метка_элемента*).

Секция объявлений

Секция объявлений не является обязательной. В случае использования она начинается после *секции заголовка* и оканчивается перед ключевым, словом BEGIN. Эта секция содержит объявления переменных, констант, курсоров, исключений, функций и процедур PL/SQL, которые будут использоваться в выполняемой секции и секции исключений. Все объявления переменных и констант должны размещаться до объявлений функций или процедур. Объявление сообщает PL/SQL о том, что нужно создать переменную, константу, курсор, функцию или процедуру согласно приведенной спецификации.

Когда выполнение базового блока завершается, все элементы, объявленные в секции объявлений, перестают существовать. Элементы, объявленные в секции объявлений базового блока, могут использоваться только в пределах этого блока. Одним словом, все, что находится в *секции объявлений*, принадлежит блоку и может использоваться только внутри него, а следовательно, суще-

ствуется только на протяжении его времени жизни. Часть кода, в которой может использоваться переменная, называется *областью видимости* (scope).

Выполняемая секция

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

```
BEGIN
    Один_и_более_PL/SQL-операторов
    [Секция_исключений]
END [имя_функции_или_процедуры] ;
```

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

Секция исключений

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

PL/SQL во всех этих случаях предоставляет средства *обработки исключений* (exception handling). Рассмотрим структуру *секции исключений*.

```
EXCEPTION
    WHEN имя_исключения
    THEN действия, предпринимаемые при
```

Пакет DBMSOUTPUT и процедура PUTLINE являются частью базы данных; вместе они позволяют построчно отображать текст на экране SQL*Plus.

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

Создание простой PL/SQL- процедуры

Синтаксис создания хранимой процедуры имеет вид:

```
CREATE PROCEDURE спецификация_процедуры IS тело_процедуры
```

При создании функции ключевое слово PROCEDURE заменяется на FUNCTION:

```
CREATE FUNCTION спецификация_функции IS тело_функции
```

Прямой слэш (/) сообщает PL/SQL, что ввод программы завершен и нужно перейти к выполнению команд. Процедуру или функцию можно создать заново, используя команду CREATE OR REPLACE вместо CREATE. Это приведет к уничтожению старого определения и замене его на новое. При отсутствии старого определения будет просто создано новое.

```
CREATE OR REPLACE спецификация_процедуры IS тело_процедуры
```

Команда SET SERVEROUTPUT ON позволяет увидеть выходные данные. Команда EXECUTE запускает процедуру на выполнение.

Вызов процедур и функций

Процедура или функция может иметь формальные параметры как со значениями по умолчанию, так и без них. Фактически она может вообще не иметь формальных параметров. В каждом случае способ вызова процедуры или функции будет отличаться.

Типы данных фактических параметров должны совпадать с типами данных соответствующих формальных параметров или допускать преобразование в них.

Фактические параметры должны быть указаны для всех формальных параметров, не имеющих значений по умолчанию.

При вызове процедуры без каких-либо параметров можно указывать только ее имя, со скобками или без скобок:

имя_процедуры() ;

или

имя_процедуры;

Для вызова функций используется такой же синтаксис за одним исключением: если функция вызывается как часть выражения, точка с запятой не ставится.

Когда процедура имеет параметры со значениями по умолчанию, и все эти параметры находятся в конце списка формальных параметров в спецификации процедуры, при вызове процедуры можно не указывать их значения. Однако все формальные параметры, для которых во время вызова указываются значения, должны быть перечислены до любых формальных параметров, для которых значения не указываются. В итоге вызов будет выглядеть следующим образом:

*имя_процедуры (факт_парам_1, факт_парам_2, ...
факт_парам_N) ;*

К функциям применимы те же методы вызова. Однако функции могут появляться внутри других выражений и, соответственно, не иметь в конце точки с запятой.

Переменные и константы PL/SQL

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

Объявление переменных PL/SQL

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

имя_переменной *тип_данных* [[NOT NULL] := *выражение_по_умолчанию*];

имя_переменной *тип_данных* [[NOTNULL] DEFAULT *выражение_по_умолчанию*];

Имя_переменной – это любой правильный идентификатор PL/SQL. Правильный идентификатор PL/SQL должен:

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

Типы данных – это любой допустимый тип данных SQL или PL/SQL.

Вы уже знаете о типах данных SQL – NUMBER, VARCHAR2 и DATE. Кроме них PL/SQL имеет дополнительные типы данных, отсутствующие в SQL. Полный их список можно найти в справочниках по PL/SQL, издаваемых корпорацией.

Объявление констант PL/SQL

Синтаксис объявления константы имеет следующий вид:

имя_переменной *тип_данных* CONSTANT := *выражение*;

В отличие от переменных, константам обязательно присваивается значение, которое нельзя изменять на протяжении времени жизни константы. Константы очень полезны для поддержания безопасности и дисциплины при разработке больших и сложных приложений. Например, если требуется, чтобы процедура PL/SQL не модифицировала передаваемые ей данные, желательно объ-

явить их константами. Если процедура все же попытается их модифицировать, PL/SQL возбудит исключение.

Присваивание значений переменным

Есть три способа изменения значения переменной. Во-первых, ей можно присвоить значение выражения, используя оператор присваивания PL/SQL. Вот соответствующий синтаксис:

имя_переменной := выражение;

Во-вторых, переменная может быть передана PL/SQL-Процедуре в качестве фактического параметра, соответствующего одному из формальных параметров IN OUT или OUT.

Управляющие структуры в PL/SQL

Во многих случаях программа должна выполнять разные действия в зависимости от того, выполнено ли некоторое условие. Например, если сумма заказа превышает одну величину, делается скидка в 7%, а если сумма превышает другую величину, скидка увеличивается до 10%. Такая же логика может потребоваться в приложении, которое распечатывает окончательные счета для клиентов. Это называется *условной обработкой* (conditional processing) данных. В зависимости от результата проверки условия выполняются разные части кода.

PL/SQL дает возможность выполнять условную и итеративную обработку. Предоставляемые им конструкции изменяют *ход выполнения программы* (program flow), управляя *последовательностью выполнения* (flow of execution).

Оператор IF

Оператор IF имеет следующий синтаксис:

```
IF условие_1 THEN действие_1;
[ELSIF условие_2 THEN действие_2;]
[ELSE альтернативное_действие;]
END IF;
```

Действие и альтернативное_действие представляют один или несколько PL/SQL-операторов. Каждая группа операторов выполняется только в том слу-

чае, если выполнено соответствующее условие. После того как обнаружено выполнение одного из условий, остальные условия не проверяются.

Циклы

PL/SQL предоставляет три различные конструкции для итеративной обработки. Каждая из них позволяет циклически выполнять набор операторов PL/SQL. Выход из цикла осуществляется в зависимости от некоторого условия.

LOOP

Конструкция LOOP имеет следующий синтаксис:

```
<<имя цикла>>
LOOP
    операторы;
    EXIT имя цикла [WHEN условие_выхода];
    операторы;
END LOOP;
```

При наличии конструкции WHEN все операторы в теле цикла повторяются до тех пор, пока выражение *условие_выхода* не примет положительное значение (т.е. не станет истинным). Условие выхода проверяется на каждом проходе, иначе называемом итерацией. Как только выражение принимает значение «истина», все операторы после EXIT пропускаются, итерации прекращаются и выполнение продолжается с первого оператора, следующего за END LOOP. Если условие WHEN отсутствует, операторы между LOOP и EXIT выполняются только один раз. Очевидно, что опустив условие WHEN, вы поступите нелогично.

Цикл WHILE

Еще одной разновидностью цикла является цикл WHILE. Он хорошо подходит в ситуациях, когда количество итераций заранее неизвестно, и определяется некоторым внешним фактором. Цикл WHILE имеет следующий синтаксис:

```
WHILE условие_выхода LOOP
    операторы;
END LOOP;
```

Условие WHILE проверяется перед каждым входом в цикл. Если оно имеет значение «истина», то выполняется очередная итерация.

Цикл FOR

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

```
FOR счетчик IN [REVERSE] нижняя_граница..верхняя_граница
    LOOP
        операторы;
    END LOOP;
```

Курсоры

Курсор — это исключительно важная конструкция PL/SQL, лежащая в основе взаимодействия PL/SQL и SQL. Название «курсор» означает «текущий набор записей». Курсор представляет собой специальный элемент PL/SQL, с которым связан SQL-оператор SELECT. Используя курсор, можно отдельно обрабатывать каждую строку связанного с ним SQL-оператора. Курсор объявляется в секции объявлений базового блока. Он открывается командой OPEN, а выборка строк осуществляется с помощью команды FETCH. После завершения всей обработки курсор закрывается командой CLOSE. Закрытие курсора освобождает те системные ресурсы, которые использовались, пока он был открыт. Строки, выбранные курсором, можно заблокировать, чтобы предотвратить их модификацию другими пользователями.

Закрытие курсора или выполнение явной операции COMMIT или ROLLBACK приведет к разблокированию строк.

Для SQL-операторов, используемых в коде PL/SQL, применяются скрытые, или *неявные* (implicit), курсоры.

Рассмотрим синтаксис явного курсора. Курсор объявляется в процедуре PL/SQL следующим образом:

```
CURSOR имя_курсора [ (параметр_1 [, параметр_2...]) ]
```

```

[RETURN спецификация_возврата]
IS
    оператор_select
    [FOR UPDATE
        [OF таблица_или_столбец_1
        [, таблица_или_столбец_2...]]
    ]
]

```

Параметры курсора похожи на параметры процедуры, затем исключением, что они всегда являются входными (IN). Использование параметров OUT или IN OUT невозможно, поскольку курсор не может их модифицировать. Параметры используются в конструкции WHERE курсорного оператора SELECT. Спецификация возврата показывает, записи какого типа будут выбираться оператором SELECT. *Таблица_или_столбец* – это имя столбца, который предстоит обновлять, или имя таблицы, в которой предстоит удалять или обновлять строки. Оно должно входить в число имен таблиц и столбцов, указанных в операторе SELECT курсора, и предназначено для документирования, показывая, какие элементы могут быть потенциально модифицированы кодом, использующим данный курсор. Команда FOR UPDATE блокирует строки, выбранные оператором SELECT при открытии курсора. Строки остаются заблокированными до тех пор, пока вы не закроете курсор рассмотренными выше способами.

Курсор имеет ряд индикаторов, показывающих его состояние. Они называются *атрибутами курсора* и приведены в табл. 12.1.

Таблица 12.1 – Атрибуты курсора

Атрибут	Описание
имя_курсора%ISOPEN	Позволяет проверить, открыт ли курсор. Если курсор <i>имя_курсора</i> уже открыт, возвращается значение TRUE
имя_курсора%ROWCOUNT	Количество строк таблицы, возвращенных оператором SELECT курсора
имя_курсора%FOUND	Позволяет проверить, была ли успешной последняя попытка получения записи из курсора. Если запись была выбрана, воз-

	возвращается значение TRUE
имя_курсора%NOTFOUND	Противоположен атрибуту FOUND. Если записей больше не найдено, возвращается значение TRUE

Запись PL/SQL

Запись PL/SQL – это набор данных базовых типов. К ней можно обращаться, как к единому целому. Для доступа к отдельным полям записи применяется нотация *имя_записи.имя_поля*, которую вы уже использовали для столбцов таблицы. Записи могут иметь один из трех типов, перечисленных ниже.

1. Основанные на таблице (table-based). Эти записи имеют поля, совпадающие по имени и типу со столбцами таблицы. Если курсор выбирает всю строку – например, оператором `SELECT * FROM некоторая_таблица` – то возвращаемые им записи можно непосредственно копировать в переменную, имеющую тип записи, основанной на таблице *некоторая_таблица*.

2. Основанные на курсоре (cursor-based). Поля этих записей совпадают по имени, типу и порядку с заключительным списком столбцов в курсорном операторе `SELECT`.

3. Определенные программистом (programmer-defined). Это записи, тип которых определяете вы сами.

Использование команд OPEN, FETCH и CLOSE

Команды открытия курсора, выборки из курсора и закрытия курсора имеют следующий синтаксис:

```
OPEN имя_курсора;

FETCH имя_курсора INTO переменная_или_список_переменных;

CLOSE имя_курсора;
```

После открытия курсор содержит набор записей, если в результате успешного выполнения оператора `SELECT` из базы данных были выбраны заданные строки. Каждая команда `FETCH` удаляет запись из открытого курсора и перемещает ее содержимое либо в переменную PL/SQL, тип записи которой совпадает с типом записи курсора, либо в группу переменных PL/SQL, где каж-

дая переменная в списке совпадает по типу с соответствующим полем в записи курсора.

Перед тем как пытаться выбрать из курсора очередную запись, следует проверить с помощью атрибутов FOUND и NOTFOUND, есть ли в нем еще записи. Выборки из пустого курсора будут все время давать последнюю запись, не приводя к ошибке. Необходимо проверять атрибуты FOUND и NOTFOUND при использовании FETCH.

Фактическая обработка записей из курсора обычно выполняется внутри цикла. При написании такого цикла желательно начать с проверки, была ли найдена запись в курсоре. Если да, можно продолжать необходимую обработку; в противном случае следует выйти из цикла. Тоже самое можно сделать более простым способом, используя курсорный цикл FOR. При этом PL/SQL будет осуществлять открытие, выборку и закрытие без вашего участия.

Курсорный цикл FOR

Синтаксис курсорного цикла FOR имеет следующий вид:

```
FOR запись_курсора IN имя_курсора LOOP
    операторы;
END LOOP;
```

Этот цикл выбирает записи из курсора в переменную типа *запись_курсора*. Поля *записи_курсора* можно использовать для доступа к данным из операторов PL/SQL, выполняемых в цикле. Когда все записи выбраны, цикл завершается. Для удобства открытие и закрытие курсора производится автоматически.

Попытавшись выбрать запись из неоткрытого курсора, вы получите сообщение `invalidcursor` (недействительный курсор). Если не закрывать курсоры, то в конце концов количество открытых курсоров достигнет максимальной величины, допускаемой системой. Имейте в виду, что неявные курсоры, которые будут рассмотрены позже, тоже вносят вклад в достижение этого предела.

Конструкция WHERE CURRENT OF

Когда курсор открывается для обновления или удаления выбранных за-

писей, можно использовать конструкцию

WHERE CURRENT OF *имя_курсора*

для доступа к таблице и строке, которые соответствуют последней записи, выбранной в конструкции WHERE оператора UPDATE или DELETE.

Методика и порядок выполнения работы

Методика выполнения работы

1. Рассмотрим пример структуры блока PL/SQL. Введем команду, которая позволит просматривать в SQL*Plus информацию, выводимую программой

SET SERVEROUTPUT ON

Теперь введем код анонимного блока и сравните полученные результаты с рис. 12.1.

```
DECLARE

num_a NUMBER:=6;

num_b NUMBER;

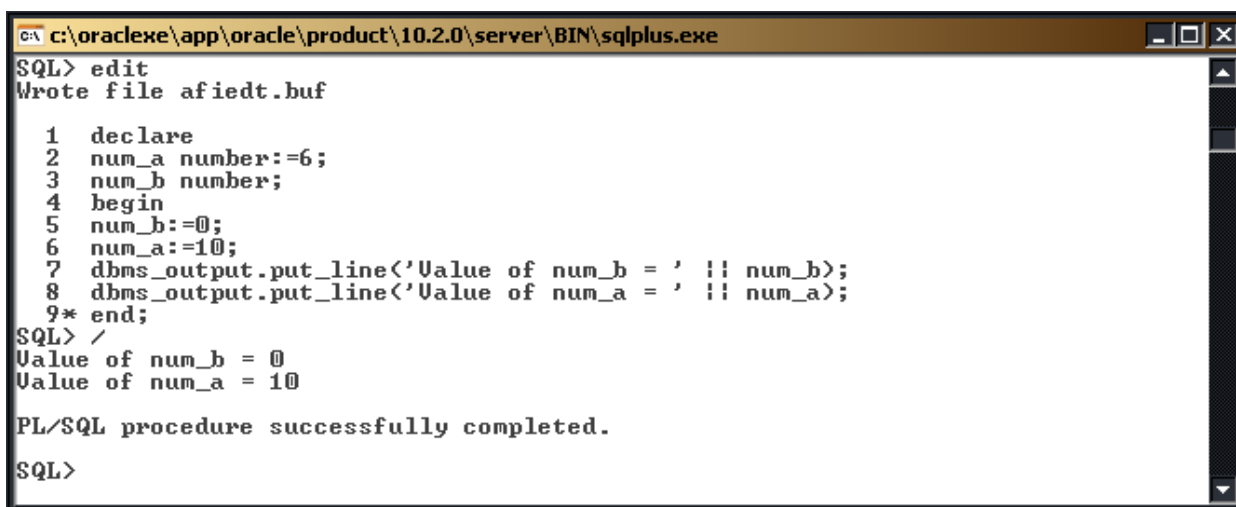
BEGIN

num_b:=0; num_a:=10;

dbms_output.put_line('Value of num_b = ' || num_b);

dbms_output.put_line('Value of num_a = ' || num_a);

END;
```



```
C:\oracle\app\oracle\product\10.2.0\server\BIN\sqlplus.exe
SQL> edit
Wrote file afiedt.buf

1  declare
2  num_a number:=6;
3  num_b number;
4  begin
5  num_b:=0;
6  num_a:=10;
7  dbms_output.put_line('Value of num_b = ' || num_b);
8  dbms_output.put_line('Value of num_a = ' || num_a);
9* end;
SQL> /
Value of num_b = 0
Value of num_a = 10

PL/SQL procedure successfully completed.
SQL>
```

Рисунок 12.1 – Результат выполнения анонимного блока на языке PL/SQL

2. Для создания простой PL/SQL- процедуры введем следующий код:

```
CREATE PROCEDURE my_first_proc IS
greetings VARCHAR2 (20);

BEGIN

greetings := 'Hello World';

dbms_output.put_line(greetings);

END my first proc; /
```

3. Теперь посмотрим, как можно вызывать эту процедуру из SQL*Plus:

```
SET SERVEROUTPUT ON

EXECUTE my_first_proc;
```

Также можно вызвать процедуру из анонимного блока, как показано ниже. Сравним полученные результаты с показанными на рис. 12.2.

```
BEGIN

my_first_proc;

END;
```

A screenshot of a Windows command prompt window titled "c:\oracle\app\oracle\product\10.2.0\server\BIN\sqlplus.exe". The window shows the following text: SQL> set serveroutput on; SQL> execute my_first_proc; Hello World; PL/SQL procedure successfully completed. SQL> begin 2 my_first_proc; 3 end; 4 /; Hello World; PL/SQL procedure successfully completed. SQL> The window has a standard Windows interface with a title bar, maximize, and close buttons.

Рисунок 12.2 – Результат выполнения процедуры my_first_proc

4. Для демонстрации изменения и присваивания значения переменной введем следующий код, результаты которого показаны на рис. 12.3.

```
DECLARE

product_guant NUMBER;

BEGIN
```

```

SELECT product_price
INTO product_guant
FROM product_n
WHERE product_name = 'Apple';

dbms_output.put_line('Apple price = ' || product_guant);

END;

/

```

```

c:\oracle\app\oracle\product\10.2.0\server\BIN\sqlplus.exe
SQL> edit
Wrote file afiedt.buf

 1  DECLARE
 2  product_guant NUMBER;
 3  BEGIN
 4  SELECT product_price
 5  INTO product_guant
 6  FROM product_n
 7  WHERE product_name = 'Apple';
 8  Dbms_output.put_line('Apple price = ' || product_guant);
 9* END;
SQL> /
Apple price = 25
PL/SQL procedure successfully completed.
SQL>

```

Рисунок 12.3 – Результат выполнения анонимной процедуры

5. Используем оператор IF при создании функции. Введем следующий пример и убедимся, что наши результаты совпадают с показанными на рис. 12.4.

```

CREATE OR REPLACE FUNCTION comp (r_order NUMBER) RETURN NUMBER
IS

```

```

    s_order NUMBER := 400;
    l_order NUMBER := 1000;
    s_dis    NUMBER := 1;
    l_dis    NUMBER := 5;

```

```

BEGIN

```

```

    IF (r_order < l_order AND r_order >= s_order) THEN
        RETURN (r_order * s_dis / 100) ;

```

```

ELSIF (r_order >= l_order) THEN
RETURN (r_order * l_dis / 100);
ELSE
RETURN(0);
END IF;
END comp;
/

```

6. Протестируем функцию, вызвав ее из анонимного блока.

```

SET SERVEROUTPUT ON

DECLARE

tiny  NUMBER := 20;
med   NUMBER := 600;
big   NUMBER := 4550;
wrong NUMBER := -35;

BEGIN

dbms_output.put_line('tiny =' || comp(tiny));
dbms_output.put_line('med =' || comp(med));
dbms_output.put_line('big =' || comp(big));
dbms_output.put_line('wrong =' || comp(wrong));

END; /

```

```

c:\oracle\app\oracle\product\10.2.0\server\BIN\sqlplus.exe
3 med  NUMBER := 600;
4 big  NUMBER := 4550;
5 wrong NUMBER := -35;
6 BEGIN
7 dbms_output.put_line('tiny =' || comp(tiny));
8 dbms_output.put_line('med =' || comp(med));
9 dbms_output.put_line('big =' || comp(big));
10 dbms_output.put_line('wrong =' || comp(wrong));
11* END;
SQL> /
tiny = 0
med = 6
big = 227,5
wrong = 0
PL/SQL procedure successfully completed.
SQL>

```

Рисунок 12.4 – Результат вызова функции comp

7. Для демонстрации использования цикла LOOP выполним приведенный ниже код и сравним результаты с рис. 12.5. В этом примере просто выводятся на печать первые десять чисел.

Для просмотра выходных данных введем команду SET SERVEROUTPUT ON.

```

DECLARE

num NUMBER := 1;

BEGIN

<<just>> LOOP

    dbms_output.put_line(num);

EXIT just

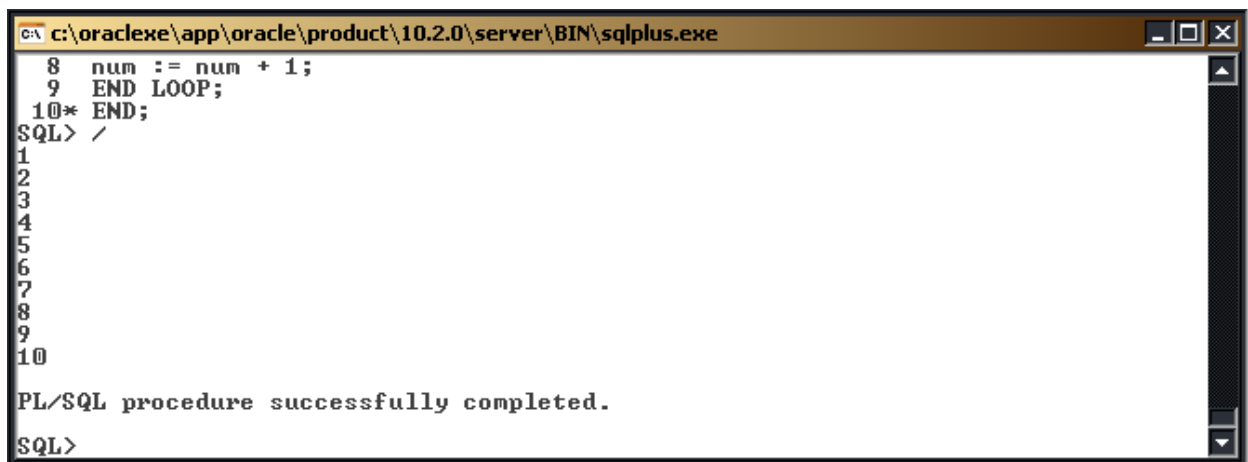
WHEN    (num >= 10);

    num := num + 1;

END LOOP;

END; /

```



```

c:\oracle\app\oracle\product\10.2.0\server\BIN\sqlplus.exe
8  num := num + 1;
9  END LOOP;
10* END;
SQL> /
1
2
3
4
5
6
7
8
9
10
PL/SQL procedure successfully completed.
SQL>

```

Рисунок 12.5 – Результат выполнения кода с использованием конструкции LOOP

8. Для демонстрации использования цикла WHILE выполним приведенный ниже код и сравним результаты с рис. 12.5.

```

DECLARE

```

```

num NUMBER := 1;

BEGIN

WHILE (num <= 10) LOOP
dbms_output.put_line(num);
num := num + 1;
END LOOP;

END; /

```

9. Для демонстрации использования цикла FOR выполним приведенный ниже код и сравним результаты с рис. 12.5.

```

BEGIN

FOR num IN 1..10 LOOP
dbms_output.put_line(num);
END LOOP;

END;

/

```

При использовании в этом цикле команды REVERSE, числа будут показаны в обратном порядке – от 10 до 1.

Порядок выполнения работы

1. Создайте и выполните хранимую процедуру, которая вставляет в таблицу некоторое количество записей.
2. Напишите функцию, которая умножает два числа и возвращает их произведение.
3. Напишите функцию, которая возвращает наибольшее значение из двух чисел.
4. Напишите процедуру, которая выводит на экран 10 раз слово «SQL», используя цикл LOOP, WHILE и FOR.

Содержание отчета и его форма

Отчет должен содержать

- название лабораторной работы;
- цель работы;
- порядок выполнения работы;
- результаты выполнения;
- вывод о проделанной работе.

Контрольные вопросы и защита работы

Контрольные вопросы

1. Что такое хранимые процедуры?
2. Что такое хранимые функции?
3. Чем отличаются хранимые функции от процедур?
4. Структура блока PL/SQL.
5. Как можно выполнить процедуру из SQL*Plus?
6. Синтаксис оператора IF.
7. Запись в языке PL/SQL.
8. Синтаксис оператора LOOP, WHILE, FOR.
9. Что такое курсоры?
10. Каково назначение команд OPEN, FETCH и CLOSE?
11. Конструкция WHERE CURRENT OF.

Защита работы

Защита работы заключается в

- выполнении заданий раздела «Методика и порядок выполнения работы»;
- ответах на контрольные вопросы;
- предоставлении отчета.

ЛАБОРАТОРНАЯ РАБОТА №13

Пакеты PL/SQL и Триггеры

Цель и содержание

Цель работы: приобрести навыки создания и использования пакетов PL/SQL, табличных триггеров.

Теоретическое обоснование

При разработке приложений PL/SQL предоставляет *пакеты* (packages) для совместного хранения полезных функций, процедур, типов записей и курсоров.

Аналогично процедуре или функции, пакет имеет спецификацию и тело. Однако спецификацию и тело пакета можно создавать по отдельности. Можно даже заменить одно тело пакета другим, сохранив совместимость с существующей спецификацией, и пакет по-прежнему будет работать.

Пользователи пакета не обязаны знать все детали его реализации, поэтому эти детали скрываются внутри тела. Тело хранится, компилируется и обрабатывается внутри базы данных и невидимо пользователям пакета. Для программирования своих задач им просто следует пользоваться спецификацией. Это очень важно, когда требуется защитить свой код, сделав его недоступным для хакеров или конкурентов.

Спецификация пакета имеет следующий вид:

```
CREATE PACKAGE имя_пакета IS
```

```

[объявления переменных_и_типов]
[спецификации курсоров]
[спецификации функций_и_процедур]
END [имя_пакета];

```

Для создания тела пакета используется следующий синтаксис:

```

CREATE OR REPLACE PACKAGE BODY имя_пакета IS
    [локальные_объявления]
    [полные_спецификации_курсоров_пакета],
    [полные_спецификации_функций_и_процедур_пакета]
BEGIN
    [выполняемые_операторы]
    [EXCEPTION]
    [обработчики_исключений]
END [имя_пакета];

```

Все переменные и типы, объявленные в спецификации пакета, доступны его пользователям. Спецификация позволяет узнать, что содержит пакет, а также какие переменные ожидаются этим содержимым на входе и выходе. Секция объявлений содержит объявления функций, процедур, исключений, курсоров, переменных и констант, доступных для использования в теле пакета. Выполняемая секция тела пакета содержит полные определения (без команды CREATE OR REPLACE) всех курсоров, функций и процедур, объявленных в его спецификации.

Переменные, присутствующие в спецификации пакета, называются *переменными пакета* (package variables). Они инициализируются только один раз – при первом обращении к нему. Когда производится вызов какой-либо составляющей пакета, загружает пакет в память, где он остается все то время, пока пользователь соединен с базой данных. При наличии нескольких сеансов переменные пакета и их значения становятся разделяемыми. Следует также заметить, что обращение к объектам, помещенным в пакет, в последующих сеансах может происходить быстрее.

Выполняемая секция пакета обычно используется для инициализации локальных переменных пакета или переменных пакета. Выполнение этой секции производится один раз при загрузке пакета, а следовательно, ее не стоит использовать для каких-либо повторяющихся действий, поскольку обратиться к ней более одного раза все равно не удастся.

Для обращения к процедурам, переменным и функциям пакета используется нотация *контейнер.содержимое*. В данном случае она имеет вид *имя_пакета.объект_пакета*. Так, например, функция *dbms_output.put_line* из пакета *dbms_output*. При ссылках на функции внутри их собственного пакета нет необходимости помещать квалификатор имя пакета перед именем функции.

Триггеры

Триггер – это процедура PL/SQL, которая выполняется автоматически, когда происходит некоторое заданное событие, называемое *триггерным событием* (triggering event). Например, можно писать триггеры, срабатывающие при выполнении над таблицей операций INSERT, UPDATE или DELETE, при выдаче команд DDL, при входе пользователя в систему или его выходе из системы, при запуске или останове базы данных, при возникновении ошибок.

Между триггерами и процедурами PL/SQL есть три различия.

1. Триггеры нельзя вызывать из кода программы. СУБД вызывает их автоматически в ответ на определенное событие.
2. Триггеры не имеют списка параметров.
3. Спецификация триггера немного отличается от спецификации процедуры.

Сходство между триггерами и процедурами состоит в следующем.

1. Тело триггера выглядит точно так же, как и тело процедуры.
 2. Триггер не возвращает значений.
 3. Триггеры автоматически генерируют производные значения столбцов.
- Например, при обработке каждого нового заказа комиссионные торгового агента могут рассчитываться и вводиться в соответствующую таблицу автоматиче-

ски.

4. Триггеры помогают предотвращать неверные транзакции. Например, триггер может воспрепятствовать увеличению зарплаты служащего на большую величину, чем предусмотрено бюджетом. Но это нельзя сделать с помощью контрольных ограничений, поскольку контрольное ограничение имеет дело только со значениями текущей строки, а следовательно, не может определять общий размер бюджета.

5. Триггеры можно использовать для реализации сложных процедур авторизации. Например, значение зарплаты могут изменять только руководители и только для людей, находящихся в их подчинении.

6. Триггеры используются для реализации сложных бизнес-правил.

Например, триггер может проверять, доступен ли заказанный товар к дате отгрузки, и если нет, то автоматически размещать заказ на пополнение запасов, чтобы на момент отгрузки на складе находилось достаточное количество этого товара.

7. Триггеры могут обеспечивать прозрачное протоколирование событий, например, отслеживать, сколько раз отдельный продавец обращался к таблице с инвентарной ведомостью.

8. Триггеры могут обеспечивать сложный аудит. Например, с помощью триггера легко выяснить, какое среднее число обращений к базе данных требуется данному продавцу для обработки одного заказа.

9. Триггеры могут собирать статистику доступа к таблицам. Например, триггер позволяет зафиксировать, сколько раз в течение дня выполнялись запросы на «Apple» к таблице с каталогом товаров.

Таким образом, триггеры представляют собой весьма мощные инструменты. Однако их следует использовать аккуратно, поскольку триггер может срабатывать при каждом обращении к таблице, тем самым увеличивая нагрузку на сервер базы данных. Разумное практическое правило состоит в том, чтобы использовать триггер только для действий, которые нельзя выполнить другими

средствами. Например, если есть возможность создать одно или несколько ограничений, выполняющих работу некоторого триггера, следует использовать эти ограничения. Если итоговые величины допустимо вычислять в нерабочее время, систему лучше спроектировать именно так, а не использовать триггеры, срабатывающие при каждой транзакции. Предположим, например, что банк принимает решение о доставке дополнительной наличности из центрального хранилища в зависимости от того, сколько 20-долларовых банкнот было выдано за текущий день. Эту величину можно определять по окончании рабочего дня. Если же для обновления счетчика 20-долларовых банкнот будет использоваться триггер, то его постоянное срабатывание может существенно замедлить скорость выполнения транзакций в рабочие часы.

Триггеры, определенные для таблиц, называются *табличными триггерами* (table triggers). Далее речь пойдет именно о них. Синтаксис создания триггера имеет следующей вид:

```
CREATE      OR      REPLACE      TRIGGER      имя_триггера      мо-
мент_срабатывания
      триггерное_событие
ON имя_таблицы
[WHEN триггерное_ограничение]
[FOR EACH ROW]
[DECLARE
      объявления]
BEGIN
      операторы
[EXCEPTION
      WHEN имя_исключения
      THEN... ]
END имя_триггера;
```

Типы триггеров

Момент_срабатывания определяет, когда будет срабатывать триггер: до

(BEFORE) или после (AFTER) наступления триггерного события (выполнения запускающего оператора). Если указано значение BEFORE, триггер выполняется до каких-либо проверок ограничений на строки, затрагиваемые триггерным событием. Никакие строки не блокируются. Триггер этого типа называется, соответственно, BEFORE-триггером (BEFORE trigger).

Если выбрать ключевое слово AFTER, то триггер будет срабатывать после того, как запускающий оператор завершит свою работу и будут выполнены проверки всех ограничений. В этом случае затрагиваемые строки блокируются на время выполнения триггера. Триггер этого типа называется AFTER-триггером (AFTER trigger).

Триггерное_событие может принимать значения INSERT, UPDATE или DELETE.

Триггерное_ограничение – это одно и более дополнительных условий, которые должны быть выполнены для срабатывания триггера.

Необязательный набор ключевых слов FOR EACH ROW указывает на необходимость выполнить тело триггера для каждой строки, затрагиваемой запускающим оператором. Такие триггеры называются *строчными* (row triggers). Если опция FOR EACH ROW отсутствует, то при наступлении триггерного события триггер выполняется только один раз. В этом случае он называется *операторным* триггером (statement trigger), поскольку выполняется только один раз для каждого запускающего оператора.

Часть кода между DECLARE и END *имя_триггера* представляет собой обычный базовый блок PL/SQL.

Различные триггерные события можно комбинировать с помощью оператора OR. Например:

DELETE OR INSERT *остальные операторы*

В случае использования UPDATE можно указать список столбцов:

UPDATE OF *столбец_1, столбец_2, ...*

При использовании UPDATE в операторах PL/SQL обращение к новой и старой строкам выполняется с помощью слов «new» и «old», предваренных

двоеточием. Так, :old.имя_столбца даст значение, которое столбец имел до обновления. Однако в триггерном ограничении имена «old» и «new» используются без двоеточий.

SQL-операторы в теле триггера не могут:

- считывать или модифицировать информацию любой таблицы, изменяющейся в результате выполнения активизирующего оператора. В число таких таблиц входит и сама активизирующая таблица.

- считывать или модифицировать информацию столбца первичного ключа, уникальных столбцов и столбцов внешних ключей таблицы, являющейся ограничивающей по отношению к изменяющейся таблице.

Модификация триггеров

Для триггеров, как и для представлений, не существует команды модификации. Старый триггер просто заменяется новым с помощью команды CREATE OR REPLACE TRIGGER.

Триггер можно удалить, выдав команду со следующим синтаксисом:

```
DROP TRIGGER имя_триггера;
```

Существующие триггеры можно деактивизировать и повторно активизировать, используя команды с таким синтаксисом:

```
ALTER TRIGGER имя_триггера DISABLE;
```

```
ALTER TRIGGER имя_триггера ENABLE;
```

Тонкости, касающиеся триггеров

При создании триггеров необходимо учитывать следующее:

- триггер, в котором выполняются операторы DML, может вызывать срабатывание других триггеров. В результате количество сработавших триггеров может оказаться довольно большим;

- в триггере, запускаемом оператором INSERT, имеет смысл обращаться

только к новым значениям столбцов. Поскольку INSERT создает строку, старым значением всегда будет NULL;

- в триггере, запускаемом оператором UPDATE, можно обращаться

к старым и новым значениям столбцов. Это касается как BEFORE, так и AFTER-триггеров;

- в триггере, запускаемом оператором DELETE, имеет смысл обращаться только к старым значениям столбцов. Поскольку удаленная строка перестает существовать, новым значением всегда будет NULL. Однако значения new нельзя модифицировать. Если вы попытаетесь это сделать, будет выдано сообщение об ошибке ORA-4084;

- в теле триггера нельзя использовать операторы ROLLBACK, COMMIT и SAVEPOINT;

- при возникновении необрабатываемых исключений производится откат всех изменений, включая те, которые были выполнены запускающим оператором;

- если для одного триггерного события определено более одного триггера, то порядок их срабатывания не определен, т. е. нельзя заранее сказать, в какой последовательности они будут срабатывать.

- когда триггер пытается прочитать таблицу, а затем записать в нее данные, возбуждается исключение «мутирующей таблицы». Хотя намерение ограничить действия такого типа вполне объяснимо, зашла в этом дальше, чем необходимо, запретив операции, при которых таблица просто блокируется, а не изменяется. Один из способов обойти эту проблему заключается в том, чтобы создать вторую таблицу с копиями столбцов, которые триггер должен прочитать из главной таблицы. В этом случае триггер получает необходимые значения из второй таблицы, а затем выполняет запланированные действия с первой таблицей. При реализации данного подхода важно обеспечить синхронизацию таблиц. Для этого все операции INSERT, UPDATE и DELETE над главной таблицей должны отражаться триггером на второй таблице.

Методика и порядок выполнения работы

Методика выполнения работы

1. Запустим приведенный ниже SQL-сценарий, чтобы увидеть, как созда-

ются и используются пакеты. На рис. 13.1 показаны выходные данные SQL*Plus.

```
CREATE OR REPLACE PACKAGE pack IS DATE_LOADED DATE;
FUNCTION comp(r_order NUMBER) RETURN NUMBER;
END pack;

CREATE OR REPLACE PACKAGE BODY pack IS
    s_order NUMBER := 400;
    l_order NUMBER := 1000;
    s_dis    NUMBER := 1;
    l_dis    NUMBER := 5;
    FUNCTION comp (r_order NUMBER) RETURN NUMBER IS
    BEGIN
        IF (r_order < l_order AND r_order >= s_order) THEN
            RETURN (r_order * s_dis / 100) ;
        ELSIF (r_order >= l_order) THEN
            RETURN (r_order * l_dis / 100);
        ELSE RETURN(0);
        END IF;
    END comp;
END pack;/
```

```

C:\oracle\app\oracle\product\10.2.0\server\BIN\sqlplus.exe
1  CREATE OR REPLACE PACKAGE pack IS
2  DATE_LOADED DATE;
3  FUNCTION comp(r_order NUMBER) RETURN NUMBER;
4* END pack;
SQL> /

Package created.

SQL> edit
Wrote file afiedt.buf

1  CREATE OR REPLACE PACKAGE BODY pack IS
2  s_order NUMBER := 400;
3  l_order NUMBER := 1000;
4  s_dis NUMBER := 1;
5  l_dis NUMBER := 5;
6  FUNCTION comp (r_order NUMBER) RETURN NUMBER IS
7  BEGIN
8  IF (r_order < l_order AND r_order >= s_order) THEN
9  RETURN (r_order * s_dis / 100) ;
10 ELSEIF (r_order >= l_order) THEN
11 RETURN (r_order * l_dis / 100);
12 ELSE
13 RETURN(0);
14 END IF;
15 END comp;
16* END pack;
SQL> /

Package body created.

SQL>

```

Рисунок 13.1 – Создание пакета pack

2. Пример использования пакета (рис. 13.2)

```

DECLARE med    NUMBER := 600;

BEGIN

dbms_output.put_line('med =' || pack.comp (med) );

END; /

```

```

C:\oracle\app\oracle\product\10.2.0\server\BIN\sqlplus.exe
1  DECLARE
2  med    NUMBER := 600;
3  BEGIN
4  dbms_output.put_line('med = ' || pack.comp (med));
5* END;
SQL> /
med = 6

PL/SQL procedure successfully completed.

SQL>

```

Рисунок 13.2 – Использование пакета pack для вызова функции comp

3. Создадим простейший триггер предотвращающий изменение (UPDATE) имени продукта (product_name) в таблице product_n:

```

CREATE OR REPLACE TRIGGER ins
BEFORE INSERT OR UPDATE ON product_n FOR EACH ROW

```

```

DECLARE
no_name_change EXCEPTION;
BEGIN
IF (UPDATING AND (:NEW.product_name != :OLD.product_name))
THEN RAISE no_name_change;
END IF;
EXCEPTION
WHEN no_name_change THEN
dbms_output.put_line('Change of product name not allowed');
END INS;

```

4. Вызовем срабатывание триггера, результат на рис. 13.3:

```

UPDATE product_n SET product_name = 'LOOP'
WHERE product_name = 'Apple'

```

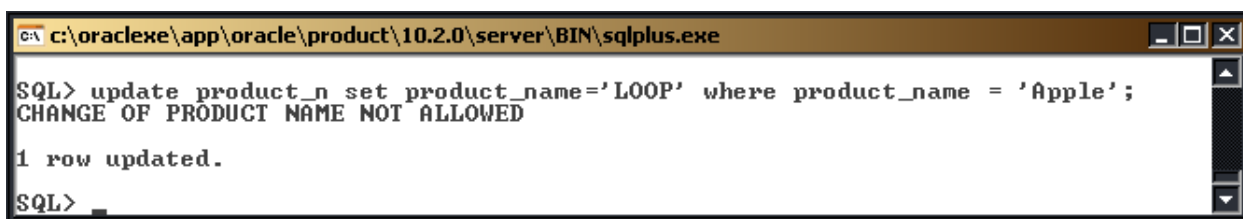


Рисунок 13.3 – Результат срабатывания триггера

Порядок выполнения работы

1. Создайте пакет PL/SQL из процедур и функций лабораторной работы №12.
2. Измените приведенный пример триггера на срабатывание команды DROP.
3. Деактивируйте и протестируйте ваш триггер.
4. Удалите созданный триггер.

Содержание отчета и его форма

Отчет должен содержать

- название лабораторной работы;
- цель работы;
- порядок выполнения работы;

- результаты выполнения;
- вывод о результатах создания, использования пакетов PL/SQL и табличных триггеров.

Контрольные вопросы и защита работы

Контрольные вопросы

1. Для чего необходимы пакеты PL/SQL?
2. Что такое триггер?
3. Какие типы триггеров существуют?
4. Какие существуют отличия между триггерами и процедурами PL/SQL?
5. Что необходимо учитывать при создании триггеров?

Защита работы

Защита работы заключается в

- выполнении заданий раздела «Методика и порядок выполнения работы»;
- ответах на контрольные вопросы;
- предоставлении отчета.

АППАРАТУРА И МАТЕРИАЛЫ

Для выполнения лабораторных работ необходим персональный компьютер с установленной СУБД.

УКАЗАНИЯ ПО ТЕХНИКЕ БЕЗОПАСНОСТИ

Необходимо выполнять требования по технике безопасности, предусмотренные для компьютерного класса

1. Не включать компьютер без разрешения администратора класса и преподавателя.
2. Запрещается самостоятельно вскрывать корпус ПК.
3. Не касаться одновременно экрана монитора и клавиатуры (возможен разряд повышенного электростатического потенциала).
4. Не загромождать верхние панели устройств бумагами и посторонними устройствами.
5. Не прикасаться одновременно к металлическим частям ПК и устройствам, имеющим естественное заземление.
6. Запрещается производить отключение питания во время выполнения активной задачи.
7. Не допускать попадание влаги на поверхность системного блока, монитора, клавиатуру, принтеры и другие устройства.

СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ

Перечень основной литературы:

1. Братченко, Н. Ю. (Северо-Кавказский федеральный университет). Базы данных: учебник / Н. Ю. Братченко ; Сев.-Кав. федер. ун-т. – Ставрополь : СКФУ, 2021. – 160 с. [электронный вариант]

2. Маркин, А. В. Постреляционные базы данных. MongoDB Электронный ресурс : Учебное пособие / А. В. Маркин. - Саратов : Ай Пи Ар Медиа, 2019. - 336 с. - Книга находится в премиум-версии ЭБС IPR BOOKS. - ISBN 978-5-4497-0077-3.

3. Верхолат, А. М. Проектирование структуры базы данных Электронный ресурс / Верхолат А. М., Суслов В. П. - 2-е, испр. и доп. - Санкт-Петербург : БГТУ "Военмех" им. Д.Ф. Устинова, 2018. - 65 с.

Перечень дополнительной литературы:

1. Братченко, Н. Ю. Распределенные базы данных [Электронный ресурс]: лабораторный практикум / Н. Ю. Братченко. – Электрон. текстовые данные. – Ставрополь: Северо-Кавказский федеральный университет, 2014. – 180 с. – 2227-8397. – Режим доступа: <http://www.iprbookshop.ru/63129.html>

2. Братченко, Н. Ю. (СКФУ). Базы данных. Основы работы с СУБД Oracle 10g Express Edition : Учебное пособие : для студентов высших учебных заведений / Н. Ю. Братченко ; ФГАОУ ВПО Сев.-Кав. федер. ун-т. – Ставрополь : СКФУ, 2013. – 253 с

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕ-
ЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

МЕТОДИЧЕСКИЕ УКАЗАНИЯ
для самостоятельной работы студентов по
дисциплине «Базы данных»

для студентов направления подготовки
11.03.02 Инфокоммуникационные технологии и
системы связи, Направленность (профиль)
«Интеллектуальная обработка данных в
инфокоммуникационных системах и сетях»

Ставрополь, 2026

Методические указания по организации самостоятельной работы студентов составлено в соответствии с требованиями программы дисциплины «Базы данных» для бакалавров направления подготовки 11.03.02 Инфокоммуникационные технологии и системы связи. Учебно-методическое пособие посвящено планируемой учебной работе студентов, выполняемой во внеаудиторное (аудиторное) время по заданию и при методическом руководстве преподавателя, но без его непосредственного участия, при частичном, непосредственном участии преподавателя.

Составитель: Братченко Н.Ю.

СОДЕРЖАНИЕ

1. Самостоятельная работа как важнейшая форма учебного процесса	179
2. Цели и основные задачи СРС.....	180
3. Виды самостоятельной работы	181
4. Организация СРС	181
5. Общие рекомендации по организации самостоятельной работы	183
6. Самостоятельная работа студента - необходимое звено становления исследователя.....	186
7. Методические рекомендации для студентов по отдельным формам самостоятельной работы.....	187
8. Учебно-методическое и информационное обеспечение дисциплины	193

1. Самостоятельная работа как важнейшая форма учебного процесса

Самостоятельная работа - планируемая учебная, учебно-исследовательская, научно-исследовательская работа студентов, выполняемая во внеаудиторное (аудиторное) время по заданию и при методическом руководстве преподавателя, но без его непосредственного участия (при частичном непосредственном участии преподавателя, оставляющем ведущую роль за работой студентов).

Необходимо ознакомиться с технологической картой самостоятельной работы обучающегося.

Самостоятельная работа студентов в ВУЗе является важным видом учебной и научной деятельности студента. Самостоятельная работа студентов играет значительную роль в рейтинговой технологии обучения. Обучение в ВУЗе включает в себя две, практически одинаковые по объему и взаимовлиянию части – процесса обучения и процесса самообучения. Поэтому СРС должна стать эффективной и целенаправленной работой студента.

Концепцией модернизации российского образования определены основные задачи профессионального образования - "подготовка квалифицированного работника соответствующего уровня и профиля, конкурентоспособного на рынке труда, компетентного, ответственного, свободно владеющего своей профессией и ориентированного в смежных областях деятельности, способного к эффективной работе по специальности на уровне мировых стандартов, готового к постоянному профессиональному росту, социальной и профессиональной мобильности".

Решение этих задач невозможно без повышения роли самостоятельной работы студентов над учебным материалом, усиления ответственности преподавателей за развитие навыков самостоятельной работы, за стимулирование профессионального роста студентов, воспитание творческой активности и инициативы.

К современному специалисту общество предъявляет достаточно широкий перечень требований, среди которых немаловажное значение имеет наличие у выпускников определенных способностей и умения самостоятельно добывать знания из различных источников, систематизировать полученную информацию, давать оценку конкретной финансовой ситуации. Формирование такого умения происходит в течение всего периода обучения через участие студентов в лабораторных занятиях, выполнение контрольных заданий и тестов, написание курсовых и выпускных квалификационных работ. При этом самостоятельная работа студентов играет решающую роль в ходе всего учебного процесса. Формы самостоятельной работы студентов разнообразны. По данной дисциплине «Базы данных» предусмотрены следующие:

Использование материала учебно-методического комплекса дисциплины

На первом этапе необходимо ознакомиться с обязательным минимумом содержания основной образовательной программы по изучаемой дисциплине. Далее необходимо изучить рабочую программу дисциплины, в которой рассмотрено содержание тем дисциплины лекционного курса, взаимосвязь тем лекций с лабораторными занятиями, темы и виды самостоятельной работы. По каждому виду самостоятельной работы предусмотрены определённые формы отчетности, представленные в учебной программе по дисциплине «Базы данных».

Работа с литературой

Для успешного освоения дисциплины, необходимо самостоятельно детально изучить представленные темы по рекомендуемым источникам информации, которые представлены в учебной программе. Самостоятельная работа приобщает студентов к научному творчеству, поиску и решению актуальных современных проблем.

2. Цели и основные задачи СРС

Ведущая цель организации и осуществления СРС должна совпадать с целью обучения студента – подготовкой бакалавра с высшим образованием. При организации СРС важным и необходимым условием становятся формирование умения самостоятельной работы для приобретения знаний, навыков и возможности организации учебной и научной деятельности.

Целью самостоятельной работы студентов является овладение фундаментальными знаниями, профессиональными умениями и навыками деятельности по профилю, опытом творческой, исследовательской деятельности. Самостоятельная работа студентов способствует развитию самостоятельности, ответственности и организованности, творческого подхода к решению проблем учебного и профессионального уровня.

Задачами СРС являются:

- систематизация и закрепление полученных теоретических знаний и практических умений студентов;
- углубление и расширение теоретических знаний;
- формирование умений использовать нормативную, правовую, справочную документацию и специальную литературу;
- развитие познавательных способностей и активности студентов: творческой инициативы, самостоятельности, ответственности и организованности;
- формирование самостоятельности мышления, способностей к саморазвитию, самосовершенствованию и самореализации;

- развитие исследовательских умений;
- использование материала, собранного и полученного в ходе самостоятельных занятий на семинарах, на лабораторных занятиях, при написании курсовых и выпускной квалификационной работ, для эффективной подготовки к итоговым зачетам и экзаменам.

3. Виды самостоятельной работы

В образовательном процессе высшего профессионального образовательного учреждения выделяется два вида самостоятельной работы – аудиторная, под руководством преподавателя, и внеаудиторная. Тесная взаимосвязь этих видов работ предусматривает дифференциацию и эффективность результатов ее выполнения и зависит от организации, содержания, логики учебного процесса (межпредметных связей, перспективных знаний и др.):

Аудиторная самостоятельная работа по дисциплине выполняется на учебных занятиях под непосредственным руководством преподавателя и по его заданию.

Внеаудиторная самостоятельная работа выполняется студентом по заданию преподавателя, но без его непосредственного участия.

Основными видами самостоятельной работы студентов без участия преподавателя по данной дисциплине являются:

- формирование и усвоение содержания конспекта лекций на базе рекомендованной лектором учебной литературы, включая информационные образовательные ресурсы (электронные учебники, электронные библиотеки и др.);
- подготовка к лабораторным работам, их оформление.

Основными видами самостоятельной работы студентов с участием преподавателей являются:

- текущие консультации;
- прием и защита лабораторных работ (во время проведения л/р).
- выполнение курсовой работы.

4. Организация СРС

Методика организации самостоятельной работы студентов зависит от структуры, характера и особенностей дисциплины «Базы данных», объема часов на ее изучение, вида заданий для самостоятельной работы студентов, индивидуальных качеств студентов и условий учебной деятельности.

Процесс организации самостоятельной работы студентов включает в себя следующие этапы:

- подготовительный (определение целей, составление программы, подготовка методического обеспечения, подготовка оборудования);
- основной (реализация программы, использование приемов поиска информации, усвоения, переработки, применения, передачи знаний, фиксирование результатов, самоорганизация процесса работы);
- заключительный (оценка значимости и анализ результатов, их систематизация, оценка эффективности программы и приемов работы, выводы о направлениях оптимизации труда).

Деятельность студентов по формированию и развитию навыков учебной самостоятельной работы

В процессе самостоятельной работы студент приобретает навыки самоорганизации, самоконтроля, самоуправления, саморефлексии и становится активным самостоятельным субъектом учебной деятельности.

Выполняя самостоятельную работу под контролем преподавателя студент должен:

- освоить минимум содержания, выносимый на самостоятельную работу студентов и предложенный преподавателем в соответствии с образовательными стандартами высшего образования;
- планировать самостоятельную работу в соответствии с графиком самостоятельной работы, предложенным преподавателем;
- самостоятельную работу студент должен осуществлять в организационных формах, предусмотренных учебным планом и рабочей программой преподавателя;
- выполнять самостоятельную работу и отчитываться по ее результатам в соответствии с графиком представления результатов, видами и сроками отчетности по самостоятельной работе студентов.

студент может:

сверх предложенного преподавателем (при обосновании и согласовании с ним) и минимума обязательного содержания, определяемого программой по данной дисциплине:

- самостоятельно определять уровень (глубину) проработки содержания материала;
- предлагать темы и вопросы для самостоятельной проработки;
- в рамках общего графика выполнения самостоятельной работы предлагать обоснованный индивидуальный график выполнения и отчетности по результатам самостоятельной работы;

– предлагать свои варианты организационных форм самостоятельной работы;

–использовать для самостоятельной работы методические пособия, учебные пособия, разработки сверх предложенного преподавателем перечня;

–использовать не только контроль, но и самоконтроль результатов самостоятельной работы в соответствии с методами самоконтроля, предложенными преподавателем или выбранными самостоятельно.

Самостоятельная работа студентов должна оказывать важное влияние на формирование личности будущего бакалавра, она планируется студентом самостоятельно. Каждый студент самостоятельно определяет режим своей работы и меру труда, затрачиваемого на овладение учебным содержанием по каждой дисциплине. Он выполняет внеаудиторную работу по личному индивидуальному плану, в зависимости от его подготовки, времени и других условий.

5. Общие рекомендации по организации самостоятельной работы

Основной формой самостоятельной работы студента является изучение конспекта лекций, их дополнение, рекомендованной литературы, активное участие на лабораторных занятиях. Но для успешной учебной деятельности, ее интенсификации, необходимо учитывать следующие субъективные факторы:

1. Знание школьного программного материала, наличие прочной системы знаний, необходимой для усвоения основных вузовских курсов. Это особенно важно для математических дисциплин. Необходимо отличать пробелы в знаниях, затрудняющие усвоение нового материала, от малых способностей. Затратив силы на преодоление этих пробелов, студент обеспечит себе нормальную успеваемость и поверит в свои способности.

2. Наличие умений, навыков умственного труда:

а) умение конспектировать на лекции и при работе с книгой;
б) владение логическими операциями: сравнение, анализ, синтез, обобщение, определение понятий, правила систематизации и классификации.

3. Специфика познавательных психических процессов: внимание, память, речь, наблюдательность, интеллект и мышление. Слабое развитие каждого из них становится серьезным препятствием в учебе.

4. Хорошая работоспособность, которая обеспечивается нормальным физическим состоянием. Ведь серьезное учение - это большой многосторонний и разнообразный труд. Результат обучения оценивается не количеством сообщаемой информации, а качеством ее усвоения, умением ее использовать и развитием у себя способности к дальнейшему самостоятельному образованию.

5. Соответствие избранной деятельности, профессии индивидуальным способностям. Необходимо выработать у себя умение саморегулировать свое эмоциональное состояние и устранять обстоятельства, нарушающие деловой настрой, мешающие намеченной работе.

6. Овладение оптимальным стилем работы, обеспечивающим успех в деятельности. Чередование труда и пауз в работе, периоды отдыха, индивидуально обоснованная норма продолжительности сна, предпочтение вечерних или утренних занятий, стрессоустойчивость на экзаменах и особенности подготовки к ним.

7. Уровень требований к себе, определяемый сложившейся самооценкой.

Адекватная оценка знаний, достоинств, недостатков - важная составляющая самоорганизации человека, без нее невозможна успешная работа по управлению своим поведением, деятельностью.

Одна из основных особенностей обучения в высшей школе заключается в том, что постоянный внешний контроль заменяется самоконтролем, активная роль в обучении принадлежит уже не столько преподавателю, сколько студенту.

Зная основные методы научной организации умственного труда, можно при наименьших затратах времени, средств и трудовых усилий достичь наилучших результатов.

Эффективность усвоения поступающей информации зависит от работоспособности человека в тот или иной момент его деятельности.

Работоспособность - способность человека к труду с высокой степенью напряженности в течение определенного времени. Различают внутренние и внешние факторы работоспособности.

К внутренним факторам работоспособности относятся интеллектуальные особенности, воля, состояние здоровья.

К внешним:

- организация рабочего места, режим труда и отдыха;
- уровень организации труда - умение получить справку и пользоваться информацией;

- величина умственной нагрузки.

Выдающийся русский физиолог Н. Е. Введенский выделил следующие условия продуктивности умственной деятельности:

- во всякий труд нужно входить постепенно;
- мерность и ритм работы. Разным людям присущ более или менее разный темп работы;
- привычная последовательность и систематичность деятельности;
- правильное чередование труда и отдыха.

Отдых не предполагает обязательного полного бездействия со стороны человека, он может быть достигнут простой переменой дела. В течение дня работоспособность изменяется. Наиболее плодотворным является *утреннее время (с 8 до 14 часов)*, причем максимальная работоспособность приходится на период с 10 до 13 часов, затем *послеобеденное* - (с 16 до 19 часов) и *вечернее* (с 20 до 24 часов).

Очень трудный для понимания материал лучше изучать в начале каждого отрезка времени (лучше всего утреннего) после хорошего отдыха. Через 1-1,5 часа нужны перерывы по 10 - 15 мин, через 3 - 4 часа работы отдых должен быть продолжительным - около часа.

Составной частью научной организации умственного труда является овладение техникой умственного труда.

Время, которым располагает студент для выполнения учебного плана, складывается из двух составляющих: одна из них - это аудиторная работа в вузе по расписанию занятий, другая - внеаудиторная самостоятельная работа. Задачи и материалы для самостоятельной работы выдаются во время учебных занятий по расписанию, на этих же занятиях преподаватель осуществляет контроль за самостоятельной работой, а также оказывает помощь студентам по правильной организации работы.

Самостоятельные занятия потребуют интенсивного умственного труда, который необходимо не только правильно организовать, но и стимулировать. При этом очень важно уметь поддерживать устойчивое внимание к изучаемому материалу. Выработка внимания требует значительных волевых усилий. Именно поэтому, если студент замечает, что он часто отвлекается во время самостоятельных занятий, ему надо заставить себя сосредоточиться. Подобную процедуру необходимо проделывать постоянно, так как это является тренировкой внимания. Устойчивое внимание появляется тогда, когда человек относится к делу с интересом.

Следует правильно организовать свои занятия по времени: 50 минут - работа, 5-10 минут - перерыв; после 3 часов работы перерыв - 20-25 минут. Иначе нарастающее утомление повлечет неустойчивость внимания. Очень существенным фактором, влияющим на повышение умственной работоспособности, являются систематические занятия физической культурой. Организация активного отдыха предусматривает чередование умственной и физической деятельности, что полностью восстанавливает работоспособность человека.

6. Самостоятельная работа студента - необходимое звено становления исследователя

Прогресс науки и техники, информационных технологий приводит к значительному увеличению научной информации, что предъявляет более высокие требования не только к моральным, нравственным свойствам человека, но и в особенности, постоянно возрастающие требования в области образования – обновление, модернизация общих и профессиональных знаний, умений специалиста.

Всякое образование должно выступать как динамический процесс, присущий человеку и продолжающийся всю его жизнь. Овладение научной мыслью и языком науки является необходимой составляющей в самоорганизации будущего специалиста исследователя. Под этим понимается не столько накопление знаний, сколько овладение научно обоснованными способами их приобретения. В этом, вообще говоря, состоит основная задача вуза.

Специфика вузовского учебного процесса, в организации которого самостоятельной работе студента отводятся все больше места, состоит в том, что он является как будто бы последним и самым адекватным звеном для реализации этой задачи. Ибо во время учебы в вузе происходит выработка стиля, навыков учебной (познавательной) деятельности, рациональный характер которых будет способствовать постоянному обновлению знаний высококвалифицированного выпускника вуза.

Однако до этого пути существуют определенные трудности, в частности, переход студента от синтетического процесса обучения в средней школе, к аналитическому в высшей. Это связано как с новым содержанием обучения (расширение общего образования и углубление профессиональной подготовки), так и с новыми, неизвестными до сих пор формами: обучения (лекции, семинары, лабораторные занятия и т.д.). Студент получает не только знания, предусмотренные программой и учебными пособиями, но он также должен познакомиться со способами приобретения знаний так, чтобы суметь оценить, что мы знаем, откуда мы это знаем и как этого знания мы достигли. Ко всему этому приходят через собственную самостоятельную работу.

Это и потому, что самостоятельно приобретенные знания являются более оперативными, они становятся личной собственностью, а также мотивом поведения, развивают интеллектуальные черты, внимание, наблюдательность, критичность, умение оценивать. Роль преподавателя в основном заключается в руководстве накопления знаний (по отношению к первокурсникам), а в последующие годы учебы, на старших курсах, в совместном установлении проблем и заботе о самостоятельных поисках студента, а также контролирования за их деятельностью. Отметим, что нельзя ограничиваться только приобретением знаний предусмотренных программой изучаемой дисциплины, надо постоянно углублять полученные знания, сосредотачивая их на какой-нибудь узкой определенной области, соответствующей интересам студента. Углубленное изучение всех предметов, предусмотренных программой, на практике является возможным, и хорошая организация работы позволяет экономить время, что создает условия для глубокого, систематического, заинтересованного изучения самостоятельно выбранной студентом темы.

Конечно, все советы, примеры, рекомендации в этой области, даваемые преподавателем, или определенными публикациями, или другими источниками, не гарантируют никакого успеха без проявления собственной активности в этом деле, т. е. они не дают готовых рецептов, а должны способствовать анализу собственной работы, ее целей, организации в соответствии с индивидуальными особенностями. Учитывая личные возможности, существующие условия жизни и работы, навыки, на основе этих рекомендаций, возможно, выработать индивидуально обоснованную совокупность методов, способов, найти свой стиль или усовершенствовать его, чтобы изучив определенный материал, иметь время оценить его значимость, пригодность и возможности его применения, чтобы, в конечном счете, обеспечить успешность своей учебы с будущей профессиональной деятельности

7. Методические рекомендации для студентов по отдельным формам самостоятельной работы

Система вузовского обучения подразумевает значительно большую самостоятельность студентов в планировании и организации своей деятельности.

Работа с литературой

При работе с литературой необходимо подобрать литературу, научиться правильно ее читать, вести записи. Для подбора литературы в библиотеке используются алфавитный и систематический каталоги.

Важно помнить, что рациональные навыки работы с книгой - это всегда большая экономия времени и сил.

Правильный подбор учебников рекомендуется преподавателем, читающим лекционный курс. Необходимая литература может быть также указана в методических разработках по данному курсу.

Изучая материал по учебнику, следует переходить к следующему вопросу только после правильного уяснения предыдущего, описывая на бумаге все выкладки и вычисления (в том числе те, которые в учебнике опущены или на лекции даны для самостоятельного вывода).

При изучении любой дисциплины большую и важную роль играет самостоятельная индивидуальная работа.

Особое внимание следует обратить на определение основных понятий курса. Студент должен подробно разбирать примеры, которые поясняют такие определения, и уметь строить аналогичные примеры самостоятельно. Нужно добиваться точного представления о том, что изучаешь. Полезно составлять опорные конспекты. При изучении материала по учебнику полезно в тетради (на специально отведенных полях) дополнять конспект лекций. Там же следует отмечать вопросы, выделенные студентом для консультации с преподавателем.

Выводы, полученные в результате изучения, рекомендуется в конспекте выделять, чтобы они при перечитывании записей лучше запоминались.

Опыт показывает, что многим студентам помогает составление листа опорных сигналов, содержащего важнейшие и наиболее часто употребляемые формулы и понятия. Такой лист помогает запомнить формулы, основные положения лекции, а также может служить постоянным справочником для студента.

Различают два вида чтения; первичное и вторичное. *Первичное* - это внимательное, неторопливое чтение, при котором можно остановиться на трудных местах. После него не должно остаться ни одного непонятого слова. Содержание не всегда может быть понятно после первичного чтения.

Задача *вторичного* чтения полное усвоение смысла целого (по счету это чтение может быть и не вторым, а третьим или четвертым).

Чтение научного текста является частью познавательной деятельности. Ее цель – извлечение из текста необходимой информации. От того на сколько осознанна читающим собственная внутренняя установка при обращении к печатному слову (найти нужные сведения, усвоить информацию полностью или частично, критически проанализировать материал и т.п.) во многом зависит эффективность осуществляемого действия.

Выделяют **четыре основные установки в чтении научного текста:**

1. информационно-поисковый (задача – найти, выделить искомую информацию)

2. усваивающая (усилия читателя направлены на то, чтобы как можно полнее осознать и запомнить как сами сведения излагаемые автором, так и всю логику его рассуждений)

3. аналитико-критическая (читатель стремится критически осмыслить материал, проанализировав его, определив свое отношение к нему)

4. творческая (создает у читателя готовность в том или ином виде – как отправной пункт для своих рассуждений, как образ для действия по аналогии и т.п. – использовать суждения автора, ход его мыслей, результат наблюдения, разработанную методику, дополнить их, подвергнуть новой проверке).

С наличием различных установок обращения к научному тексту связано существование и нескольких **видов чтения**:

1. библиографическое – просматривание карточек каталога, рекомендательных списков, сводных списков журналов и статей за год и т.п.;

2. просмотрное – используется для поиска материалов, содержащих нужную информацию, обычно к нему прибегают сразу после работы со списками литературы и каталогами, в результате такого просмотра читатель устанавливает, какие из источников будут использованы в дальнейшей работе;

3. ознакомительное – подразумевает сплошное, достаточно подробное прочтение отобранных статей, глав, отдельных страниц, цель – познакомиться с характером информации, узнать, какие вопросы вынесены автором на рассмотрение, провести сортировку материала;

4. изучающее – предполагает доскональное освоение материала; в ходе такого чтения проявляется доверие читателя к автору, готовность принять изложенную информацию, реализуется установка на предельно полное понимание материала;

5. аналитико-критическое и творческое чтение – два вида чтения близкие между собой тем, что участвуют в решении исследовательских задач. Первый из них предполагает направленный критический анализ, как самой информации, так и способов ее получения и подачи автором; второе – поиск тех суждений, фактов, по которым или в связи с которыми, читатель считает нужным высказать собственные мысли.

Из всех рассмотренных видов чтения основным для студентов является изучающее – именно оно позволяет в работе с учебной литературой накапливать знания в различных областях. Вот почему именно этот вид чтения в рамках учебной деятельности должен быть освоен в первую очередь. Кроме того, при овладении данным видом чтения формируются основные приемы, повышающие эффективность работы с научным текстом.

Основные виды систематизированной записи прочитанного:

1. Аннотирование – предельно краткое связное описание просмотренной или прочитанной книги (статьи), ее содержания, источников, характера и назначения;
2. Планирование – краткая логическая организация текста, раскрывающая содержание и структуру изучаемого материала;
3. Тезирование – лаконичное воспроизведение основных утверждений автора без привлечения фактического материала;
4. Цитирование – дословное выписывание из текста выдержек, извлечений, наиболее существенно отражающих ту или иную мысль автора;
5. Конспектирование – краткое и последовательное изложение содержания прочитанного.

Конспект – сложный способ изложения содержания книги или статьи в логической последовательности. Конспект аккумулирует в себе предыдущие виды записи, позволяет всесторонне охватить содержание книги, статьи. Поэтому умение составлять план, тезисы, делать выписки и другие записи определяет и технологию составления конспекта.

Методические рекомендации по составлению конспекта:

1. Внимательно прочитайте текст. Уточните в справочной литературе непонятные слова. При записи не забудьте вынести справочные данные на поля конспекта.
2. Выделите главное, составьте план.
3. Кратко сформулируйте основные положения текста, отметьте аргументацию автора.
4. Законспектируйте материал, четко следуя пунктам плана. При конспектировании старайтесь выразить мысль своими словами. Записи следует вести четко, ясно.
5. Грамотно записывайте цитаты. Цитируя, учитывайте лаконичность, значимость мысли.

В тексте конспекта желательно приводить не только тезисные положения, но и их доказательства. При оформлении конспекта необходимо стремиться к емкости каждого предложения. Мысли автора книги следует излагать кратко, заботясь о стиле и выразительности написанного. Число дополнительных элементов конспекта должно быть логически обоснованным, записи должны распределяться в определенной последовательности, отвечающей логической структуре произведения. Для уточнения и дополнения необходимо оставлять поля.

Овладение навыками конспектирования требует от студента целеустремленности, повседневной самостоятельной работы.

Лабораторные занятия

Следует подчеркнуть, что только после усвоения лекционного материала с определенной точки зрения (а именно с той, с которой он излагается на лекциях) он будет закрепляться на лабораторных занятиях как в результате обсуждения и анализа лекционного материала, так и с помощью выполнения заданий лабораторной работы, поставленных задач. При этих условиях студент не только хорошо усвоит материал, но и научится применять его на практике, а также получит дополнительный стимул (и это очень важно) для активной проработки лекции.

При самостоятельном выполнении заданий нужно обосновывать каждый этап решения, исходя из теоретических положений курса. Если студент видит несколько путей решения проблемы (задачи), то нужно сравнить их и выбрать самый рациональный. Полезно до начала вычислений составить краткий план решения проблемы (задачи). Решение задач или примеров следует излагать подробно, вычисления располагать в строгом порядке, отделяя вспомогательные вычисления от основных. Решения при необходимости нужно сопровождать комментариями, схемами, чертежами и рисунками.

Следует помнить, что выполнение каждого учебного задания должно доводиться до окончательного логического ответа, которого требует условие, и по возможности с выводом. Выполнение заданий данного типа нужно продолжать до приобретения твердых навыков в их решении.

Самопроверка

После изучения определенной темы по записям в конспекте и учебнику, а также решения достаточного количества соответствующих заданий на лабораторных занятиях и самостоятельно студенту рекомендуется, используя лист опорных материалов, воспроизвести по памяти определения, выводы формул, формулировки основных положений и доказательств.

В случае необходимости нужно еще раз внимательно разобраться в материале.

Иногда недостаточность усвоения того или иного вопроса выясняется только при изучении дальнейшего материала. В этом случае надо вернуться назад и повторить плохо усвоенный материал. Важный критерий усвоения теоретического материала - умение решать задачи или пройти тестирование по пройденному материалу. Однако следует помнить, что правильное решение за-

дачи может получиться в результате применения механически заученных формул без понимания сущности теоретических положений.

Консультации

Если в процессе самостоятельной работы над изучением теоретического материала или при выполнении заданий у студента возникают вопросы, разрешить которые самостоятельно не удастся, необходимо обратиться к преподавателю для получения у него разъяснений или указаний. В своих вопросах студент должен четко выразить, в чем он испытывает затруднения, характер этого затруднения. За консультацией следует обращаться и в случае, если возникнут сомнения в правильности ответов на вопросы самопроверки.

Подготовка к зачету по дисциплине «Базы данных».

Систематическая подготовка к занятиям в течение семестра позволит использовать время экзаменационной сессии для систематизации знаний.

Текущая аттестация студентов проводится преподавателями, ведущими лабораторные занятия по дисциплине в следующих формах: собеседование при защите лабораторной работы с предоставлением письменного отчета. Допуск к лабораторным работам происходит при наличии у студентов печатного варианта отчета. Защита отчета проходит в форме собеседования.

Максимальную оценку студент получает, если он полностью выполнил задания по лабораторной работе, подготовил отчет и активно участвует в работе, владеет материалом, умеет логично и четко излагать мысли, творчески подходит к решению задач, показывает самостоятельность мышления.

Основанием для снижения оценки являются:

- не качественное выполнение отчета, не соблюдение требований, отраженных в методических указаниях;
- не верное выполнение поставленных задач в лабораторных работах.
- допущение ошибок и незнание этапов проектирования БД;
- не верное применение команд SQL.

В качестве вопросов к лабораторным работам могут быть следующие:

Работа с СУБД

1. В чем состоят особенности создания таблиц базы данных средствами WEB-интерфейса?
2. Что такое WEB-интерфейс?
3. Каковы возможности WEB-интерфейса?
4. Как изменить содержание таблицы?

Управление таблицами и данными

1. Какая команда служит для создания таблицы?
2. Какая команда показывает структуру таблицы?
3. Какой командой добавляются записи в таблицу?
4. Какая команда служит для переименования таблицы?

5. Какой командой добавляется новый столбец в таблицу?
6. Какой командой осуществляю выборку данных из таблицы?
7. Какая команда удаляет таблицу из БД?
8. Какую позицию в таблице занимает столбец после добавления?
9. Что в «SELECT product_name, product_price * 1.15 FROM hase;» является выражением?
10. Какая команда удаляет таблицу из БД?

Более сложные манипуляции с данными

1. Какие операторы служат для вывода диапазона значений?
2. Какие операторы используются для вывода записей из диапазона значений.
3. Для чего служит оператор «<>»?
4. Для чего нужен оператор IN?
5. Для чего нужен оператор %?
6. С помощью какого оператора можно найти NULL значения?
8. Команда для сортировки данных.
9. Команда для выборки уникальных значений.
10. Команда для обновления записей в таблице.
11. Команда для удаления записей в таблице.
12. Команда ROLLBACK.
13. Для чего необходима комбинация операторов IS NOT NULL?
14. Для чего нужны точки сохранения?

Управление SQL*Plus

1. Синтаксис команды COLUMN для чисел.
2. Для чего нужны файлы сценариев SQL?
3. Синтаксис команды COLUMN для форматирования заголовков столбцов.
4. Команда для создания файла буферизации выходных данных.
5. Синтаксис команды ACCEPT.
6. Для чего нужны переменные подстановки?
7. Какие отличия переменных подстановки для чисел от текста?

Встроенные функции SQL

1. Для чего предназначена системная переменная SYSDATE?
2. Системные переменные USER и USERENV.
3. Функция ROUND и TRUNC.
4. Функции UPPER и LOWER.
5. Функция LENGTH.
6. Функция SUBSTR.
7. Функция INSTR..
8. Функция DECODE.
9. Комментарии в SQL-сценариях.
10. Функция SUM.
11. Функция COUNT.
12. Функции MAX и MIN.

13. Оператор GROUPBY.
14. Что представляют собой системные переменные?
15. Для чего нужна конструкция HAVING?

Связи между таблицами

1. Что представляет собой связь?
2. Для чего применяется первичный ключ?
3. С какой целью создается внешний ключ?
4. Могут ли значения первичного и внешнего ключа быть различными?
5. Синтаксис для создания ограничения внешнего ключа.
6. Синтаксис оператора SELECT для отображения данных из нескольких таблиц.
7. Что обозначает ключевое слово JOIN?

Написание подзапросов

1. Что такое подзапрос?
2. Функция IN.
3. В каких операторах можно использовать подзапросы?
4. Какие проблемы можно решить с помощью подзапросов?
5. Что такое многострочный подзапрос?

Перенос данных между таблицами. Последовательности. Синонимы.

1. Перенос данных с помощью Insert
2. Создание новой таблицы на основе уже существующей
3. Создание таблицы на основе уже имеющихся.
4. Как соединить записи из двух таблиц?

Представления.

1. Что такое представления?
2. Как создаются представления?
3. Как удалить представления?
1. Возможно ли осуществить вывод первых N записей?
2. Какие способы создания представлений известны?

Последовательности

1. Что представляют собой последовательности и синонимы. Как они реализуются средствами PL/SQL?
2. Параметры CURRVAL и NEXTVAL.
3. Параметр INCREMENT BY.
4. Что такое последовательность?
5. Синтаксис команды для создания последовательности.

Синонимы

1. Команда создания синонима.
2. Команда удаления синонима.

3. 1. Что такое синонимы?
4. 2. Команда создания синонима.

PL/SQL, хранимые процедуры, функции

1. Что такое курсоры?
2. Каково назначение команд OPEN, FETCH и CLOSE?
3. Что такое хранимые процедуры?
4. Что такое хранимые функции?
5. Чем отличаются хранимые функции от процедур?
6. Структура блока PL/SQL.
7. Как можно выполнить процедуру из SQL*Plus?
8. Синтаксис оператора IF.
9. Запись в языке PL/SQL.
10. Синтаксис оператора LOOP, WHILE, FOR.
11. Конструкция WHERECURRENTOF.

Пакеты PL/SQL и триггеры

1. Какие существуют отличия между триггерами и процедурами PL/SQL?
2. Что необходимо учитывать при создании триггеров?
3. Для чего необходимы пакеты PL/SQL?
4. Что такое триггер?
5. Какие типы триггеров существуют?

Индексы и ограничения

1. Индексы и ограничения
2. Что такое индекс?
3. Команда создания индекса.
4. Что такое ограничения?
5. Какие виды ограничений известны?
6. Каково назначение ограничений?
7. Ограничение NOTNULL, UNIQUE и CHECK.
8. Синтаксис команд для изменения и удаления существующих ограничений.
9. Каковы команды по работе с индексами таблиц БД

Критерии оценивания результатов самостоятельной работы по курсу «Базы данных» приведены в Фонде оценочных средств по дисциплине.

8. Учебно-методическое и информационное обеспечение дисциплины

Основная литература	1. Братченко, Н. Ю. (Северо-Кавказский федеральный университет). Базы данных: учебник. / Н. Ю. Братченко ; Сев.-Кав. федер. ун-т. – Ставрополь : СКФУ, 2021. – 160 с. [электронный вариант] 2. Маркин, А. В. Постреляционные базы данных. MongoDB Электронный ресурс : Учебное пособие / А. В. Маркин. - Саратов : Ай Пи Ар Медиа, 2019. - 336 с. - Книга
---------------------	--

	находится в премиум-версии ЭБС IPR BOOKS. - ISBN 978-5-4497-0077-3. 3. Верхолат, А. М. Проектирование структуры базы данных Электронный ресурс / Верхолат А. М., Суслов В. П. - 2-е, испр. и доп. - Санкт-Петербург : БГТУ "Военмех" им. Д.Ф. Устинова, 2018. - 65 с.
Дополнительная литература	1. Братченко, Н. Ю. Распределенные базы данных [Электронный ресурс]: лабораторный практикум / Н. Ю. Братченко. – Электрон. текстовые данные. – Ставрополь: Северо-Кавказский федеральный университет, 2014. – 180 с. – 2227-8397. – Режим доступа: http://www.iprbookshop.ru/63129.html 2. Братченко, Н. Ю. (СКФУ). Базы данных. Основы работы с СУБД Oracle 10g Express Edition : Учебное пособие : для студентов высших учебных заведений / Н. Ю. Братченко ; ФГАОУ ВПО Сев.-Кав. федер. ун-т. – Ставрополь : СКФУ, 2013. – 253 с.