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

Кафедра информатики

А. Н. Тараканов

МОДЕЛИ ДАННЫХ И СУБД

Методическое пособие
к лабораторным работам
для студентов специальности I-31 03 04 «Информатика»
всех форм обучения

Минск 2008

УДК 681.3.016 (075.8)
ББК 32.973.26 – 018.2 я 73
Т 19

Р е ц е н з е н т

нач. сектора информационных технологий СП ЗАО «Международный деловой альянс», канд. физ.-мат. наук, доц. И. И. Пилецкий

Тараканов, А. Н.

Т 19 Модели данных и СУБД : метод. пособие к лаб. работам для студ. спец. I-31 03 04 «Информатика» всех форм обуч. / А. Н. Тараканов. – Минск : БГУИР, 2008. – 56 с. : ил.

ISBN 978-985-488-212-3

Методическое пособие составлено в соответствии с рабочей программой курса «Модели данных и СУБД». В него включены основные теоретические сведения и рекомендации по выполнению лабораторных работ. Приводятся примеры выполнения заданий.

Методическое пособие предназначено для студентов специальности «Информатика», может быть рекомендовано студентам технических специальностей для изучения основных конструкций и принципов применения языка SQL.

УДК 681.3.016 (075.8)
ББК 32.973.26 – 018.2 я 73

ISBN 978-985-488-212-3

СОДЕРЖАНИЕ

Лабораторная работа №1
«Основы SQL*Plus»

Цель работы

1. Ознакомление с возможностями среды SQL*Plus.
2. Ознакомление с основными командами среды SQL*Plus.
3. Использование SQL*Plus для выполнения и редактирования запросов SQL.

Теоретические сведения

SQL и SQL*Plus

SQL – это командный язык для работы с Oracle Server с помощью любой утилиты или приложения. Oracle SQL содержит многочисленные расширения по сравнению со стандартом.

Когда пользователь вводит SQL-предложение, оно сохраняется в части памяти, называемой SQL-буфером, и остается там до того, как будет введено следующее предложение.

SQL*Plus может распознавать SQL-предложения и направлять их на Oracle Server для обработки. Также SQL*Plus содержит свой встроенный командный язык.

SQL*Plus позволяет пользователю непосредственно вводить и исполнять SQL-предложения. Для больших объемов кода предусмотрен ввод SQL из файлов. В SQL*Plus также присутствует простой редактор для модификации предложений SQL. Результаты обработки запросов могут быть отформатированы средствами SQL*Plus в простейшие отчеты.

SQL*Plus позволяет менять настройки среды и получать доступ как к локальным, так и удаленным базам данных.

Командный язык SQL*Plus не предоставляет возможностей для непосредственной работы с данными, хотя с его помощью и можно просмотреть структуру таблиц. Команды SQL*Plus вводятся по одной и выполняются немедленно после ввода и не сохраняются в буфере. Если команду нужно разделить на несколько строк, используется символ переноса (-). Команды, как правило, можно сокращать.

Команды SQL*Plus

Основными функциями SQL*Plus являются:

- выполнение SQL-предложений для получения, изменения, добавления и удаления информации;
- форматирование, выполнение вычислений, хранение и печать результатов запросов в форме отчетов;
- создание скриптовых файлов для хранения и повторного использования SQL-предложений.

Команды SQL*Plus можно разделить на несколько основных категорий, приведенных в табл. 1.

Таблица 1

Категории команд SQL*Plus

Категория	Назначение
Изменение окружения	Влияет на общее поведение SQL-предложений во время сессии
Форматирование	Форматирование результатов запроса
Работа с файлами	Запись, чтение и выполнение файлов скриптов
Выполнение	Отправка предложений SQL из SQL-буфера на Oracle Server
Редактирование	Модификация SQL-предложений в буфере
Взаимодействие	Позволяет создавать и передавать для выполнения SQL-предложения, выводить значения переменных и сообщения
Прочие	Различные команды для соединения с базой данных, управления окружением SQL*Plus и отображения определений столбцов

Вход в SQL*Plus

Запуск SQL*Plus может осуществляться в зависимости от особенностей операционной системы как с помощью графического интерфейса, так и из командной строки.

Запуск с помощью графического интерфейса:

1. Выбрать SQL*Plus в меню кнопки Пуск.

2. В появившемся диалоге заполнить поля username, password и database и подтвердить ввод нажатием кнопки ok.

Запуск из командной строки:

Ввести команду запуска в следующем формате:

```
sqlplus [username[/password [@database]]]
```

где

username – имя пользователя в базе данных;
password – пароль пользователя в базе данных;
@database – строка соединения с базой данных.

Указание. Для сохранности пароля рекомендуется не вводить его в командной строке операционной системы, где он не скрывается. Вместо этого, введя в командной строке только имя пользователя, следует ввести пароль по приглашению SQL*Plus на ввод пароля.

После успешного входа в SQL*Plus будет отображена строка, похожая на приведенную ниже:

```
SQL*Plus : Release 8.0.3.0.0 - Production on Mon Oct 06
16:03:43 1997
(c) Copyright 1997 Oracle Corporation. All rights
reserved.
```

Вывод структуры таблиц

В SQL*Plus можно просмотреть структуру таблицы, используя команду DESCRIBE. Результатом выполнения команды будет информация о названиях и типах данных столбцов, а также о том, могут ли столбцы содержать пустые значения.

Синтаксис:

DESC[RIBE] *имя_таблицы*

где

имя_таблицы – название любой существующей таблицы, вида, или синонима, доступного пользователю.

```
DESCRIBE STUDENT
Name                 Null?          Type
-----------------    --------       ------------
STUDID               NOT NULL       NUMBER(2)
NAME                                VARCHAR2(14)
GROUPNAME                           VARCHAR2(13)
```

Приведенный пример демонстрирует работу команды DESCRIBE. *Name* содержит имя столбца, *Null?* показывает, должен ли столбец обязательно содержать данные (на это указывает NOT NULL), *Type* показывает тип данных в столбце.

Типы данных

NUMBER(*p*,*s*)	Числовое значение, *p* – максимальное количество цифр в числе, *s* – количество цифр после точки
VARCHAR2(*s*)	Символьное значение переменной длины с максимальной длиной *s*
DATE	Значение времени и даты между 1 января 4712 до н. э. и 31 декабря 9999 н.э.
CHAR(*s*)	Символьное значение фиксированной длины *s*

Команды редактирования

Если при вводе предложений SQL нажата клавиша [Return], в то время как ввод предложения еще не окончен, SQL*Plus выдаст приглашение на ввод с

номером следующей строки. Для окончания ввода в буфер SQL используется либо один из символов окончания ввода (; или /), либо двойное нажатие [Return]. Затем будет выведено приглашение на ввод нового предложения.

Для редактирования содержимого буфера SQL используются команды SQL*Plus, приведенные в табл. 2.

Таблица 2

Команды для редактирования SQL-буфера

Команда	**Описание**
A[PPEND] *text*	Добавляет текст в конец текущей строки
C[HANGE] / *old* / *new*	Заменяет текст *old* на текст *new* в текущей строке
C[HANGE] / *text* /	Удаляет текст *text* в текущей строке
CL[EAR] BUFF[ER]	Удаляет все строки из буфера SQL
DEL	Удаляет текущую строку
DEL *n*	Удаляет строку с номером *n*
DEL *m n*	Удаляет строки с номерами от *m* до *n*
I[NPU T]	Вставляет неопределенное число строк
I[NPUT] *text*	Вставляет строку, состоящую из *text*
L[IST]	Выводит содержимое SQL-буфера
L[IST] *n*	Выводит строку с номером *n*
L[IST] *m n*	Выводит диапазон строк (от *m* до *n)*
R[UN]	Отображает и выполняет текущее предложение из SQL-буфера
n	Делает строку с номером *n* текущей
n text	Заменяет строку с номером *n* на текст *text*
0 *text*	Добавляет строку перед первой строкой

Команды работы с файлами

Команды SQL*Plus, предназначенные для работы с файлами, приводятся в табл. 3.

Таблица 3

Команды для работы с файлами

Команда	**Описание**
1	**2**
SAV[E] *filename* [.ext] [REP[LACE]|APP[END]]	Сохраняет текущее содержимое SQL-буфера в файл. Для записи в конец существующего файла используется APPEND; при использовании REPLACE существующий файл перезаписывается. По умолчанию файл имеет расширение .sql
GET *filename* [.ext]	Записывает содержимое сохраненного ранее файла в буфер SQL. По умолчанию файл имеет расширение .sql

Окончание табл. 3

1	2
STA[RT] *filename* [.ext]	Запускает ранее сохраненный файл
@ filename	Запускает ранее сохраненный файл (аналогична команде START)
ED[IT]	Запускает редактор и сохраняет содержимое буфера в файл *afiedt.buf*
ED[IT] [*filename*[.ext]]	Запускает редактор для редактирования содержимого указанного файла
SPO[OL] [*filename*[.ext]| OFF|OUT]	Сохраняет результаты запроса в файле. OFF закрывает файл спулинга. OUT закрывает файл и отправляет результаты на принтер
EXIT	Выход из SQL*Plus

Лабораторная работа №2
«SQL: простейшие запросы»

Цель работы

1. Изучение структуры запроса SELECT.
2. Получение практических навыков в использовании следующих базовых возможностей:
   1) выборка данных из одной таблицы;
   2) использование арифметических выражений и работа с символьными значениями в запросах;
   3) сортировка результатов запроса.

Теоретические сведения

Простейшее предложение SELECT

Предложение SELECT служит для получения информации из базы данных. SELECT позволяет производить с реляционной базой данных следующие операции:

1. *Выборка.* Эта возможность используется для определения строк таблицы, которую вернет запрос. Для выбора возвращаемых строк могут использоваться различные условия.

2. *Проекция.* Эта операция определяет, какие столбцы будут присутствовать в таблице, возвращенной по запросу. В запросе перечисляются нужные столбцы.

3. *Соединение.* Возможности SQL позволяют объединять в результате запроса информацию из разных таблиц по общему для этих таблиц столбцу.

В своей простейшей форме предложение SELECT должно включать следующие части:

SELECT – указывает столбцы, которые необходимо отобразить;

FROM – указывает таблицу, содержащую столбцы, перечисленные в части SELECT.

Синтаксис такого предложения выглядит следующим образом:

```
SELECT  [DISTINCT] {*, столбец [псевдоним], … }
FROM таблица;
```

Рекомендации по написанию SQL-предложений:

- SQL-предложения не чувствительны к регистру;
- SQL-предложения могут быть записаны на одной или нескольких строках;
- ключевые слова нельзя разрывать на несколько строк или сокращать;
- отдельные части предложения обычно размещаются в отдельных строках для облегчения чтения и редактирования предложения;
- для улучшения читаемости кода могут использоваться табуляция и отступы;
- ключевые слова обычно набираются в верхнем регистре, а остальные слова, такие как названия таблиц и столбцов, – в нижнем регистре.

Выбор столбцов

Приведенная выше простейшая форма предложения SELECT позволяет реализовать операцию проекции.

Выборка всех столбцов таблицы. Для того чтобы получить выборку всех столбцов некоторой таблицы, нужно в запросе после ключевого слова SELECT поставить символ *.

Тот же самый результат можно получить, перечислив после слова SELECT все столбцы таблицы:

Выборка всех строк из произвольных столбцов. Для того чтобы результат запроса содержал только некоторые нужные столбцы, необходимо перечислить названия этих столбцов через запятую после ключевого слова SELECT.

При этом порядок столбцов при отображении результата запроса определяется порядком перечисления столбцов в списке.

Арифметические выражения

Иногда может потребоваться изменить способ отображения данных, произвести с данными таблицы некоторые вычисления. Для этого используются арифметические выражения. Арифметическое выражение может содержать имена столбцов, числовые константы и арифметические операторы. В SQL доступны операторы +, –, *, / – сложение, вычитание, умножение и

деление соответственно. Арифметические операторы допускается использовать в любой части SQL-предложения, за исключением секции FROM.

Если выражение содержит несколько арифметических операторов, то выполнение умножения и деления имеет приоритет перед сложением и вычитанием. В рамках равного приоритета операции выполняются слева направо. Для изменения порядка вычислений могут использоваться скобки.

NULL-значения

Если в строке отсутствует информация в каком-либо поле, такое значение называют пустым, или null-значением. Пустое значение – это такое значение, которое недоступно, не присвоено, неизвестно или неприменимо. Понятие пустого значения отличается от нуля или пробела. Нуль является конкретным, определенным числом, а пробел – символом. Пустое значение могут содержать столбцы с любым типом данных при условии, что столбец не был определен как NOT NULL или PRIMARY KEY при создании таблицы.

Если в каком-либо столбце, присутствующем в арифметическом выражении, встречается значение null, то результат всего выражения также будет равен null. Здесь хорошо проявляются различия пустого значения и нуля. Если произвести деление на нуль, то возникнет ошибка деления. В то же время, если выполнить деление на null-значение, то результат будет null, или неизвестным.

Псевдонимы столбцов

При отображении результата запроса SQL*Plus обычно использует имена столбцов в качестве заголовков. Во многих случаях такие заголовки могут быть недостаточно описательными и, следовательно, трудными для понимания. Заголовок столбца можно изменить, используя псевдоним столбца. Псевдоним указывается в списке SELECT через пробел после имени столбца. По умолчанию псевдонимы печатаются в верхнем регистре. Если псевдоним содержит пробелы, специальные символы (например # или $) или чувствителен к регистру, его нужно поместить в двойные кавычки (" ").

Ключевое слово AS между именем столбца и псевдонимом необязательно, но делает чтение предложения более удобным.

Указание. В SQL-предложении псевдонимы могут использоваться в секциях SELECT и ORDER BY. В секции WHERE использование псевдонимов не допускается. Обе эти особенности соответствуют стандарту ANSI SQL 92.

```
SELECT   fio [AS] "ФИО"
FROM     student;
```

Оператор конкатенации

Существует возможность соединять столбцы с другими столбцами, арифметическими выражениями или константами, получая при этом

символьное значение. Для этого используется оператор конкатенации (||). Значения столбцов, соединенных оператором, объединяются в одно строковое значение и выводятся одним столбцом.

```
SELECT fio||’ ’||passport AS "ФИО и номер паспорта"
FROM student;
```

Литералы

Литерал – это любой символ, выражение или число, включенное в список SELECT, которое не является именем столбца или псевдонимом. Он печатается для каждой возвращенной строки. Строки текста произвольного формата могут включаться в результат запроса и обрабатываются так же, как столбцы в списке SELECT. Литералы, состоящие из символов или дат, должны заключаться в одиночные кавычки (‘ ‘); числовые литералы – нет.

```
SELECT fio||’, паспорт:’||
    passport AS "ФИО и номер паспорта"
FROM student;
```

```
ФИО и номер паспорта
--------------------------------
Иванов И.И., паспорт: АА1234567
...
14 rows selected.
```

Повторяющиеся строки

По умолчанию SQL*Plus отображает результаты запроса, не устраняя повторяющиеся строки.

```
SELECT group_id
FROM student

GROUP_ID
---------
1
1
2
3
2
3
...
14 rows selected.
```

Чтобы устранить повторяющиеся строки в результате запроса, следует использовать ключевое слово DISTINCT непосредственно после ключевого слова SELECT. После DISTINCT можно указывать несколько столбцов. DISTINCT влияет на все выбранные столбцы, предотвращая появление в результате запроса идентичных строк.

```
SELECT DISTINCT group_id
FROM student

GROUP_ID
---------
1
2
3

3 rows selected.
```

Секция ORDER BY

По умолчанию порядок строк, возвращаемых запросом, не определен. Секция ORDER BY используется для сортировки строк. Если секция ORDER BY используется, ее необходимо указывать последней. При сортировке могут использоваться выражения или псевдонимы.

```
SELECT выражение
FROM таблица
[WHERE условие]
[ORDER BY {столбец, выражение} [ASC|DESC]];
```

где

ASC – сортировка строк по возрастанию (используется по умолчанию);

DESC – сортировка строк по убыванию.

Если секция ORDER BY не используется, то порядок строк не определен. По умолчанию строки сортируются по возрастанию:

- для числовых значений первыми отображаются меньшие (1–999);
- для дат первыми отображаются наиболее ранние (01-JAN-92 перед 01-JAN-95);
- символы сортируются в алфавитном порядке;
- пустые значения отображаются последними для возрастающих последовательностей и первыми – для убывающих.

Результаты запроса можно сортировать по нескольким столбцам. Ограничением является только количество столбцов в рассматриваемой

таблице. Имена столбцов указываются в секции ORDER BY через запятую. Если необходимо поменять порядок сортировки для какого-либо столбца, после его имени указывается ключевое слово DESC. Сортировку можно осуществлять по любым столбцам таблицы, даже если они не перечислены в секции SELECT. Порядок перечисления столбцов в ORDER BY определяет порядок, в котором эти столбцы используются для сортировки результата.

Лабораторная работа №3
«SQL: DDL»

Цель работы

1. Изучение основных возможностей Data Definition Language (DDL)
2. Получение практических навыков выполнения с его помощью следующих операций:
    1) создание и удаление таблиц,
    2) изменение таблиц.

Теоретические сведения

Создание таблицы

```
CREATE TABLE table
(
 column datatype [DEFAULT expr] [,
 column datatype [DEFAULT expr]]
  ...
);
```

где

table – название таблицы;
column – название столбца;
datatype – тип данных столбца;
DEFAULT *expr* – значение по умолчанию для столбца.

Названия таблиц и столбцов должны:

- начинаться с буквы и иметь длину 1–30 символов;
- состоять из букв, цифр, знака подчеркивания и символов $ и # (последние два символа допустимы, но использовать их не рекомендуется);
- имена таблиц не должны повторять имена других объектов, принадлежащих тому же пользователю;
- имена не могут являться зарезервированными словами Oracle.

Пример:

```
CREATE TABLE student
(
    id number(10, 0),
    first_name VARCHAR2(50),
    last_name VARCHAR2(50),
    group_id number(10,0)
)
```

Типы данных

VARCHAR2(size)	Символьные данные переменной длины (size – максимальная длина строки, 1<=size<=4000)
CHAR(size)	Символьные данные фиксированной длины (size – максимальная длина строки, 1<=size<=2000)
NUMBER(p,s)	Число общей длиной p десятичных знаков с s десятичными знаками после запятой
DATE	Дата и время в интервале от 1 января 4712 до н.э. до 31 декабря 9999 н.э.
LONG	Символьные данные переменной длины до 2 Гбайт
CLOB	Последовательность однобайтовых символьных данных длиной до 4 Гбайт
RAW(size)	Двоичные данные длины size. 1<=size<=2000
LONG RAW	Двоичные данные переменной длины до 2 Гбайт
BLOB	Двоичные данные до 4 Гбайт
BFILE	Двоичные данные, сохраненные во внешнем файле; до 4 Гбайт

Создание таблицы на основе строк другой таблицы

```
CREATE TABLE table
[column(, column...)]
AS subquery;
```

где

table – название таблицы;

column – название столбца, значение по умолчанию и ограничение целостности;

subquery – запрос SELECT, задающий строки, которые будут добавлены в новую таблицу.

Если названия столбцов опущены, то они будут такими же, как названия столбцов в запросе.

Изменение таблицы

Добавление столбца:

```
ALTER TABLE table
ADD (column datatype [DEFAULT expr]
[, column datatype]...);
```

где

column – имя столбца;
datatype – тип данных столбца.

Изменение столбца:

```
ALTER TABLE table
MODIFY (column datatype [DEFAULT expr]
[, column datatype]...);
```

Удаление столбца:

```
ALTER TABLE table DROP column;
```

Лабораторная работа №4
«SQL: INSERT / UPDATE / DELETE»

Цель работы

Получение практических навыков вставки/удаления/модификации данных в таблицах с помощью соответствующих операторов Data Manipulation Language.

Теоретические сведения

Оператор INSERT

Новые строки добавляются в таблицу с помощью оператора INSERT. Один оператор используется для добавления одной строки.

```
INSERT INTO table [(column [, column...])]
VALUES (value [, value...]);
```

Если список столбцов не указан, то порядок и количество значений должны совпадать с порядком и количеством полей при создании таблицы. Явное указание столбцов позволяет не указывать значения для некоторых

столбцов добавляемой строки и указывать значения столбцов в порядке, отличном от заданного при создании таблицы.

Существует несколько способов добавления пустых значений (NULL) в таблицу (в примере пустое значение добавляется в поле номера группы group_no):

опустить столбец в списке вставки:

```
INSERT INTO student (id, fio)
    VALUES (121, 'Ivanov I.A.');
```

явно указать пустое значение:

```
INSERT INTO student (id, fio, groupno)
    VALUES (121, 'Ivanov I.A.', NULL);
```

или, если перечислены значения всех полей:

```
INSERT INTO student VALUES (121, 'Ivanov I.A.', NULL);
```

Оператор INSERT может использоваться для копирования строк из другой таблицы. Для этого вместо списка значений используется подзапрос.

```
INSERT INTO table (column, column,…)
    subquery;
```

Оператор UPDATE

Оператор используется для изменения значений существующих строк в таблице. Оператор может изменять несколько строк сразу, если это требуется.

```
UPDATE table
SET column = value [, column = value]
[WHERE condition];
```

где

column – название столбца;
value – новое значение;
condition – условие выборки изменяемых строк.

Оператор DELETE

Оператор DELETE используется для удаления строк из таблицы. Синтаксис сходен с оператором UPDATE.

```
DELETE [FROM] table
[WHERE condition];
```

Транзакции

Транзакции состоят из одного из перечисленных ниже элементов:

- нескольких операций DML, составляющих одно целостное изменение данных;
- одной операции DDL;
- одной операции DCL.

Каждая транзакция начинается с первого исполняемого предложения SQL. Транзакция завершается после:

- выполнения COMMIT или ROLLBACK;
- выполнения операции DDL или DCL (автоматический неявный COMMIT);
- выхода пользователя из системы;
- аварийного завершения работы.

Управление транзакциями осуществляется с помощью команд, приведенных в табл. 4.

Таблица 4

Команды управления транзакциями

Команда	Описание
COMMIT	Завершает текущую транзакцию и фиксирует все накопившиеся изменения
SAVEPOINT *имя*	Создает позицию отката с указанным именем внутри текущей транзакции
ROLLBACK	Завершает транзакцию с отменой всех сделанных изменений
ROLLBACK TO SAVEPOINT *имя*	Отменяет изменения, сделанные в текущей транзакции, возвращаясь к указанной позиции отката

Лабораторная работа №5
«SQL: WHERE, JOIN»

Цель работы

1. Получение практических навыков построения выборок с использованием секции WHERE оператора SELECT

2. Построение выборок из нескольких таблиц с использованием различных приемов объединения таблиц.

Теоретические сведения

Выборка строк

Ограничить выборку строк, возвращаемых запросом, можно с помощью секции WHERE. Секция WHERE содержит условие, которое должно быть выполнено, и следует непосредственно за секцией FROM. Предложение с секцией WHERE имеет следующий синтаксис:

```
SELECT [DISTINCT] {*, столбец [псевдоним], ...}
FROM таблица
[WHERE условие(я)];
```

WHERE указывает, что в запросе должны присутствовать только строки, удовлетворяющие условию. Условие состоит из названий столбцов, выражений, констант и операторов сравнения.

В секции WHERE могут сравниваться значения столбцов, литералы, арифметические выражения или функции.

Строки символов и даты

Строки символов и даты, используемые в секции WHERE, должны заключаться в апострофы (‘ ‘). Это правило не относится к числовым константам. Поиск строк всегда чувствителен к регистру, а поиск дат – к формату даты.

Oracle хранит даты во внутреннем формате, представляющем век, год, месяц, день, часы, минуты и секунды. Используемый по умолчанию формат даты имеет вид DD-MON-YY.

Операторы сравнения

Oracle допускает следующие простейшие операторы сравнения: =, >, <, >=, <=, <> (равно, больше, меньше, больше либо равно, меньше либо равно и не равно соответственно). Они используются в секции WHERE следующим образом:

```
WHERE выражение оператор значение

WHERE hdate='01-JAN-95'
WHERE amt>=1500
WHERE name='SMITH'
```

Следует иметь в виду, что в качестве *выражения* недопустимо использование псевдонима.

К операторам сравнения также относятся BETWEEN … AND …, IN(*список*), LIKE, IS NULL.

BETWEEN

Оператор используется для отображения строк с каким-либо значением, входящим заданный диапазон. При этом в операторе указывается верхняя и нижняя границы диапазона. Оператор ищет значения, удовлетворяющие диапазону, включая границы. Нижний предел всегда указывается первым.

IN

Для проверки принадлежности значения поля некоторому списку значений используется оператор IN. Оператор IN может использоваться с любым типом данных.

LIKE

При поиске строк может оказаться, что точная строка или строки, которые нужно найти, не известны. С помощью оператора LIKE можно выбирать строки на основании совпадения с некоторым образцом. Операция поиска совпадений с образцом обычно называется поиском по шаблону. Для построения строки поиска используются два специальных символа: % обозначает последовательность из нуля и более произвольных символов, _ обозначает один произвольный символ.

В тех случаях, когда требуется точное совпадение для самих символов «%» и «_», используется опция ESCAPE. Это ключевое слово показывает, какой символ играет роль escape-символа.

```
SELECT name
FROM firm
WHERE name LIKE '%A\_B%' ESCAPE '\';
```

IS NULL

Для сравнения с пустым значением используется оператор IS NULL. Как уже отмечалось, пустое значение – это значение, которое недоступно, не присвоено, не известно, или неприменимо. Таким образом, для проверки нельзя использовать оператор =, так как пустое значение не может быть равным или не равным любому значению (в том числе и другому пустому значению).

Логические операторы

Логические операторы используются для объединения двух результатов вычисления условий в один, или для обращения результата вычисления одного условия. SQL использует три логических оператора: AND, OR, NOT. Логические операторы AND и OR дают возможность использовать в секции WHERE несколько условий.

Указание. Оператор NOT также можно использовать в сочетании с другими операторами SQL, такими, как BETWEEN, LIKE и NULL, например:

```
WHERE job NOT IN ('assistant', 'professor')
WHERE amt NOT BETWEEN 1000 AND 1500
WHERE name NOT LIKE '%A%'
WHERE val IS NOT NULL
```

Правила предшествования

При вычислении условий в секции WHERE соблюдается определенный порядок вычисления операторов. В первую очередь вычисляются все операторы сравнения, затем операторы NOT, затем AND, и затем обладающие самым низким приоритетом операторы OR (табл. 5).

Таблица 5

Приоритет логических операторов

Порядок вычисления	Оператор
1	Операторы сравнения
2	NOT
3	AND
4	OR

Для того чтобы явно задать или изменить порядок выполнения операторов, можно использовать скобки.

Соединение таблиц

Строки из одной таблицы могут быть объединены со строками в другой таблице в соответствии с общими значениями, существующими в соответствующих столбцах, которые, как правило, являются первичными или внешними ключами. Для отображения данных из двух или более таблиц в секции FROM пишется условие соединения.

```
SELECT table1.column, table2.column
FROM table1 [INNER|LEFT|RIGHT] JOIN table2
     ON table1.column1 = table2.column2;
```

где

table.column – указывает таблицу и столбец, из которых извлекаются данные;

table1.column1 = table2.column2 – условие, которое объединяет (или соотносит) таблицы.

Указания:

- При написании предложения SELECT, осуществляющего объединение таблиц, перед именами столбцов следует писать имена таблиц. Это повысит читаемость предложений и улучшит доступ к базе данных.

- Если один и тот же столбец присутствует более чем в одной таблице, имя таблицы должно обязательно указываться перед именем столбца.
- Для объединения вместе n таблиц требуется $(n-1)$ условий объединения.

Декартово произведение

Когда условие объединения не верно или просто опущено, результатом будет декартово произведение, в котором будут представлены все комбинации строк. Все строки из первой таблицы соединяются со всеми строками из второй таблицы.

Декартово произведение, как правило, создает большое количество строк, и результаты его выполнения редко бывают полезны. Следует всегда указывать правильное условие объединения в секции FROM или WHERE, за исключением случаев, когда есть специальная необходимость получить комбинации всех строк во всех таблицах.

```
SELECT student.fio, groups.number
FROM student, groups;
```

Псевдонимы таблиц

Записывать имя таблицы перед каждым столбцом может оказаться очень трудоемкой работой, особенно если имена таблиц достаточно длинные. Существует возможность использовать вместо имен таблиц *псевдонимы*. Так же, как псевдоним столбца дает новое имя столбцу, псевдоним таблицы дает новое имя таблице. Псевдонимы таблиц помогают сделать SQL-код компактнее, благодаря чему он требует меньше памяти. Псевдонимы таблиц задаются в секции FROM. Имя таблицы указывается полностью, затем через пробел указывается псевдоним.

```
SELECT s.fio, g.number
FROM student s INNER JOIN groups g ON s.group_id=g.id;
```

Указания:

- Псевдонимы таблиц могут иметь длину до 30 символов, но чем они короче, тем лучше.
- Если псевдоним таблицы используется для некоторого имени таблицы в секции FROM, то этот псевдоним будет заменяться соответствующим именем таблицы во всем предложении SELECT.
- Псевдонимы таблиц должны быть осмысленными.
- Псевдоним таблицы действителен только в пределах текущего предложения SELECT.

Объединение более чем двух таблиц

Выполняется последовательное присоединение таблиц с помощью JOIN в секции FROM:

```
SELECT s.fio, g.number
FROM (student s INNER JOIN groups g
     ON s.group_id=g.id) INNER JOIN department d
     ON g.dept_id=d.id
WHERE d.name='Dept. of information science';
```

Объединение одной таблицы и объединение по неэквивалентности

Часто бывает необходимо соединить две копии одной и той же таблицы либо использовать для соединения условие, отличное от равенства. Следующий пример демонстрирует обе эти возможности на примере расписания сеансов в кинотеатре.

```
SELECT f1.id, f1.start_time, f2.id, f2.start_time
FROM film f1 INNER JOIN film f2
     ON f1.end_time+10/(24*60) < f2.start_time;
```

В результате выполнения приведенного запроса будут выбраны все такие пары сеансов, что между окончанием первого и началом второго сеанса из пары проходит не менее 10 минут.

Лабораторная работа №6
«SQL*Plus форматирование вывода, переменные подстановки»

Цель работы

1. Получение практических навыков применения подстановочных переменных для получения данных от пользователя в процессе выполнения командных скриптов и реализации интерактивных запросов.
2. Изучение возможностей SQL*Plus для форматирования вывода результатов запроса и построения простейших отчетов.

Теоретические сведения

Примеры интерактивности

С помощью SQL*Plus можно создавать отчеты, которые во время выполнения запрашивают у пользователя некоторые значения для выборки возвращаемых данных. Для создания интерактивных отчетов в командный

файл или в отдельное SQL-предложение можно включать переменные подстановки. Переменная может представляться как контейнер, в котором временно хранятся значения.

В SQL*Plus можно использовать одноамперсандные (&) переменные подстановки для временного хранения значений. Переменные в SQL*Plus можно предопределять с помощью команд ACCEPT и DEFINE. ACCEPT читает строку ввода пользователя и сохраняет ее в переменной. DEFINE создает и присваивает значение переменной.

Интерактивные возможности допускают непосредственное взаимодействие пользователя с секцией WHERE (например, для построения различных выборок при неизменном тексте запроса). Те же принципы могут использоваться и для достижения других целей. Например, для

- динамического изменения верхних и нижних колонтитулов;
- получения входных значений из файла, а не от пользователя;
- передачи значений из одного предложения SQL в другое.

Интерактивный ввод может также использоваться для ввода различных значений во время отладки запросов.

Переменные подстановки с одним амперсандом (&)

При запуске отчета пользователю часто необходимо осуществлять динамическую выборку данных. SQL*Plus предоставляет такую возможность с помощью пользовательских переменных. Для обозначения переменной в тексте SQL-предложения используется амперсанд (&).

```
SELECT id, fio, group_no
FROM student
WHERE id = &student_id;

Enter value for sudent_id: 121

id      fio          group_no
------- ------------ ---------
    121 Ivanov I.A.  001001
```

При использовании одиночного амперсанда пользователю будет предложено ввести новое значение для переменной при каждом запуске предложения при условии, что эта переменная не существует.

&user_variable – указывает на переменную в предложении SQL; если переменная не существует, SQL*Plus предлагает пользователю ввести значение (SQL*Plus удаляет новую переменную после использования).

Команда SET VERIFY

Для подтверждения изменений в предложении SQL можно использовать команду SQL*Plus SET VERIFY. Установка SET VERIFY ON приводит к тому,

что SQL*Plus будет отображать текст команды до и после замены переменной подстановки соответствующим значением.

Ввод символьных данных и дат с помощью переменных подстановки

В секции WHERE даты и символьные данные должны заключаться в апострофы. То же правило относится и к переменным подстановки.

Для того чтобы данные не приходилось указывать в апострофах каждый раз во время выполнения, в апострофы можно заключить саму переменную в тексте SQL-предложения.

Совместно с амперсандом можно также использовать функции, такие как UPPER и LOWER. Например, можно записать UPPER('&fio ') для автоматического преобразования введенной пользователем фамилии в верхний регистр.

Переменные подстановки с двумя амперсандами

Переменную подстановки с двумя амперсандами можно применять в том случае, когда требуется использовать переменную повторно, не запрашивая данные у пользователя каждый раз. В этом случае приглашение для ввода значения будет выдано пользователю только один раз.

```
SELECT id, fio, &&column_name
FROM student
ORDER BY &column_name;
```

В этом примере пользователю будет предложено ввести значение переменной *column_name* только один раз. Значение, введенное пользователем, будет использовано как для отображения данных, так и для сортировки.

SQL*Plus сохраняет полученные значения с помощью команды DEFINE и использует их в дальнейшем везде, где встречается имя данной переменной. Чтобы удалить переменную пользователя, определенную таким образом, нужно использовать команду UNDEFINE.

Создание пользовательских переменных

Переменные можно создавать до выполнения предложения SELECT. SQL*Plus предоставляет две команды для создания и установки значений пользовательских переменных: DEFINE и ACCEPT. Эти команды описаны в табл. 6.

Таблица 6

Команды управления пользовательскими переменными

Команда	Описание
DEFINE *variable* = *value*	Создает пользовательскую переменную с типом CHAR и присваивает ей значение
DEFINE *variable*	Отображает переменную, ее значение и ее тип
DEFINE	Отображает все пользовательские переменные, их значения и типы
ACCEPT	Читает строку ввода от пользователя и сохраняет его в переменной

```
ACCEPT variable [datatype] [FORMAT format] [PROMPT text]
{HIDE}
```

где

variable – имя переменной, которая хранит значение. Если переменной с таким именем не существует, SQL*Plus создаст ее;

datatype – NUMBER, CHAR или DATE. CHAR имеет ограничение по длине не более 240 байт. DATE проверяется на соответствие формату и имеет тип данных CHAR;

FOR[MAT] *format* – указывает шаблон формата. Например, A10 или 9.999;

PROMPT *text* – отображает текст подсказки перед тем, как пользователю будет предложено ввести значение;

HIDE скрывает данные, введенные пользователем (например пароль).

При указании имени переменной подстановки в команде ACCEPT перед ним не следует писать амперсанд.

Команды форматирования SQL*Plus

Свойствами отчета можно управлять с помощью команд, приведенных в табл. 7.

Таблица 7

Команды форматирования отчета

Команда	Описание
COL[UMN] [*column option*]	Управляет форматом столбцов
TTI[TLE] [*text*|OFF|ON]	Указывает заголовок (верхний колонтитул), выводимый в начале каждой страницы
BTI[TLE] [*text*|OFF|ON]	Указывает нижний колонтитул, выводимый в конце каждой страницы отчета
BRE[AK] [ON *report_element*]	Замена повторяющихся значений пустыми строками и разбивка строк отчета на секции

Замечания:

• Все команды форматирования действуют до окончания текущего сеанса SQL*Plus или до тех пор, пока конкретная настройка форматирования не будет перезаписана или очищена.

• Следует сбрасывать все настройки SQL*Plus к значениям по умолчанию после каждого отчета.

• Не существует команды SQL*Plus, позволяющей вернуть переменной ее значение по умолчанию; нужно либо знать конкретное значение, либо выйти из системы и войти снова.

• Если столбцу присваивается псевдоним, то необходимо указывать псевдоним столбца, а не его имя.

Возможности команды COLUMN

Команда COLUMN используется для управления отображением столбцов. Ее параметры приведены в табл. 8.

Таблица 8

Параметры команды COLUMN

Параметр	Описание
CLE[AR]	Очищает любое форматирование столбцов
FOR[MAT] *format*	Изменяет способ отображения данных столбца
HEA[DING] *text*	Устанавливает заголовок столбца. Вертикальная черта (|) вызовет принудительный перевод строки в заголовке, если не используется выравнивание
JUS[TIFY] {*align*}	Выравнивает заголовок столбца (не данные) по левому или правому краю или по центру
NOPRI[NT]	Скрывает столбец
NUL[L] *text*	Указывает текст, отображаемый для пустых значений
PRI[NT]	Отображает столбец
TRU[NCATED]	Усекает строку в конце первой строки вывода
WRA[PPED]	Переносит конец строки на следующую строку

Отображение и очистка настроек

Чтобы просмотреть или очистить текущие настройки команды COLUMN, используются следующие команды, приведенные в табл. 9.

Таблица 9

Управление настройками COLUMN

Команда	Описание
COL[UMN] *column*	Отображает текущие настройки для указанного столбца
COL[UMN]	Отображает текущие настройки для всех столбцов
COL[UMN] column CLE[AR]	Очищает настройки для указанного столбца
CLE[AR] COL[UMN]	Очищает настройки для всех столбцов

Если команда не умещается на одной строке, ее можно продолжить на следующей строке, окончив текущую знаком переноса (-).

Шаблоны форматов COLUMN

Некоторые варианты команды COLUMN требуют использования строки формата. Описание символов строки формата приведено в табл. 10.

Таблица 10

Шаблоны форматов

Элемент	Описание	Пример	Результат
A*n*	Устанавливает ширину отображения в *n*	N/A	N/A
9	Одна цифра с подавлением нулей	999999	1234
0	Принудительная печать ведущего нуля	099999	01234
$	Знак доллара	$9999	$1234
L	Знак местной валюты	L9999	L1234
.	Позиция десятичной точки	9999.99	1234.00
,	Разделитель тысяч	9,999	1,234

Oracle Server отображает строку из символов решетки (#) на месте целого числа, число цифр в котором превосходит число цифр, указанное шаблоном формата. Решетки также отображаются на месте значения, для которого указан буквенно-цифровой шаблон формата, и которое само при этом является числовым.

Использование команды BREAK

Команда BREAK используется для разбивки строк на секции и подавления повторяющихся значений. Для обеспечения эффективной работы команды BREAK ее следует применять для столбцов, предварительно отсортированных секцией ORDER BY.

```
BREAK on column[|alias|row] [skip n|dup|page] on .. [on
report]
```

где

page – начинает новую страницу, когда значение разрыва изменяется;

skip n – пропускает *n* строк, когда значение разрыва изменяется, разрывы могут быть активны на:

- столбце,
- строке,
- странице,
- отчете;

duplicate – отображает повторяющиеся значения.

Очистить все настройки команды BREAK можно, используя команду CLEAR:

```
CLEAR BREAK
```

Использование команд TTITLE и BTITLE

Команда TTITLE используется для форматирования верхних колонтитулов, а команда BTITLE – для форматирования нижних колонтитулов. Нижние колонтитулы печатаются внизу на страницы в соответствии со значением PAGESIZE.

Синтаксис BTITLE и TTITLE идентичен. Для разделения текста заголовка на несколько строк можно использовать вертикальную черту (|).

```
TTI[TLE] [text|OFF|ON]
```

где

text – текст заголовка. Заключается в одиночные кавычки, если состоит из более чем одного слова.

Создание файла сценария для выполнения отчета

Команды SQL*Plus можно либо вводить по одной в командной строке SQL, либо поместить все команды, включая предложение SELECT, в командный файл (или файл сценария). Типичный сценарий состоит из по меньшей мере одного предложения SELECT и нескольких команд SQL*Plus.

Последовательность создания файла сценария в ходе работы:

1. Создайте SQL-предложение SELECT с использованием командной строки SQL. Убедитесь в точности данных, необходимых для отчета, прежде чем сохранить предложение в файл и добавить команды форматирования.

Убедитесь, что присутствует соответствующая секция ORDER BY, если планируется использование разрывов.

2. Сохраните предложение SELECT в файл сценария.

3. Отредактируйте файл сценария, чтобы добавить команды SQL*Plus.

4. Добавьте необходимые команды форматирования перед предложением SELECT. Не размещайте команды SQL*Plus внутри предложения SELECT.

5. Убедитесь, что предложение SELECT завершается символом запуска, либо точкой с запятой (;), либо косой чертой (/).

6. Добавьте команды SQL*Plus, очищающие настройки формата, после символа запуска. Или, альтернативно, можно вызвать файл сброса, содержащий все команды очистки форматирования.

7. Сохраните файл сценария со сделанными изменениями.

8. В SQL*Plus запустите файл сценария, набрав START *имя_файла* или *@ имя_файла*. Эта команда необходима для чтения и выполнения файла сценария.

Замечания:

• Между командами SQL*Plus в файле сценария можно вставлять пустые строки.

• Команды SQL*Plus можно сокращать (см. синтаксис).

• Чтобы восстановить первоначальные настройки среды SQL*Plus, рекомендуется включать команды сброса настроек в конце файла сценария.

Лабораторная работа №7
«SQL: Функции»

Цель работы

Изучение различных групп встроенных однострочных функций в SQL и получение навыков их использования.

Теоретические сведения

Функции в SQL

Функции являются очень важной и мощной составляющей языка SQL. Они могут использоваться для произведения вычислений с данными, изменения отдельных элементов данных, управления выводом групп строк, форматирования дат и числовых данных перед отображением, преобразования типов данных столбцов.

Функции принимают аргументы и возвращают значения.

Функции можно разделить на две группы: функции, работающие с одной строкой, и функции, работающие с несколькими строками.

К первой группе относятся функции, которые одновременно работают только с одной строкой и возвращают один результат для каждой строки. Функции второй группы работают с группами строк, выдавая некоторый общий результат для всей группы. Далее в этой работе рассматриваются только функции, принадлежащие к первой группе.

Функции принимают один или более аргументов и возвращают одно значение для каждой строки, возвращенной запросом. Аргументами могут быть константы, указанные пользователем, имена переменных, имена столбцов, выражения.

Свойства однострочных функций:

- работают с каждой строкой, возвращенной запросом;
- возвращают по одному результату для каждой строки;
- могут принимать один или несколько аргументов;
- могут использоваться в секциях SELECT, WHERE и ORDER BY;
- вызовы функций могут быть вложенными.

имя_функции(*столбец*|*выражение*,[*арг2, арг3, …*])

где

имя_функции – имя вызываемой функции;
столбец – любой поименованный столбец таблицы;
выражение – любое символьное или числовое выражение;
арг2, арг3 … – любые аргументы, которые будут использоваться функцией.

Символьные функции

Эти функции подразделяются на два вида: функции для манипуляций с символами и функции преобразования регистра (табл. 11).

Таблица 11

Символьные функции

Функция	Назначение
LOWER(*column*|*expression*)	Преобразует текстовые символы к нижнему регистру
UPPER(*column*|*expression*)	Преобразует текстовые символы к нижнему регистру
INITCAP(*column*|*expression*)	Преобразует первые символы в каждом слове к верхнему регистру, остальные преобразует к нижнему регистру
CONCAT(*column1*|*expression1* ,*column2*|*expression2*)	Выполняет конкатенацию двух символьных значений/выражений. Эквивалентна оператору конкатенации (||)
SUBSTR(*column*|*expression*,*m*[,*n*])	Подстроку длиной *n* из переданной символьной строки, начиная с позиции *m*. Если *m* отрицательно, то счет идет от конца строки. Если параметр *n* отсутствует, то возвращаются все символы до конца переданной строки
LENGTH(*column*|*expression*)	Возвращает число символов в переданном значении
INSTR(*column*|*expression*,*m*)	Возвращает номер позиции указанного символа в символьном значении
LPAD(*column*|*expression*, *n*,'*string*')	Дополняет символьное значение слева переданным символом до общей длины в *n* позиций (т.е. выполняется выравнивание по правому краю)

Числовые функции

Функции этой группы принимают и возвращают значения числового типа. В табл. 12 рассмотрены некоторые из таких функций.

Таблица 12

Числовые функции

Функция	Назначение
ROUND(*column*|*expression*, *n*)	Округляет значение в столбце, результат вычисления выражения или некоторое значение до *n* знаков после запятой. По умолчанию принято *n=0*. Значение *n* может быть отрицательным. В этом случае считаются знаки до запятой, что позволяет округлять, например, до десятков, сотен, тысяч и т.д.
TRUNC(*column*|*expression*,*n*)	Усекает значение столбца, выражения или константы до *n* знаков после запятой. По умолчанию принято *n=0*. Если *n* отрицательно, то все знаки после запятой отбрасываются, а *n* знаков до запятой заменяются нулями.
MOD(*m*,*n*)	Возвращает остаток от деления *m* на *n*.

Работа с датами

Oracle хранит даты в числовом формате, представляющем век, год, месяц, день, часы, минуты и секунды. По умолчанию даты отображаются в формате DD-MON-YY. Все даты в Oracle должны находиться в диапазоне от 1 января 4712 до н.э. до 31 декабря 9999 н.э.

SYSDATE – функция, которая возвращает текущие дату и время. SYSDATE можно использовать просто как имя столбца. Например, можно отобразить текущую дату, выбрав SYSDATE из некоторой таблицы. Как правило, для этого используется фиктивная таблица DUAL.

Арифметика с датами

Так как база данных хранит даты как числа, с датами можно производить арифметические действия, такие как сложение и вычитание. Допустимые операции перечислены в табл. 13.

Таблица 13

Операции с датами

Операция	Результат	Описание
Дата + число	Дата	Прибавляет число дней к дате
Дата – число	Дата	Отнимает число дней от даты
Дата – дата	Число дней	Вычисляет разницу в днях между двумя датами
Дата + число/24	Дата	Добавляет число часов к дате

Функции для работы с датами

Функции этой группы работают с датами в формате Oracle. Все функции возвращают значение типа DATE, кроме функции MONTHS_BETWEEN, которая возвращает число.

• MONTHS_BETWEEN(*date1, date2*) – возвращает число месяцев между датами *date1* и *date2*.

В зависимости от того, «позже» ли первая дата, чем вторая, или наоборот, результат может быть соответственно положительным либо отрицательным. Результат может быть дробным, что представляет часть месяца, если между датами неполное число месяцев.

• ADD_MONTHS(*date, n*) – добавляет *n* календарных месяцев к дате *date*. *n* должно быть целым, но может быть отрицательным.

• NEXT_DAY(*date, 'char'*) – находит дату следующего ближайшего к дате дня недели, указанного в параметре '*char*'. *char* может быть либо номером дня, либо строкой.

• LAST_DAY(*date)*. – находит дату последнего дня в месяце, который содержит дату *date*.

• ROUND(*date*[,'*fmt*']) . Возвращает дату *date*, округленную до единиц, указанных в шаблоне формата *fmt*. По умолчанию округление происходит до дней.

• TRUNC(*date*[, '*fmt*']) Возвращает *date*, усеченную до единиц, указанных в *fmt*. По умолчанию усечение происходит до дней.

Шаблоны формата даты приведены ниже в табл. 15.

Функции преобразования типов

В дополнение к типам Oracle столбцы таблиц в базе данных Oracle8 могут быть определены с использованием типов данных ANSI, DB2 и SQL/DS. Тем не менее Oracle Server для внутреннего использования преобразует такие типы данных к типам Oracle8.

В некоторых случаях Oracle Server допускает использование какого-то типа данных, хотя на самом деле ожидается другой тип данных. Такая ситуация допускается, если Oracle Server может автоматически преобразовать данные к необходимому типу. Такое преобразование типов может осуществляться *неявно*, самим Oracle Server, или *явно*, пользователем. Для явного преобразования применяются функции, приведенные в табл. 14.

Таблица 14

Функции преобразования типов

Функция	Назначение
TO_CHAR(*number*|*date*,['*fmt*'])	Преобразует числовое значение или дату в строку VARCHAR2, используя шаблон формата *fmt*
TO_NUMBER(*char*)	Преобразует строку символов в число
TO_DATE(C*har*,['*fmt*'])	Преобразует строку символов, представляющую дату, в дату в соответствии с шаблоном формата *fmt*. (Если *fmt* не указан, подразумевается формат

	DD-MON-YY.)

Форматы даты и времени

Таблица 15

Элементы форматирования даты

Элемент	Описание
SCC или CC	Век, S дополняет даты до н.э. знаком минус
YYYY или SYYYY	Год, S дополняет даты до н.э. знаком минус
YYY или YY или Y	Последние 3, 2, или 1 цифра года
Y,YYY	Год с запятой в указанной позиции
IYYY, IYY, IY, I	4, 3, 2, или 1 цифры года в соответствии со стандартом ISO
SYEAR или YEAR	Год прописью, S дополняет даты до н.э. знаком минус
BC или AD	Добавляет к году BC/AD
B.C. или A.D.	То же с точками
Q	Квартал года
MM	Месяц, двухзначное число
MONTH	Название месяца, дополненное пробелами до длины в 9 символов
MON	Трехбуквенное сокращение названия месяца
RM	Месяц римскими цифрами
WW или W	Неделя года или месяца
DDD или DD или D	День года, месяца или недели
DAY	Имя дня недели, дополненное пробелами до длины в 9 символов.
DY	Трехбуквенное сокращение имени дня недели
J	Юлианская дата

Приведенные ниже форматы используются для отображения информации о времени (табл. 16) и для преобразования чисел в цифровой записи в значения прописью (табл. 17).

Таблица 16

Элементы форматирования времени

Элемент	Описание
AM или PM	Индикатор AM/PM
A.M. или P.M.	То же с точками
HH или HH12 или HH24	Час суток или час (1–12) или час (0–23)
MI	Минута (0–59)
SS	Секунда (0–59)
SSSSS	Секунды после полуночи (0–86399)

Таблица 17

Дополнительные элементы форматирования

Элемент	Описание
/ . ,	Пунктуация воспроизводится в результате
"of the"	Строки в двойных кавычках воспроизводятся в результате
TH	Порядковое числительное (например DDTH для 4TH)
SP	Число прописью (например DDSP для FOUR)
SPTH или THSP	Порядковое числительное прописью (например DDSPTH для FOURTH)

Числовые форматы

При преобразовании числа в строку можно пользоваться следующими элементами форматирования, приведенными в табл. 18.

Таблица 18

Преобразование чисел в строки

Элемент	Описание	Пример	Результат
9	Цифровая позиция (количество символов 9 определяет ширину поля)	999999	1234
0	Отображать нулевые старшие разряды	099999	001234
$	Добавлять знак доллара	$999999	$1234
L	Добавлять знак местной валюты	L999999	FF1234
.	Десятичная точка в указанной позиции	999999.99	1234.00
,	Запятая в указанной позиции	999,999	1,234
MI	Знак минуса справа (отрицательные значения)	999999MI	1234-
PR	Отрицательные значения в треугольных скобках	999999PR	<1234>
EEEE	Экспоненциальное представление числа	99.999EEEE	1.234E+03
V	Умножить на 10 *n* раз (*n* равно количеству 9 после V)	9999V99	123400
B	Отображать нулевые значения пробелом, а не 0	B9999.99	1234.00

Функция NVL

Функция используется для преобразования пустого значения в какое-либо действительное значение.

NVL (*expr1*, *expr2*)

где

expr1 – исходное значение или выражение, которое может содержать null;

expr2 – значение, в которое преобразуется null.

Функция NVL может использоваться для преобразования любого типа данных, но возвращаемое значение всегда имеет тот же тип, что и *expr1*.

Примеры:

NUMBER	NVL(*number_column*,9)
DATE	NVL(*date_column*, '01-JAN-95')
CHAR или VARCHAR2	NVL(*character_column*, 'Unavailable')

Функция DECODE

Функция DECODE разбирает выражение аналогично алгоритму конструкции IF-THEN-ELSE, используемой в различных языках. Функция DECODE декодирует *expr*, сравнивая его с каждым из значений *search*. Если выражение совпадает с *search*, в результате возвращается значение *result*.

```
DECODE(expr, search1, result1, search2, result2 ...)
```

Если значение по умолчанию опущено, то функция вернет значение NULL, если не будет обнаружено совпадений в списке поиска.

Лабораторная работа №8
«SQL: Группировки»

Цель работы

1. Изучение средств группировки данных выборки с помощью секции GROUP BY оператора SELECT

2. Изучение средств последующего ограничения выборки с помощью секции HAVING.

Теоретические сведения

Группировка данных

В некоторых случаях требуется разделить таблицу на меньшие группы. Это может быть сделано с помощью секции GROUP BY. Секция GROUP BY может использоваться для разделения строк таблицы на группы. Затем можно использовать агрегатные функции для получения информации по каждой группе.

```
SELECT column, group_function(column)
FROM table
[WHERE condition]
[GROUP BY group_by_expression]
[ORDER BY column];
```

Выражение group_by_expression указывает столбцы, значения которых определяют способ группировки строк таблицы.

Указания:

- Если в секцию SELECT включена агрегатная функция, то этой же секции нельзя указывать одиночных значений, за исключением столбцов, указанных в секции GROUP BY. Включение других столбцов в секцию SELECT приведет к сообщению об ошибке.
- Используя секцию WHERE, можно осуществить предварительную выборку строк перед их группировкой.
- В секции GROUP BY необходимо указывать имена столбцов.
- В секции GROUP BY недопустимо использование псевдонимов столбцов.
- По умолчанию строки сортируются по возрастанию значений в столбцах, указанных в секции GROUP BY. Этот порядок можно изменить, используя секцию ORDER BY.
- При использовании секции GROUP BY необходимо убедиться, что все столбцы в секции SELECT, к которым не применяются агрегатные функции, включены в секцию GROUP BY.
- Столбец, указанный в секции GROUP BY, необязательно должен присутствовать в секции SELECT. Агрегатные функции можно использовать для сортировки строк в секции ORDER BY.
- Иногда возникает необходимость получить результат для групп внутри групп. Результаты применения агрегатных функций к подгруппам можно получить, указывая более одного столбца в секции GROUP BY. Способ сортировки, применяемый по умолчанию к результатам запроса, можно изменить, меняя порядок перечисления столбцов в секции GROUP BY
- Любой столбец или выражение в списке секции SELECT, не являющиеся агрегатной функцией, должны присутствовать в секции GROUP BY.
- Секция HAVING используется для задания групп, которые необходимо выбрать. Таким образом осуществляется выборка групп на основе информации, получаемой с помощью агрегатных функций. Секция WHERE не может быть использована для задания ограничений для групп.

```
SELECT column, group_function
FROM table
[WHERE condition]
[GROUP BY group_by_expression]
[HAVING group_condition]
[ORDER BY column];
```

где

group_condition – условие, ограничивающее возвращаемую выборку только теми группами строк, для которых оно выполняется.

Oracle Server обрабатывает секцию HAVING в следующем порядке:

1. Выполняется группировка строк.
2. Агрегатная функция применяется к группе.
3. Отображаются группы, соответствующие критерию в секции HAVING.

Секцию GROUP BY можно применять без использования агрегатных функций в списке SELECT.

Если производится выборка строк на основе результатов вычисления агрегатной функции, то необходимо совместно с секцией HAVING использовать секцию GROUP BY.

Агрегатные функции

В отличие от функций, работающих с одной строкой, агрегатные функции проводят операции над наборами строк, возвращая один результат для каждого набора. Эти наборы могут быть как целой таблицей, так и таблицей, разделенной на группы.

Каждая функция может принимать различные аргументы. В табл. 19 приведен синтаксис некоторых агрегатных функций.

Таблица 19

Агрегатные функции

Функция	**Описание**
AVG([DISTINCT|ALL]*n*)	Среднее значение *n*, исключая пустые значения
COUNT({*|[DISTINCT|ALL]*expr*})	Число строк, где expr не является пустым значением. Посчитать все выбранные строки можно, используя *, при этом результат будет включать все повторяющиеся строки и строки с пустыми значениями.
MAX([DISTINCT|ALL]*expr*)	Максимальное значение для *expr*, исключая пустые значения
MIN([DISTINCT|ALL]*expr*)	Минимальное значение для *expr*, исключая пустые значения
STDDEV([DISTINCT|ALL]*x*)	Среднеквадратичное отклонение *n*, исключая пустые значения
SUM([DISTINCT|ALL]*n*)	Сумма значений *n*, исключая пустые значения
VARIANCE([DISTINCT |ALL]*x*)	Дисперсия *n*, исключая пустые значения

DISTINCT гарантирует, что повторяющиеся значения будут учтены только один раз; ALL гарантирует, что будет учтено каждое значение, включая

дубликаты. По умолчанию принята опция ALL, поэтому ее использование необязательно.

Везде, где в функции указано expr, аргументами могут быть типы данных CHAR, VARCHAR2, NUMBER или DATE.

Все агрегатные функции кроме, COUNT(*), игнорируют пустые значения. Чтобы заменить значения для пустых значений, требуется использовать функцию NVL.

Допускается использование вложенных агрегатных функций. Уровень вложенности не ограничен.

Лабораторная работа №9
«SQL: Подзапросы»

Цель работы

Изучение способов применения подзапросов в операторе SELECT для построения сложных выборок.

Теоретические сведения

Типы подзапросов

Существует возможность писать подзапросы в секции WHERE другого SQL-предложения, для того чтобы получить значения на основании некоторого неизвестного значения, задаваемого условием.

Внутренний запрос, или подзапрос, возвращает значение, которое затем используется внешним или основным запросом. Использование подзапроса эквивалентно выполнению двух последовательных запросов, при котором результаты первого запроса используются во втором в качестве критерия поиска.

Подзапрос – это предложение SELECT, которое включается в какую-либо секцию другого предложения SELECT. С помощью подзапросов из простых предложений можно строить сложные запросы. Они могут оказаться полезными в случаях, когда необходимо выбрать строки из таблицы на основании условия, которое в свою очередь зависит от данных в таблице.

Подзапрос может размещаться в следующих секциях:

- WHERE
- HAVING
- FROM

```
SELECT select_list
FROM table
WHERE expr operator
    (SELECT select_list
```

```
FROM table);
```

где

operator – оператор сравнения, такой как >, =, или IN.

В общем случае подзапрос выполняется первым, а затем его результат используется для формирования некоторого условия основного запроса.

Указания:

- Подзапрос должен быть заключен в круглые скобки.
- Подзапрос должен находиться на правой стороне оператора сравнения.
- Подзапрос не может содержать секцию ORDER BY. В предложении SELECT может быть только одна секция ORDER BY, и если она указана, она должна размещаться в конце основного предложения SELECT.

В подзапросах используются два класса операторов сравнения: однострочные операторы и многострочные операторы.

Количество подзапросов в одном предложении не лимитируется Oracle Server, но ограничено размером буфера, используемого запросом.

Существуют различные типы подзапросов:

1. *Однострочные подзапросы* – запрос возвращает только одну строку.
2. *Многострочные подзапросы* – запрос возвращает более одной строки.
3. *Многостолбцовые подзапросы* – запрос возвращает более одного столбца значений.

Однострочный подзапрос возвращает только одну строку в качестве результата выполнения внутреннего предложения SELECT. Этот тип подзапросов использует однострочные операторы сравнения.

Для того чтобы подзапрос возвращал одну строку, можно использовать в нем агрегатные функции.

Распространенной ошибкой при использовании подзапросов является применение однострочного подзапроса, возвращающего более одной строки. Внешний запрос принимает результат подзапроса и использует этот результат в своей секции WHERE. Секция WHERE содержит однострочный оператор сравнения, предполагающий только одно значение. Однострочный оператор не может принять более одного значения от подзапроса, поэтому он генерирует ошибку. Сходная ситуация также возникает, когда подзапрос не возвращает ни одной строки.

Подзапросы, которые возвращают в качестве результата более одной строки, называют многострочными подзапросами. С такими подзапросами используются многострочные операторы сравнения. Многострочный оператор может работать с одним или несколькими значениями.

Многостолбцовые подзапросы

Все однострочные и многострочные подзапросы, которые рассматривались до сих пор, возвращали только один столбец данных, который затем использовался в секциях WHERE или HAVING основного предложения SELECT. Если возникала необходимость в сравнении двух и более столбцов, то

приходилось использовать составную секцию WHERE с применением логических операторов. Многостолбцовые подзапросы позволяют комбинировать несколько условий секции WHERE в одно.

```
SELECT column, column, ...
FROM table
WHERE (column, column, ...) IN
    (SELECT column, column, ...
     FROM table
     WHERE condition);
```

Лабораторная работа №10
«SQL: Составные запросы»

Цель работы

1. Закрепление навыков применения различных способов построения запросов.

2. Комбинирование различных способов построения запросов для получения сложных выборок.

Теоретические сведения

Рассмотрим ряд примеров написания SQL-запросов, перечисленных по мере возрастания сложности.

Приведенные ниже примеры запросов используют следующую схему:

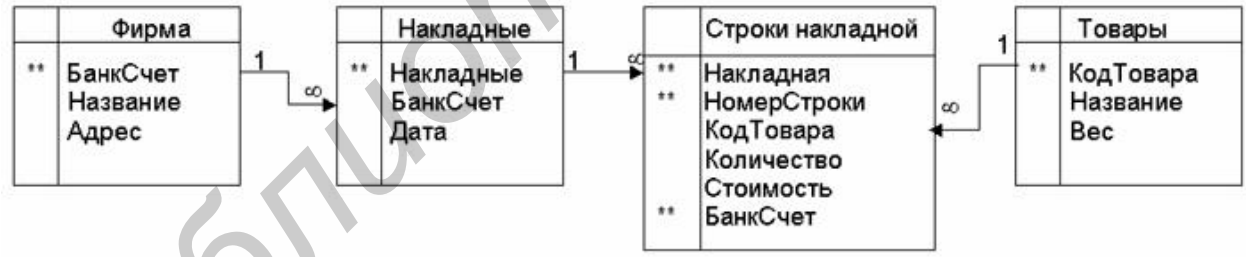

Схема данных «Учет»

Пример 1: Выбрать все товары, вес которых больше 1 кг.

```
SELECT *
FROM Товары
WHERE вес>1;
```

или

```
SELECT КодТовара, Название
FROM Товары
```

```
WHERE вес>1;
```

Пример 2: Выбрать названия товаров, код накладной, номер в строке для всех строк накладной, в которых более 1000 единиц товаров.

```
SELECT СтрокиНакладной.Накладная,
    СтрокиНакладной.НомерСтроки,
    Товары.Название
FROM СтрокиНакладной, Товары
WHERE СтрокиНакладной.КодТовара=Товары.КодТовара
    AND СтрокиНакладной.Количество>1000;
```

Альтернативный синтаксис для соединения – использование ключевого слова JOIN в части FROM.

```
SELECT Накладная, НомерСтроки, Название
FROM СтрокиНакладной NATURAL JOIN Товары ON
WHERE Количество>1000;
```

Пример 3: Вычислить вес товаров, перечисленных в строках накладной (для каждой).

```
SELECT Накладная,
    НомерСтроки,
    Название,
    Количество*Вес AS Вес_В_Строке
FROM СтрокиНакладной NATURAL JOIN Товары;
```

Пример 4: Напечатать все номера накладных, в которых упоминается «сахар».

```
SELECT DISTINCT Накладная
FROM СтрокиНакладной NATURAL JOIN Товары
WHERE Название='сахар';
```

Пример 5: Выдать цену сахара и его количество для каждой строки накладной.

```
SELECT Количество, Стоимость
FROM СтрокиНакладной NATURAL JOIN Товары
WHERE Название='сахар';
```

или

```
SELECT Количество, Стоимость
FROM СтрокиНакладной
WHERE КодТовара=
    (SELECT КодТовара
     FROM Товары
     WHERE Название='сахар');
```

Это пример запроса с подзапросом. Сначала выполняется подзапрос, а затем основной запрос. Подзапрос должен возвращать единственное значение в единственном столбце.

Пример 6: Выдать названия фирм, которые ни разу не поставили сахар.

```
SELECT Название
FROM Фирмы
WHERE БанкСчет NOT IN
    (SELECT DISTINCT БанкСчет
     FROM (Накладные NATURAL JOIN СтрокиНакладной)
         NATURAL JOIN Товары
     WHERE Название='сахар');
```

В отличие от предыдущего примера подзапрос возвращает множество значений, с которым и работает оператор In.

В коррелированном подзапросе есть ссылки на таблицу, которая имеется во внешнем запросе.

Пример 7: Найти все фирмы, которые сделали поставки 11.11.2002.

Вариант 1 (некоррелированный):

```
SELECT DISTINCT Название
FROM Фирмы NATURAL JOIN Накладные
WHERE data='11.11.02';
```

Вариант 2 (коррелированный):

```
SELECT Название
FROM Фирмы
WHERE '11.11.02' IN
    (SELECT Дата
     FROM Накладные
     WHERE Накладные.БанкСчет=Фирмы.БанкСчет);
```

Порядок выполнения запроса следующий:

Сначала выбирается какая-либо фирма. Для нее выполняется подзапрос, в котором находится множество дат, в которые фирма делала поставки. Затем

проверяется условие (11.11.02 находится в этом множестве) и решается, поступит ли фирма в окончательное множество. Эта процедура выполняется для каждой фирмы.

Коррелированные подзапросы сильно снижают производительность, поэтому следует избегать их использования там, где это возможно.

Пример 8: Выдать названия фирм, которые не поставляли товар.

```
SELECT Название
FROM Фирмы
WHERE NOT EXISTS
    (SELECT *
     FROM Накладные
     WHERE БанкСчет=Фирмы.БанкСчет);
```

Предикат EXISTS (NOT EXISTS) проверяет, является ли множество, возвращаемое подзапросом, непустым (пустым).

Пример 9: Выбрать названия товаров, у которых названия одинаковые, а коды разные.

```
SELECT DISTINCT Название
FROM Товары, Товары Товары2
WHERE Товары.Название=Товары2.Название AND
    Товары.Код<Товары2.Код;
```

Если необходимо соединить таблицу саму с собой (self join), то возникает следующая проблема: необходимо синтаксически различать две различные ее копии. Для этого используется переименование. В нашем примере Товары переименованы в Товары2.

Пример 10: Выдать количество поставок, которые фирмы сделали в сентябре.

```
SELECT Название, COUNT(Накладная) AS КоличествоПоставок
FROM Фирмы NATURAL JOIN Накладные
WHERE Дата BETWEEN '01.09.02' AND '31.09.02'
GROUP BY БанкСчет, Название;
```

Рассмотрим множество строк, которое возвращается оператором Select без части Group By. Все это множество разбивается на подмножества таким образом, что все записи с одинаковым банковским счетом попадают в одно подмножество. Над каждым подмножеством выполняется агрегирующая функция (для этого примера – количество элементов подмножества), результат которой и возвращается в запросе.

Пример 11: Рассчитать вес поставки.

```
SELECT Накладная, SUM(Количество*Вес) AS ВесПоставки
FROM СтрокиНакладной NATURAL JOIN Товары
GROUP BY Накладная;
```

Пример 12: Для каждой фирмы подсчитать число поставок, вес которых превышает одну тонну.

```
SELECT Название,
    COUNT(Накладные) AS КоличествоБольшихПоставок
FROM Фирмы NATURAL JOIN Накладные
WHERE Накладная IN
    (SELECT Накладная
     FROM СтрокиНакладной NATURAL JOIN Товары
     GROUP BY Накладная
     HAVING SUM(Количество*Вес)>1000)
GROUP BY БанкСчет, Название;
```

В Having части записываются те условия, в которых используется агрегирующая функция.

Пример 13: Среди накладных с пятью строками найти среднее арифметическое номеров строк, содержащих товары с максимальным весом для заданной накладной.

```
SELECT AVG(НомерСтроки)
FROM СтрокиНакладной Нкл1
WHERE Нкл1.Накладная IN
    (SELECT Накладная
     FROM СтрокиНакладной
     GROUP BY Накладная
     HAVING COUNT(*)=5
    )
    AND Нкл1.НомерСтроки IN
    (SELECT НомерСтроки
     FROM СтрокиНакладной Нкл2 NATURAL JOIN Товары Тв
     WHERE Нкл1.Накладная=Нкл2.Накладная AND
          ВесТовара*Количество=
          (SELECT MAX(ВесТовара*Количество)
           FROM СтрокиНакладной NATURAL JOIN Товары
           WHERE СтрокиНакладной.Накладная=
               Нкл2.Накладная
          )
    );
```

Пример 14: Напечатать названия фирм, у которых были накладные со следующими свойствами: общий вес сахара больше 10 тонн, общий вес масла до одной тонны.

```
SELECT Название
FROM Фирмы
WHERE EXISTS
    (SELECT Накладная
     FROM Накладные
     WHERE Накладные.БанкСчет=Фирмы.БанкСчет AND
         10000<
         (SELECT SUM(Количество*Вес)
          FROM СтрокиНакладной NATURAL JOIN Товары
          WHERE Товары.Название='Сахар' AND
              Накладные.Накладная=
                  СтрокиНакладной.Накладная
          GROUP BY СтрокиНакладной.Накладная
         )
         AND 1000>
         (SELECT SUM(Количество*Вес)
          FROM СтрокиНакладной NATURAL JOIN Товары
          WHERE Товары.Название='Масло'
              AND Накладные.Накладная=
                  СтрокиНакладной.Накладная
          GROUP BY СтрокиНакладной.Накладная
         )
    );
```

Лабораторная работа №11
«PL/SQL: Переменные, анонимный блок, неявный курсор»

Цель работы

1. Изучение основных элементов программы на PL/SQL.
2. Приобретение практических навыков написания простейших программ, использующих выборку данных из таблиц БД.

Теоретические сведения

Блоки

PL/SQL – процедурное расширение языка SQL.

Блок – базовая программная единица, состоящая из декларативной части, процедурной части, обработки исключительных ситуаций. Неименованный

блок PL/SQL имеет три раздела: Declaration (объявления), Body (тело) и, как правило, Exception (исключения).

Стандартная конструкция неименованного блока:

```
DECLARE
-- объявления
...
BEGIN
-- выполняемый код
...
EXCEPTION
-- обработка исключений
...
END;
```

Переменные

Переменные объявляются в декларативной части блока.

```
identifier [CONSTANT] datatype [NOT NULL]
[:= | DEFAULT expr];
```

где

identifier – имя переменной;

CONSTANT – служит для определения констант; константы должны быть инициализированы;

datatype – тип данных; допустимы те же типы, что и используемые для столбцов таблиц;

NOT NULL – то же, что и ограничение для столбца таблицы.

expr – выражение с типом datatype; служит для задания начального значения;

При объявлении переменных можно пользоваться атрибутами %TYPE и %ROWTYPE. Первый задает тип объекта, к которому применен.

Например: student.FIO%TYPE – тип столбца FIO таблицы student.

Второй — тип-запись всей строки таблицы.

Например: student%ROWTYPE – тип-запись с полями, названия и типы которых совпадают с названиями и типами столбцов таблицы student.

Неявный курсор

В исполняемой части блока можно выполнять любые инструкции DML, SELECT, а также управлять транзакциями с помощью COMMIT, ROLLBACK, SAVEPOINT.

Оператор SELECT внутри блока дополняется секцией INTO для сохранения результатов выполнения запроса, т.е.

```
SELECT id, FIO
INTO v_id, v_FIO
FROM student
...
```

Очевидно, такой оператор должен выбирать не более одной строки. Без секции INTO применение оператора внутри блока недопустимо.

При выполнении в блоке любой инструкции SQL сервер создает область памяти для обработки этой инструкции. Эта область называется курсором. Курсор может создаваться программистом явно с помощью специальных конструкций языка или неявно – при выполнении одиночных операций INSERT, UPDATE, DELETE, SELECT INTO.

Лабораторная работа №12
«PL/SQL: Циклы, условия, явный курсор»

Цель работы

1. Изучение основных управляющих конструкций программы на PL/SQL.
2. Приобретение практических навыков написания программ, использующих ветвление, циклическую обработку и последовательное извлечение данных многострочных выборок с использованием явного курсора.

Теоретические сведения

Условия

```
IF condition THEN
statements;
[ELSIF condition THEN
statements;]
[ELSE
statements;]
END IF;
```

где

condition – логическая переменная или выражение (TRUE, FALSE, NULL);

statements – любое количество предложений PL/SQL.

Количество секций ELSIF не ограничено.

Циклы

```
LOOP
statement1;
. . .
EXIT [WHEN condition];
END LOOP;
```

EXIT – выход из цикла. Если указана секция WHEN, выход осуществляется, если *condition* принимает значение TRUE.

```
FOR counter in [REVERSE]
lower_bound..upper_bound LOOP
statement1;
statement2;
. . .
END LOOP;
```

где

counter – неявно объявляемая в точке начала цикла целочисленная переменная, которая увеличивается (уменьшается, если используется REVERSE) на 1 на каждой итерации;

lower_bound – наименьшее значение переменной цикла;

upper_bound – наибольшее значение переменной цикла.

Независимо от использования REVERSE меньшее значение указывается первым.

```
WHILE condition LOOP
statement1;
statement2;
. . .
END LOOP;
```

где

condition – логическая переменная или выражение (TRUE, FALSE, NULL);

statements – любое количество предложений PL/SQL.

Явный курсор

Явный курсор применяется, если необходимо обработать выборку из более чем одной строки. Последовательность действий при работе с явным курсором включает следующие этапы:

1. Объявление.

```
CURSOR cursor_name[(params)] IS
select_statement;
```

где

cursor_name – имя курсора;

params – список входных параметров (если таковые требуются);

select_statement – запрос, на основании которого осуществляется выборка строк. Запрос может быть параметризованным, для чего используются входные параметры из *params.*

2. Открытие.

```
OPEN cursor_name[(params)];
```

где

params – список фактических параметров для параметризованного запроса.

3. Выборка данных.

```
FETCH cursor_name INTO [variable1, variable2, ...]
| record_name];
```

Выборка осуществляется либо в список переменных с типами, соответствующими полям выборки запроса курсора, либо в переменную-запись, имеющую такую же структуру, как и выборка запроса (для объявления такой переменной удобно использовать cursor_name%ROWTYPE).

4. Закрытие курсора

```
CLOSE cursor_name;
```

По окончании выполнения блока все открытые в нем курсоры будут автоматически закрыты. Курсор можно закрыть явно командой CLOSE. Это полезно, если необходимо обработать выборку курсора еще раз. Для этого после закрытия курсор снова открывается командой OPEN.

Атрибуты курсора

%ISOPEN – TRUE, если курсор открыт;

%NOTFOUND – TRUE, если последняя выборка данных не вернула строки;

%FOUND –TRUE, если последняя выборка данных вернула строку;

%ROWCOUNT – количество выбранных на данный момент строк.

Курсорный цикл

Цикл FOR допускает расширение, использующее курсор.

```
FOR record_name IN {cursor_name[(params)]
   |(select_statement)}
LOOP
statement1;
statement2;
. . .
END LOOP;
```

где

record_name – неявно объявляемая в точке начала цикла переменная типа запись, со структурой полей курсора или запроса (см. ниже); в ходе выполнения цикла переменная последовательно принимает значения всех строк используемой выборки;

cursor_name[(params)] – курсор, из которого последовательно берутся значения для записи record_name;

select_statement – вместо курсора можно непосредственно указать запрос SELECT.

Следует помнить, что при использовании курсорного цикла переменная цикла всегда имеет тип записи. Если переменная с таким именем уже объявлена (например, со скалярным типом), то это объявление **будет перекрыто внутри цикла**.

```
DECLARE
v_id student.id%TYPE;
BEGIN
v_id:=11;                      ß здесь v_id имеет тип
...                              student.id%TYPE
FOR v_id in
(SELECT id FROM student)       ß внутри цикла v_id
LOOP                             имеет тип запись
...                              с единственным полем id!
    IF v_id=11 ...             ß ошибка!
    IF v_id.id=11 ...          ß правильно
...
END LOOP;
...
END;
```

Лабораторная работа №13 «PL/SQL: Процедуры и функции»

Цель работы

1. Изучение синтаксиса определения именованных подпрограмм.
2. Приобретение практических навыков написания подпрограмм на языке PL/SQL.

Теоретические сведения

Синтаксис

Помимо неименованных блоков PL/SQL позволяет создавать именованные блоки кода. Процедуры и функции хранятся в базе данных. Они могут быть вызваны извне базы данных, из других процедур и функций внутри БД, в SQL-предложениях.

Объявление процедуры в PL/SQL выглядит следующим образом:

```
PROCEDURE name[(parameter[,parameter...])]
IS
[declaration statements]
BEGIN
executable-statements
[ EXCEPTION
```

```
exception handler statements]
END [ name ];
```

где

name – имя процедуры,

parameters – необязательный список параметров, которые определяются для передачи значений между процедурой и вызывающим ее блоком кода.

Назначение других элементов процедуры совпадает с назначением аналогичных элементов неименованного блока.

Определение функции отличается от определения процедуры наличием ключевого слова RETURN, которое служит для определения типа возвращаемого функцией значения. Функции определяются следующим образом:

```
FUNCTION name [ ( parameter [, parameter ... ] ) ]
RETURN return_datatype
IS
[ declaration statements ]
BEGIN
executable statements
[ EXCEPTION
exception handler statements ]
END [ name ];
```

где

name – имя функции;

parameters – необязательный список параметров, которые определяются для передачи значений между функцией и вызывающим ее блоком кода;

return_datatype – тип возвращаемого функцией значения. Это может быть любой из скалярных или составных типов, поддерживаемых в PL/SQL .

Оператор RETURN

Оператор RETURN применяется для возвращения значения из функции:

```
RETURN возвращаемое_значение;
```

Кроме того, этот оператор может использоваться без параметра для завершения выполнения процедуры. После выполнения оператора RETURN выполнение функции/процедуры прекращается и управление передается в вызывающий блок.

Функция в PL/SQL может иметь несколько операторов RETURN, при этом один из них обязательно должен быть выполнен при вызове функции. Если выполнение функции завершается без вызова оператора RETURN, возникает следующая ошибка:

```
ORA-6503: PL/SQL: Function returned without value
```

Типы параметров

Передаваемые параметры могут быть трех типов:

in – входные, значения параметров нельзя изменять, внутри подпрограммы эти параметры используются как константы;

out – выходные, формальным параметрам этого типа можно присвоить значение, которое затем передается в фактические параметры, указанные при вызове подпрограммы; параметры типа out не могут использоваться для передачи значений в подпрограмму, в начале выполнения подпрограммы их значение не определено;

inout – обладают свойствами двух предыдущих типов параметров, могут использоваться как для передачи данных в подпрограмму, так и для возвращения данных из подпрограммы в вызывающий блок.

PL/SQL поддерживает два способа сопоставления формальных и фактических параметров: позиционный и именной.

При позиционной передаче соответствие фактических параметров формальным определяется порядком перечисления фактических параметров при вызове подпрограммы.

При именной передаче соответствие параметров задается явно с помощью символа => следующим образом:

```
sub_name(formal_parameter_name => argument_value, ...)
```

В одном вызове подпрограммы нельзя смешивать различные способы передачи параметров.

Лабораторная работа №14
«PL/SQL: Триггеры»

Цель работы

1. Изучение синтаксиса определения триггеров.
2. Приобретение практических навыков написания простейших триггеров обработки событий изменения данных.

Теоретические сведения

Триггеры представляют собой именованные блоки, которые выполняются в ответ на ряд определенных событий БД или при изменении данных в таблицах. Существует заранее определенный набор событий, которые могут быть связаны с триггером. В данной работе рассматриваются только события, связанные с изменением данных.

Создание триггера

Синтаксис создания триггера выглядит следующим образом:

```
TRIGGER trigger_name
BEFORE | AFTER | INSTEAD OF
DELETE | INSERT | UPDATE [OF column [, column] ...]
[OR DELETE | INSERT | UPDATE [OF column [, column] ...]]
...
ON table | DATABASE
FOR EACH ROW
[WHEN (condition)]]
PL/SQL_block;
```

где

trigger_name – имя триггера;

UPDATE OF column – триггер активизируется при выполнении оператора UPDATE на одном из перечисленных столбцов;

table – имя таблицы, для которой создается триггер;

condition – условие срабатывания триггера;

PL/SQL_block – стандартный блок PL/SQL, описывающий тело триггера.

События DML

События DML включают операторы INSERT, UPDATE или DELETE.

Триггеры бывают двух уровней: уровня оператора и уровня строки. Триггер уровня оператора срабатывает по одному разу для каждого оператора, модифицирующего таблицу. Иначе говоря, если таблица изменяется оператором Delete, который удаляет много строк, то триггер будет вызван единственный раз для этого оператора. Триггер уровня строки срабатывает для каждой строки, модифицируемой оператором.

Если один триггер используется для обработки нескольких событий, то в теле триггера можно использовать предикаты INSERTING, UPDATING, DELETING для определения, каким событием был активирован триггер:

```
IF INSERTING THEN ...
```

Триггер может срабатывать до (BEFORE) после (AFTER) или вместо (INSTEAD OF) активировавшего его события. При этом триггеры INSTEAD OF всегда вызываются для каждой обработанной строки, т.е. являются триггерами уровня строки. Порядок, в котором выполняются различные виды триггеров, выглядит следующим образом:

1. BEFORE – уровень оператора.
2. FOR EACH ROW:
    a) BEFORE – уровень строки;

b) событие, активирующее триггер / INSTEAD OF триггер;
c) AFTER – уровень строки.

3. AFTER – уровень оператора.

Псевдозаписи :new и :old

Во время выполнения триггера уровня строки операторы PL/SQL и предложения SQL могут получать доступ к старым и новым значениям столбцов текущей строки, обрабатываемой оператором, вызвавшим срабатывание триггера. Существуют две псевдозаписи для текущей строки изменяемой таблицы: одна – для старого значения строки (:old), и одна – для нового значения (:new).

```
IF :new.Sal > 10000 ...
IF :new.Sal < :old.Sal ...
```

В зависимости от события, вызвавшего триггер, использование этих записей может быть различным:

INSERT: триггер имеет доступ только к записи *:new*. Так как обрабатываемая триггером строка создается оператором INSERT, старых значений не существует, и поля записи *:old* имеют значение NULL;

UPDATE: триггер имеет доступ к обеим записям;

DELETE: триггер имеет доступ только к записи *:old*, так как строка больше не существует после удаления. Столбцы записи *:new* имеют значение NULL. Попытка изменять значение столбцов *:new* приведет к ошибке ORA-4084.

Значения :old и :new доступны для триггеров уровня строки, объявленных как BEFORE или AFTER. Значения столбцов записи :new могут быть изменены (например присваиванием) в триггерах BEFORE, при этом эти изменения вносятся в модифицируемую таблицу. Эту запись нельзя изменять в триггерах AFTER, так как событие, вызывающее триггер, происходит до срабатывания триггера. Если триггер BEFORE изменяет значение столбца :new, то это изменение будет видно в триггере AFTER, сработавшим на то же событие.

ЛИТЕРАТУРА

1. Дейт, К. Дж. Введение в системы баз данных / К. Дж. Дейт. – Киев : Диалектика, 1998.

2. Гарсиа-Молина, Г. Системы баз данных / Г. Гарсия-Молина, Дж. Ульман, Дж. Уидом. – М. : Издательский дом «Вильямс», 2003.

3. Мейер, Д. Теория реляционных баз данных / Д. Мейер. – М. : Мир. 1987.

4. Терьо, М. 101 Oracle 9i DBA. Администрирование баз данных / М. Терьо. – М. : Лори, 2005.

5. Кайт, Т. Oracle для профессионалов / Т. Кайт. – М. : «Вильямс», 2007.

6. Миллсап, К. Oracle. Оптимизация производительности / К. Миллсап, Д. Хольт. – СПб. : Символ-Плюс, 2005.

7. Гринвальд, Р. Oracle. Справочник / Р. Гринвальд, Д. Крейнс. – СПб. : Символ-Плюс, 2005.

8. Мишра, С. Секреты Oracle SQL / С. Мишра, А. Бьюли. – СПб. : Символ-Плюс, 2003.

Св. план 2007, поз. 65

Учебное издание

Тараканов Александр Николаевич

МОДЕЛИ ДАННЫХ И СУБД

Методическое пособие
к лабораторным работам
для студентов специальности I-31 03 04 «Информатика»
всех форм обучения

Редактор Т. П. Андрейченко
Корректор Е. Н. Батурчик

Подписано в печать 08.02.2008. Формат 60×84 1/16. Бумага офсетная.
Гарнитура «Таймс». Печать ризографическая. Усл. печ. л. 3,37.
Уч.-изд. л. 2,8. Тираж 125 экз. Заказ 331.

Издатель и полиграфическое исполнение: Учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники».
ЛИ № 02330/0056964 от 01.04.2004. ЛП № 0233/0131666 от 30.04.2004.
220013, Минск, П.Бровки, 6