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

Факультет информационных технологий и управления

Кафедра систем управления

М. А. Крупская

ИНФОРМАЦИОННОЕ ОБЕСПЕЧЕНИЕ СИСТЕМ УПРАВЛЕНИЯ. ЛАБОРАТОРНЫЙ ПРАКТИКУМ

Рекомендовано УМО по образованию в области информатики и радиоэлектроники в качестве учебно-методического пособия для специальности 1-53 01 07 «Информационные технологии и управление в технических системах»

Минск БГУИР 2019

УДК 004.65(076.5)
ББК 32.972.134я73
К84

Р е ц е н з е н т ы:

кафедра информационных систем и технологий Международного института дистанционного образования Белорусского национального технического университета (протокол №8 от 15.04.2019);

доцент кафедры управления информационными ресурсами Академии управления при Президенте Республики Беларусь кандидат технических наук, доцент Н. И. Белодед

Крупская, М. А.

К84 Информационное обеспечение систем управления. Лабораторный практикум : учеб.-метод. пособие / М. А. Крупская. – Минск : БГУИР, 2019. – 147 с. : ил.

ISBN 985-985-543-512-0.

Приведены описание и порядок выполнения шести лабораторных работ по второй части курса «Информационное обеспечение систем управления». В каждой лабораторной работе сформулирована цель, изложены краткие теоретические сведения, сопровождаемые примерами и пояснениями, разработаны задания для самостоятельного выполнения и контрольные вопросы.

УДК 004.65(076.5)
ББК 32.972.134я73

ISBN 985-985-543-512-0

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Язык структурированных запросов – SQL (Structure Query Language) – это стандартный язык управления реляционными базами данных (РБД). Реляционная база данных – это база данных (БД), разделенная на логически цельные сегменты, называемые таблицами, которые внутри БД связаны между собой, что обеспечивает оптимальное представление данных и возможность организации нескольких уровней доступа.

Прототип языка структурированных запросов был разработан фирмой IBM на основе предложений доктора Эдгара Франка Кодда в статье «Реляционная модель данных для больших банков данных общего пользования». В 1979 году появился первый коммерческий продукт с использованием SQL под названием Oracle, который был выпущен компанией Relational Software, Incorporated (впоследствии переименованной в Oracle Corporation). Сегодня компания Oracle является одним из лидеров в области реализации технологий РБД. В первый раз SQL был утвержден стандартным языком в области управления базами данных в 1986 году. Основные виды команд, реализующих в SQL выполнение различных функций, описаны ниже.

Язык определения данных (DDL) предоставляет пользователю возможность создавать различные объекты базы данных (TABLE, INDEX, VIEW, SYNONYM и т. д.) и переопределять их структуру. Основные команды DDL: CREATE, ALTER, DROP.

Язык манипуляций данными (DML) предоставляет пользователю возможность манипулировать данными внутри объектов РБД. Основные команды DML: INSERT, UPDATE, DELETE.

Язык запросов к данным (DQL) является самой важной частью SQL. Команда SELECT, имеющая множество опций и необязательных параметров, используется для построения запросов к РБД. С ее помощью можно конструировать запросы любой сложности.

Команды языка управления данными (DCL) обычно используются для создания объектов, относящихся к управлению доступом пользователей, а также для назначения пользователям уровней привилегий доступа к данным в базе. Основные команды DCL: GRANT, REVOKE.

Команды администрирования данных дают пользователю возможность выполнять аудит и анализ операций внутри базы данных. Эти команды могут также помочь при анализе производительности системы данных.

Команды управления транзакциями используются для сохранения или отмены транзакции, основные из них: COMMIT, ROLLBACK, SAVEPOINT.

Язык SQL в настоящее время поддерживается большинством систем управления базами данных (СУБД) с различными отклонениями от стандарта, регламентирующего общие правила написания SQL-инструкций. Главным отличием языка SQL от других языков программирования является его «непроцедурность», т. е. посредством инструкций SQL просто указывается что сделать и ка-

кая информация из БД необходима, а как именно и откуда она извлекается определяется СУБД. Поэтому в любой СУБД для полноценной работы с базой всегда существует программное расширение SQL.

Так, PL/SQL – это процедурно-ориентированный язык, созданный для облегчения обработки команд SQL в СУБД Oracle. Его можно использовать для написания хранимых процедур, функций, исполняемых в этой СУБД. Кроме того, этот язык можно применять и в клиентских приложениях [1].

Лабораторный практикум включает шесть лабораторных работ: три по основам работы с SQL и три – с PL/SQL, которые выполняются через интерфейс командной строки SQL*Plus. SQL*Plus – это программа-интерпретатор командной строки для работы с СУБД Oracle, в которой могут выполняться команды SQL и PL/SQL в интерактивном виде или из сценария, доступен практически в любой установке программного обеспечения Oracle. Чтобы начать выполнение лабораторных работ, необходимо иметь реляционную модель БД, созданную на этапе изучения первой части курса ИОСУ в предыдущем семестре.

В лабораторном практикуме используется пакет Oracle Database 11g Express Edition (Oracle Database XE), который является единственной свободно распространяемой версией СУБД Oracle и доступен для скачивания после регистрации на сайте корпорации https://www.Oracle.com. Работа с СУБД, кроме командной строки SQL*Plus, может выполняться с помощью интуитивно понятного веб-интерфейса браузера (так называемый iSQLPlus) и позволяет выполнять все основные операции по созданию таблиц баз данных, установлению связей между таблицами, вводу данных, созданию запросов, отчетов, администрированию пользователей, а также писать программы на PL/SQL. Кроме того, можно не устанавливать сервер СУБД Oracle Database XE 11g, а использовать Oracle Application Express (Oracle Apex) – проприетарную среду быстрой разработки прикладного программного обеспечения на основе СУБД Oracle Database, целиком реализованную как веб-приложение и предоставляющую возможность получить 25 ГБ пространства на облачном сервере Oracle по ссылке https://apex.Oracle.com.

ЛАБОРАТОРНАЯ РАБОТА №1

СОЗДАНИЕ И ЗАПОЛНЕНИЕ БАЗЫ ДАННЫХ В СУБД ORACLE

Цель работы – ознакомиться с программой-интерпретатором командной строки SQL*Plus и основными элементами языка SQL; приобрести навыки создания таблиц, индексов, синонимов и последовательностей, простых ограничений столбцов, вставки и удаления строк, работы со словарем данных.

Теоретические сведения

Общие сведения об SQL. Алфавит языка SQL включает следующие символы:

1) буквы: A…Z, a…z;
2) цифры: 0…9;
3) символы: + - * / ! @ $ # = <>^ ' " ( ) | _ ; , .

Длина идентификаторов SQL может достигать 30 символов, они обязательно начинаются с буквы и могут включать в себя цифры, а также символы доллара ($), решетки (#) и подчеркивания (_). Исключение составляют имена БД, которые ограничены до восьми символов. В некоторых версиях СУБД Oracle допускается использование русских букв при соответствующих изменениях настроек сервера БД. Имя любого объекта может дополнительно включать имя схемы: [схема.]имя_объекта. Схема представляет собой набор объектов, принадлежащих конкретному пользователю, и идентифицируется его именем. Среди объектов схемы могут быть таблицы, представления, индексы, последовательности, триггеры, процедуры и функции, пакеты.

Символьные литералы записываются в одинарных кавычках: 'test'. При необходимости присутствия одинарной кавычки внутри символьного литерала она удваивается или используется квалификатор.

Числовые литералы представляют собой целое или действительное значение со знаком или без знака, при этом действительные значения могут быть записаны в формате с десятичной точкой или в экспоненциальной форме.

В SQL имеется специальное неопределенное значение NULL. По смыслу оно не эквивалентно понятию пустая строка для символьных типов и нулевому значению для числовых типов. Если в некотором столбце таблицы данные отсутствуют, говорят, что его значение NULL. Столбец с данными любого типа может содержать значение NULL, если только он специально не описан с ограничением NOT NULL. Однако следует помнить, что в Oracle проверка на сравнение пустого литерала IS NULL вернет TRUE.

В SQL могут использоваться псевдостолбцы – формируемые системой столбцы, имеющие стандартные имена. Их значения можно только просматри-

вать и использовать, но корректировать (добавлять, удалять, изменять) нельзя. К псевдостолбцам относятся: ROWID, ROWNUM, LEVEL, CURVAL, NEXTVAL.

Псевдостолбец ROWID содержит уникальные для всей базы данных физические адреса строк таблиц. Значение псевдостолбца ROWID определяется при вставке строки в таблицу и не изменяется, пока строка присутствует в таблице.

Псевдостолбец ROWNUM определяет порядковый номер строки, выбранной из таблицы при выполнении запроса. Он обычно используется для ограничения числа строк, выбираемых из таблицы запросом.

Псевдостолбец LEVEL возвращает уровень вложенности данных, позволяя тем самым строить запросы для получения информации об иерархии данных. Используется только в иерархических запросах вместе с предложением CONNECT BY.

Псевдостолбцы CURVAL, NEXTVAL предназначены для работы с текущим и следующим значениями последовательности.

Типы данных. К наиболее часто используемым типам данных относятся символьные, числовые, тип дата/время, двоичные и большие объекты.

Символьные типы данных представлены следующими основными типами: CHAR[длина], VARCHAR2(длина) и LONG.

Тип данных CHAR представляет собой символьные строки фиксированной длины. Минимальная длина равна 1 байт, максимальная – 2000 байт. Если значение, помещаемое в столбец данного типа, превосходит указанный размер, то выводится сообщение об ошибке; если длина помещаемого значения меньше указанной длины, то значение дополняется пробелами справа.

Тип данных VARCHAR2 представляет собой символьные строки переменной длины. Максимальный размер строки – 4000 байт, минимальный – 1 байт. При помещении текста в столбец большего размера дополнение пробелами не производится.

Строки этих двух символьных типов сравниваются по-разному. Строки типа CHAR – посимвольно с дополнением пробелами строки с меньшей длиной до размера строки с большей длиной. Строки VARCHAR2 – без дополнения пробелами до большей длины. Поэтому для двух одинаковых строк могут быть получены различные результаты при их сравнении. Например, для двух строк 'AB' и 'AB ' (вторая строка содержит пробел, а первая нет) типа CHAR получим, что 'AB' = 'AB'. Для этих же строк типа VARCHAR2 результатом сравнения будет 'AB' < 'AB '.

Тип данных LONG представляет собой символьные данные переменной длины, величина которой может достигать 2 ГБ. На использование переменных этого типа накладывается ряд ограничений и чаще вместо этого типа используется объектный символьный тип CLOB.

Числовые типы представлены типом NUMBER, который позволяет определить три различных типа данных:

1) NUMBER;
2) NUMBER(p);
3) NUMBER(p, s).

В первом случае определяются действительные числа, диапазон которых от $1{,}0 \cdot 10^{-130}$ до $1{,}0 \cdot 10^{126}-1$, мантисса содержит 38 знаков. Во втором случае считается, что определяется диапазон целых чисел, где p – количество цифр в числе (от 1 до 38). В третьем случае описываются числа с фиксированной точкой; p – общее количество цифр (от 1 до 38), s – масштаб (от –84 до 127) – определяет количество цифр после запятой. Если $s > 0$, то число округляется до указанного числа знаков справа от десятичной точки, если $s < 0$, то число округляется до указанного числа знаков слева от десятичной точки.

Следует отметить, что язык SQL поддерживает типы данных стандарта ANSI. Если такой тип данных (INTEGER, SMALLINT, DECIMAL, FLOAT, REAL и т. д.) встречается при определении типа столбца таблицы, то имя типа сохраняется, но сами данные хранятся в виде, определяемом одним из типов базы данных Oracle.

Тип данных дата/время. Тип DATE представляет собой специально организованный тип, который хранит столетие, год, месяц, день, часы, минуты, секунды в одном поле. Для выборки текущей даты используется функция SYSDATE, например:

```
SELECT SYSDATE “Текущее время” FROM dual;
```

Чтобы увидеть другие составляющие поля или изменить формат вывода, применяется встроенная функция TO_CHAR и шаблон, например:

```
SELECT TO_CHAR(SYSDATE,'DD.MM.YY HH24:MI:SS') "Время и дата"
FROM dual;
```

Над переменными типа DATE можно выполнить арифметическое действие – вычитание. Результат операции определяет количество дней между этими двумя датами. К дате можно прибавить или отнять от нее числовую константу, рассматриваемую как количество дней или часть дня.

В прил. 1 (табл. П.1.1–П.1.3) приведены основные встроенные функции Oracle для работы с символьными, числовыми полями и полями типа дата/время.

Двоичные типы данных используются для хранения двоичных неструктурированных данных (звуковые файлы, файлы изображений и т. д.), обработка которых не поддерживается Oracle. К ним относятся типы RAW(длина) и LONGRAW. Максимальный размер строки типа RAW 2000 байт, минимальный – 1 байт. Длина переменных типа LONGRAW может достигать 2 ГБ.

К большим объектам (*LOB-объектам*) относятся CLOB, BLOB и BFILE. Они предназначены для хранения неструктурированных данных большого

объема – до 4 ГБ. Тип CLOB хранит данные символьного типа, типы BLOB и BFILE используются для хранения двоичных данных. Столбцы типа LOB содержат не сами данные, а указатели на их местоположение.

Создание таблиц. Создание таблицы выполняется с помощью DDL-инструкции CREATE TABLE, упрощенный синтаксис которой имеет следующий вид:

```
CREATE TABLE имя_таблицы
{({ имя_столбца тип_данных [DEFAULT значение]
[ограничения_столбца] | ограничение_таблицы}
[,{ имя_столбца тип_данных [DEFAULT значение]
[ограничения_столбца] | ограничение_таблицы}]...) |
AS подзапрос};
```

Таблица может быть создана либо стандартным образом через описание ее компонент, либо в результате выполнения некоторого подзапроса. Подзапрос – это обычный запрос на выборку информации, реализуемый оператором SELECT. При создании таблицы задаваемые имена таблиц и имена столбцов должны удовлетворять правилам, предписываемым идентификаторам. При этом имена, присваиваемые таблицам, должны быть уникальными в схеме пользователя, а имена столбцов должны быть уникальными в рамках одной таблицы. Для каждого столбца указываются тип данных и, если необходимо, значение, вставляемое в столбец по умолчанию (DEFAULT значение). Важным элементом при создании таблиц является задание ограничений целостности данных, которые позволяют отслеживать правильность модификации имеющихся данных или вставляемых в таблицу новых данных. Ограничения целостности данных делятся на два типа: ограничения столбца и ограничения таблицы. Ограничения столбца позволяют определить условия, которым должны удовлетворять значения соответствующего столбца; ограничения целостности данных, накладываемые на таблицу, позволяют проверить правильность всех добавляемых или модифицируемых строк таблицы. Ограничение может быть именованным или безымянным. Удобнее использовать именованные ограничения, поскольку при выдаче информации, связанной с возникшим нарушением одного из ограничений, выдается и имя этого ограничения, что очень удобно для исправления ошибок в дальнейшем. Задание ограничений на столбец или таблицу осуществляется по следующему синтаксису:

```
[CONSTRAINT имя_ограничения] тип_ограничения
```

Существуют следующие типы ограничений, накладываемых на столбец:

1. Первичный ключ (PRIMARY KEY) – это ограничение требует, чтобы вводимые в столбец значения были уникальными и отличными от пустых (NULL), поскольку они будут использоваться в качестве первичного ключа, од-

нозначно идентифицирующего запись; первичный ключ определяется для таблицы только единожды и поддерживает ее целостность.

2. Уникальность (UNIQUE) – это ограничение требует, чтобы вводимые в столбец значения в рамках одной таблицы были уникальными.

3. Обязательное наличие данных (NOT NULL) – это ограничение требует обязательного присутствия в столбце некоторого значения, т. е. не допускаются значения NULL при заполнении таблицы. Обязательное наличие данных в столбце определяется разработчиком на стадии проектирования таблицы и зависит только от его смысловой нагрузки в данной таблице.

4. Условие на значение CHECK(выражение) – это ограничение позволяет подвергнуть определенной проверке вставляемое в столбец значение; если условия, наложенные на вставляемые значения, не выполняются, то значения в столбец не помещаются.

Рассмотрим пример, в котором для поля номера сотрудника sno создается ограничение с именем s1, которое позволяет указывать в качестве номера только числа от 101 до 199:

```
CREATE TABLE staff (sno INTEGER not null,
                    age INTEGER CHECK (age >=21),
                CONSTRAINT s1 CHECK (sno BETWEEN 101 AND 199));
```

5. Внешний ключ таблицы (FOREIGN KEY) позволяет установить взаимосвязь значений указанного столбца со значениями столбца другой таблицы при помощи ключевого слова REFERENCES. Таблица, на чей столбец ссылается другая таблица, называется главной или родительской, а таблица, ссылающаяся на нее, – подчиненной или дочерней. Взаимосвязь обеспечивается использованием следующей конструкции:

```
REFERENCES имя_таблицы-родителя[(имя_столбца)]
[ON DELETE CASCADE | ON DELETE SET NULL]
```

При внесении значения в столбец создаваемой таблицы система будет автоматически проверять наличие аналогичного значения в указанном столбце таблицы-родителя. При этом для обеспечения однозначности устанавливаемой взаимосвязи все значения, находящиеся в столбце, на которые производится ссылка, должны иметь ограничение UNIQUE или PRIMARY KEY. Если в качестве имени столбца родительской таблицы используется первичный ключ, то имя столбца можно не указывать. Конструкция ON DELETE CASCADE указывает, что при удалении строк в главной таблице автоматически осуществляется удаление соответствующих строк и в подчиненной таблице; ON DELETE SET NULL указывает, что при удалении родительских строк на месте их значений в дочерней таблице будет установлено значение NULL.

Ограничения на таблицу во многом напоминают ограничения столбца, но при этом задействуют, как правило, несколько столбцов. Также ограничение

столбца помещается в конец описания определения данного столбца после типа данных. Ограничение таблицы помещается в конец описания таблицы после последнего имени столбца, но перед заключительной круглой скобкой. Например, можно задать ограничение PRIMARY KEY на уровне таблицы, указав список имен столбцов, тем самым определив составной первичный ключ.

Можно определить составной внешний ключ для таблицы. В случае составного внешнего ключа перечень столбцов в подчиненной таблице и перечень столбцов в главной таблице должны совпадать по количеству, типу данных и порядку следования. Например:

```
FOREIGN KEY (список_имен_столбцов)
REFERENCES имя_таблицы(список_имен_столбцов)
[ON DELETE CASCADE]
```

Если ограничение CHECK затрагивает значения нескольких столбцов, увязывая их в некоторое достаточно сложное условие, то такое ограничение также удобно определить как ограничение на таблицу [2].

Рассмотрим код создания таблицы заказов. На рис. 1 приведена реляционная модель БД, включающая родительские таблицы customers (продавцы), salesreps (покупатели) и product (продукция), а также дочернюю таблицу orders (заказы).

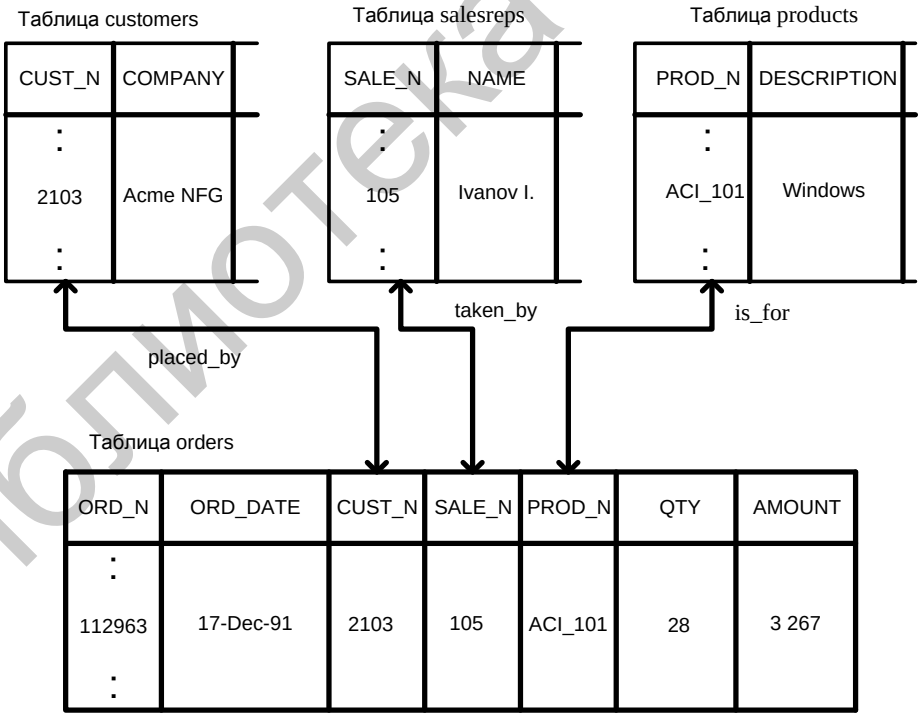

Рис. 1. Реляционная модель БД

Инструкция CREATE TABLE создаст таблицу orders при условии, что родительские таблицы, с которыми она связана к моменту ее создания, уже существуют. Если таблиц, на которые ссылается данная таблица, нет, то связи с ней могут быть созданы позже при помощи инструкции ALTER TABLE.

```
CREATE TABLE orders (ord_n INTEGER NOT NULL,
ord_date DATE NOT NULL,
cust_n VARCHAR2(5) NOT NULL,
sale_n VARCHAR2(5) NOT NULL,
prod_n VARCHAR2(10) NOT NULL,
qty INTEGER,
amount NUMBER (6,2),
PRIMARY KEY (ord_n),
CONSTRAINT placed_by FOREIGN KEY (cust_n) REFERENCES customers,
CONSTRAINT taken_by FOREIGN KEY (sale_n) REFERENCES salesreps,
CONSTRAINT is_for FOREIGN KEY (prod_n) REFERENCES products);
```

Для изменения структуры таблицы используется оператор ALTER TABLE, с помощью которого можно осуществить добавление столбцов, ограничений столбцов или таблицы (предложение ADD), изменение определения столбцов (предложение MODIFY), удаление столбцов, ограничений столбца или таблицы (предложение DROP), переименование столбцов (предложение RENAME). Рассмотрим несколько примеров.

1. Добавить новый столбец в таблицу orders:

```
ALTER TABLE orders ADD new_column VARCHAR2(10);
```

2. Изменить новый столбец в таблице orders, увеличив его размер до 30 байт:

```
ALTER TABLE orders MODIFY new_column VARCHAR2(30);
```

Изменять размер пустых столбцов можно как в большую, так и в меньшую сторону, однако при наличии в таблицах уже введенных строк изменять размер поля можно только в сторону увеличения.

3. Удалить столбец из таблицы orders можно следующим образом:

```
ALTER TABLE orders DROP COLUMN old_column;
```

4. Чтобы переименовать столбец в таблице orders, используется запись:

```
ALTER TABLE orders RENAME COLUMN old_column TO new_column;
```

Удаление таблицы можно выполнить с помощью инструкции:
DROP TABLE имя_таблицы [CASCADE CONSTRAINTS];

При наличии конструкции CASCADE CONSTRAINTS вместе с удалением таблицы уничтожаются ограничения внешнего ключа в других таблицах.

Вставка строк в таблицу осуществляется с помощью DML-инструкции INSERT, которая имеет следующий синтаксис:

```
INSERT INTO имя_таблицы [(список_столбцов)]
{VALUES (значение1 [, значение2] ...) | подзапрос};
```

Если список столбцов не указывается, то список значений в предложении VALUES должен содержать значения для всех столбцов таблицы, причем порядок их следования должен однозначно соответствовать порядку их следования в строке. Кроме этого, должно быть соответствие между типами данных столбцов и значениями, передаваемыми в предложении VALUES, в частности, символьные литералы и литералы типа дата/время заключаются в одинарные кавычки. Если формат вводимых данных отличается от текущего в сессии, то при вводе используют встроенную функцию TO_DATE для приведения данных к нужному виду. Например, TO_DATE ('12/12/2012', 'DD.MM.YYYY').

Также инструкция INSERT INTO может использоваться с подзапросом, что позволяет перенести строки из некоторой существующей таблицы в создаваемую таблицу:

```
INSERT INTO имя_таблицы [(список_столбцов)]
SELECT запрос;
```

Кроме того, следует помнить, что INSERT INTO, как любая DML – инструкция в SQL*Plus выполняется только в оперативной памяти текущего сеанса, т. е. в БД не отразятся все изменения, пока не будет явного подтверждения выполнения транзакции командой COMMIT.

Удаление строк из таблицы осуществляется с помощью оператора DELETE, который имеет следующий синтаксис:

```
DELETE [FROM] имя_таблицы [WHERE условие];
```

При отсутствии ключевого слова WHERE из таблицы удаляются все строки, но сама таблица остается.

Создание индексов. *Индекс* – это средство, обеспечивающее быстрый доступ к строкам таблицы на основе значений одного или нескольких столбцов. СУБД пользуется индексом так же, как читатели пользуются предметным указателем книги. В индексе хранятся значения данных и указатели на строки, где эти данные встречаются. Данные в индексе располагаются в отсортированном по убыванию или возрастанию порядке, чтобы СУБД могла быстро найти требуемое значение. Затем по указателю СУБД может быстро локализовать строку, содержащую искомое значение.

Наличие или отсутствие индекса совершенно незаметно для пользователя, обращающегося к таблице.

В Oracle можно создать, изменить или удалить индекс для одного или нескольких столбцов таблицы, используя следующий синтаксис:

CREATE [OR REPLACE] [UNIQUE | BITMAP] INDEX [*схема.*] *имя_индекса*
ON [*схема.*] *имя_таблицы* [*псевдоним*] (*столбец* | *выражение_для_столбца*
[ASC | DESC][, …]);

где UNIQUE означает, что значения столбцов, на которые ссылается индекс, должны быть уникальными;

BITMAP – изменение структуры индекса со сбалансированного дерева на структуру растровой карты.

Создание псевдонимов. *Псевдоним* – это назначаемое пользователем имя, которое заменяет полное имя некоторой таблицы с целью его упрощения.

В Oracle для создания псевдонимов используется инструкция CREATE SYNONYM.

CREATE [PUBLIC] SYNONYM *имя_синонима*
FOR [*схема.*] *имя_таблицы*[*@связь_БД*]

После создания псевдонима его можно использовать в запросах SQL как обычное имя таблицы. Применение псевдонимов смысл запроса не изменяет, так как и в этом случае необходимо иметь разрешение на доступ к таблицам других пользователей. Тем не менее псевдонимы упрощают инструкции SQL, и последние приобретают такой вид, как если бы вы обращались к своим собственным таблицам. Псевдоним можно удалить посредством инструкции DROP SYNONYM.

Создание последовательности. Использование последовательностей полезно для автоматической генерации уникальных первичных ключей для данных, а также для координирования ключей между различными строками или таблицами.

Без генератора последовательностей порядковые номера можно создавать лишь программным способом.

Общий синтаксис создания:

```
CREATE SEQUENCE [schema].sequence_name
[INCREMENT BY increment_num]
[START WITH start_num]
[MAXVALUE maximum_num | NOMAXVALUE]
[MINVALUE minimum_num | NOMINVALUE]
[CYCLE | NOCYCLE]
[CACHE cache_num | NOCACHE]
[ORDER | NOORDER];
```

где sequence_name – имя последовательности;

increment_num – шаг последовательности, по умолчанию это 1; абсолютное значение этого параметра должно быть меньше, чем разница между конечным и начальным значениями;

start_num – целочисленное значение, с которого начинается отсчет, по умолчанию это 1;

maximum_num – максимальное значение последовательности; значение maximum_num должно быть больше или равно значению start_num, и больше, чем значение minimum_num;

NOMAXVALUE – устанавливает максимальное значение равным 1027 для возрастающей последовательности или –1 для убывающей, используется по умолчанию;

minimum_num – минимальное значение последовательности; должно быть меньше либо равно start_num и меньше, чем maximum_num;

NOMINVALUE – определяет минимальное значение, равное 1 для возрастающей последовательности, и –1026 для убывающей, используется по умолчанию;

CYCLE – подразумевает, что последовательность начинает генерировать значения по кругу при достижении максимального или минимального значения. При обращении к последовательности, когда она достигла максимального значения, следующее сгенерированное значение будет минимальным значением последовательности. В ситуации с убывающей последовательностью при достижении минимального значения следующее сгенерированное будет максимальным;

NOCYCLE – указывает прекратить генерацию значений при достижении максимума или минимума последовательности, используется по умолчанию;

cache_num – количество значений, сохраняемых в памяти, по умолчанию это 20. Минимальное количество кэшированных значений – 2, максимальное значение высчитывается по формуле CEIL (maximum_num -minimum_num) /ABS(increment_num);

NOCACHE – отключает кэширование. Это не позволит базе данных выделить некоторое количество значений заблаговременно, что даст возможность избежать пробелов в последовательности, но увеличит затраты системных ресурсов. Пробелы могут возникнуть при остановке базы данных, когда кэшированные значения теряются. Если опции CACHE и NOCACHE не указываются, то по умолчанию кэшируется 20 значений;

ORDER – обеспечивает генерацию значений в порядке запросов, как правило, используется в среде Real Application Clusters;

NOORDER – не дает гарантий генерации значений в порядке запросов, используется по умолчанию.

Для работы с последовательностями генерируемых значений, применяемых в качестве уникальных ключей, используются псевдостолбцы:

– имя_последовательности.CURRVAL – возвращает текущее значение из указанной последовательности генерируемых значений;

– имя_последовательности.NEXTVAL – возвращает следующее значение из указанной последовательности генерируемых значений.

Значения CURRVAL и NEXTVAL используются в следующих местах:

– в списке SELECT предложения SELECT;

– в фразе VALUES предложения INSERT;

– в фразе SET предложения UPDATE.

Нельзя использовать значения CURRVAL и NEXTVAL в следующих местах:

– в подзапросе;

– в предложении SELECT с оператором DISTINCT;

– в предложении SELECT с фразой GROUP BY или ORDER BY;

– в предложении SELECT, объединенном с другим предложением SELECT оператором множеств UNION;

– в фразе WHERE предложения SELECT;

– в умалчиваемом (DEFAULT) значении столбца в предложении CREATE TABLE или ALTER TABLE;

– в условии ограничения CHECK.

Работа со словарем данных. *Словарь данных* – это набор служебных таблиц Oracle, который создается при генерации базы данных, он обновляется и обслуживается сервером. Как правило, в словаре содержится следующая информация: имена пользователей сервера Oracle; уровни привилегий пользователей; имена объектов базы данных; табличные ограничения целостности; учетные данные; параметры размещения объектов в физической памяти.

Получить информацию об объектах базы данных можно с помощью одного из следующих представлений:

– DICT[IONARY];

– USER_TABLES;

– USER_OBJECTS;

– USER_CONSTRAINTS;

– USER_CONS_COLUMNS.

Кроме этого, существует аналогичный набор представлений с префиксами ALL_ и DBA_.

Вывод списка всех представлений словаря данных, доступных пользователю, осуществляется командой

```
SELECT *FROM DICTIONARY;
```

Вывод структуры представления, например, USER_OBJECTS, в SQL*Plus выполняется командой

```
DESCRIBE user_objects
```

Вывод имен всех таблиц пользователя возможен командой

```
SELECT object_name FROM user_objects WHERE object_type ='TABLE’;
```

Просмотр определений и имен всех ограничений осуществляется из таблицы USER_CONSTRAINTS, например, проверка ограничений для таблицы EMP выглядит следующим образом:

```
SELECT constraint_name, constraint_type,
search_condition, r_constraint_name
FROM user_constraints WHERE table_name = 'EMPLOYEES';
```

Особенно полезен для ограничений, использующих системные имена, просмотр столбцов, на которые наложены ограничения, с помощью представления USER_CONS_COLUMNS:

```
SELECT constraint_name, column_name FROM user_cons_columns
WHERE table_name = 'EMPLOYEES';
```

Задание к лабораторной работе

1. Создать учебную базу данных «Предприятие по аренде недвижимости».

Для выполнения лабораторной работы предлагается учебная БД малого предприятия по аренде недвижимости, спроектированная по образцу, приведенному в [1].

БД содержит шесть таблиц: отделения (Branch), сотрудники (Staff), объекты недвижимости (Property_for_rent), арендаторы (Renter), собственники (Owner), осмотры (Viewing).

Таблица Branch предназначена для сохранения информации об отделениях (офисах) предприятия и имеет атрибуты, представленные на рис. 2.

bno	street	city	tel_no

Рис. 2. Таблица Branch

Здесь bno является первичным ключом и в соответствии с правилом целостности сущности не может принимать неопределенные значения; поле street – строковое, хранит наименование улицы и номера дома; поле city – это название города, где расположен офис, может быть ограничено перечнем допустимых городов; поле tel_no – номер телефона офиса, который должен быть уникальным.

Таблица Staff предназначена для хранения информации о сотрудниках и включает атрибуты, представленные на рис. 3.

sno	fname	lname	address	tel_no	position	sex	dob	salary	bno

Рис. 3. Таблица Staff

В данной таблице sno – первичный ключ, предназначенный для уникальной идентификации записей о сотрудниках; fname, lname – фамилия и имя сотрудника; address, tel_no – место жительства (включает город, улицу, дом) и телефон (уникальный); position – строковый атрибут, который определяет занимаемую должность; sex – пол сотрудника, который может принимать только два значения «male» или «female»; dob – атрибут типа даты с данными о днях рождения сотрудников; salary – числовой атрибут, содержащий зарплату сотрудников; bno – внешний ключ для связи с таблицей Branch.

Таблица Property_for_rent с информацией об объектах недвижимости, предлагаемых в аренду, имеет атрибуты, представленные на рис. 4.

pno	street	city	type	rooms	rent	ono	sno	bno

Рис. 4. Таблица Property_for_rent

Здесь pno – первичный ключ; street – адрес; city– город; type – строковый атрибут с информацией о типе предлагаемого объекта недвижимости, может принимать либо значение «h» (house), либо «f» (flat); поле rooms – количество комнат; rent – числовой атрибут, который имеет смысл рентной стоимости объекта; поля ono, sno, bno – внешние ключи таблицы для связи с таблицами Owner, Staff, Branch соответственно.

Таблица Renter содержит информацию о потенциальных арендаторах и включает атрибуты, представленные на рис. 5.

rno	fname	lname	address	tel_no	pref_type	max_rent	bno

Рис. 5. Таблица Renter

Здесь rno – первичный ключ; fname, lname, address, tel_no – соответственно фамилия, имя, адрес и телефон арендатора; pref_type – строковый атрибут, определяющий предпочтительный для клиента тип объекта аренды и ограниченный значениями «h» и «f»; max_rent – числовой атрибут, имеющий смысл максимальной рентной стоимости объекта с точки зрения арендатора; bno – внешний ключ для связи с таблицей Branch.

Таблица Owner определяет владельцев объектов недвижимости, которые сдаются в аренду, и включает атрибуты, представленные на рис. 6.

ono	fname	lname	address	tel_no

Рис. 6. Таблица Owner

В данной таблице ono является первичным ключом таблицы; fname, lname, address, tel_no – соответственно фамилия, имя, адрес и телефон владельца.

Таблица Viewing содержит результат осмотра арендаторами предполагаемых объектов аренды и включает атрибуты, представленные на рис. 7.

rno	pno	date_o	comment_o

Рис. 7. Таблица Viewing

Особенность данной таблицы – наличие составного первичного ключа, состоящего из атрибутов rno и pno, каждый из которых в отдельности является внешним ключом для связи с таблицами Renter (кто из потенциальных арендаторов производил осмотр) и Property_for_rent (какой из объектов осматривался). Атрибут date_o имеет тип даты и определяет дату осмотра, а comment_o – необязательный и самый «длинный» строковый атрибут, предназначенный для хранения сделанных потенциальным арендатором комментариев.

2. Заполнить таблицы произвольными данными, при этом выполнить следующие действия:

– для заполнения объектов недвижимости создать и использовать синоним – objects;

– создать для таблицы сотрудников последовательность Staff_seq, которая будет начинаться с 10 и увеличиваться на 5;

– в таблицу Owner перенести данные из таблицы Staff.

3. Создать собственную базу данных по полученному варианту (варианты заданий приведены в прил. 2), при этом выполнить следующие действия:

– создать таблицы, указав ограничение NOT NULL, где необходимо;

– придумать и создать ограничения (CHECK, UNIQUE);

– создать индексы, синонимы, последовательности, обосновать их необходимость;

– внести данные в таблицы, где необходимо в качестве первичных ключей использовать псевдостолбцы созданных последовательностей.

Контрольные вопросы

1. Что такое реляционная таблица и из каких элементов она состоит?
2. Какие типы данных применяются для определения столбцов таблиц?
3. Какие столбцы называют псевдостолбцами и почему?
4. Какие типы ограничений используются при создании таблиц?
5. Что понимают под бизнес-логикой?
6. Какие правила можно использовать для обеспечения ссылочной целостности при создании связи между таблицами БД?
7. Объясните порядок определения столбцов и ограничений при создании таблиц в инструкции CREATE TABLE.
8. В чем разница между значениями 0, пустой строкой и NULL?
9. В чем разница между первичным ключом и столбцом с уникальными значениями?
10. Для чего и как используется команда DESCRIBE?
11. Как можно создать копию существующей таблицы?

12. Что такое индексы и для чего они используются в БД?

13. Какие особенности работы индексов относят к их недостаткам?

14. Какие изменения можно вносить в определение таблицы инструкцией ALTER TABLE?

15. Для чего может применяться синоним таблицы, в чем его преимущество по сравнению с псевдонимами?

16. Как создать последовательность и для чего она необходима?

17. Перечислите основные параметры, которые можно указать при создании последовательности.

18. Что такое словарь данных, какие сведения в нем хранятся?

ЛАБОРАТОРНАЯ РАБОТА №2

СОЗДАНИЕ ЗАПРОСОВ

Цель работы – изучить синтаксис и назначение инструкции SELECT, научиться создавать условные, многотабличные, итоговые, параметрические запросы, запросы на объединение и вложенные запросы.

Теоретические сведения

Условные запросы. Инструкция SELECT извлекает информацию из базы данных и возвращает ее в виде таблицы результатов запроса. Данная инструкция состоит из шести основных предложений: SELECT, FROM, WHERE, GROUP BY, HAVING, ORDER BY.

Предложения SELECT и FROM являются обязательными, остальные четыре включаются в запрос при необходимости.

В предложении SELECT указывается список столбцов, которые должны быть возвращены инструкцией.

В предложении FROM обычно указывается список таблиц или представлений, которые содержат элементы данных, извлекаемые запросом. Например, следующий запрос извлекает из таблицы Staff три столбца, содержащих имя, фамилию и занимаемую должность каждого сотрудника:

```
SELECT fname, lname, position
FROM Staff;
```

Помимо этого, в предложении SELECT могут содержаться вычисляемые столбцы, например, выдать строки сотрудников с указанием зарплаты с 10%-й надбавкой:

```
SELECT fname, lname, position, (salary + 0.1*salary) AS percent
FROM Staff;
```

В данном запросе percent – это псевдоним столбца, который определяет название столбца в результирующей таблице запроса. Ключевое слово AS является необязательным, тогда псевдоним указывается через пробел.

Предложение WHERE показывает, что в результаты запроса следует включать только те строки, которые соответствуют условию отбора.

В SQL обычно используются пять основных условий отбора (в стандарте ANSI/ISO они называются предикатами):

- сравнение;
- проверка на принадлежность диапазону;
- проверка на членство во множестве;

– проверка на соответствие шаблону;
– проверка на равенство значению NULL.

Приведем примеры использования различных условий отбора в предложении WHERE.

Пример на простое сравнение: найти служащих, родившихся до 1988 года.

```
SELECT fname, lname
FROM Staff
WHERE dob < TO_DATE('01.01.1988', 'dd.mm.yyyy');
```

Формат записи данных типа дата/время зависит от серверных настроек и при необходимости его можно уточнить запросом SELECT sysdate FROM dual или использовать функции TO_CHAR или TO_DATE, как в примере.

Пример на принадлежность диапазону: найти служащих, родившихся в интервале времени с 1 октября 1963 года по 31 декабря 1971 года.

```
SELECT fname, lname
FROM Staff
WHERE dob BETWEEN TO_DATE('01.10.73', 'dd.mm.yyyy') AND
TO_DATE('31.12.91', 'dd.mm.yyyy');
```

Здесь следует отметить, что проверка на принадлежность диапазону не расширяет возможностей SQL, поскольку ее можно выразить в виде двух сравнений, т. е. выражение A BETWEEN B AND C эквивалентно (A>=B) AND (A<=C).

Пример на членство во множестве: вывести информацию об офисах, расположенных в Минске, Витебске и Бресте.

```
SELECT address, tel_no
FROM Branch
WHERE city IN ('Минск', 'Витебск', 'Брест');
```

Проверка IN также не добавляет новых возможностей, так как условие X IN (A, B, C) полностью эквивалентно условию (X=A) OR (X=B) OR (X=C).

Пример на соответствие шаблону: вывести информацию о всех сотрудниках, фамилии которых начинаются на букву К.

```
SELECT lname, address, tel_no
FROM Staff
WHERE lname LIKE 'К%';
```

В Oracle символ '%' замещает произвольную последовательность символов, а '_' − одиночный символ. Строки-шаблоны так же, как и обыкновенные

строки-константы, заключаются в парные одинарные кавычки. Для построения более сложных конструкций поиска используются регулярные выражения.

При работе с полями, хранящими даты, удобно применять функции преобразования типов и функции форматирования дат по определенному шаблону. В общем случае функция TO_CHAR имеет формат

TO_CHAR (поле, 'шаблон')

где шаблон – это строка, содержащая до 40 параметров.

В табл. 1 приведены наиболее часто используемые параметры для работы с полями типа дата/время.

Таблица 1

Основные параметры поля дата/время

Обозначение	Значение
MM	Номер месяца (1–12)
MON	Сокращенное название месяца
MONTH	Полное название месяца
DD	День месяца (1–31)
DY	Сокращенное название дня
DAY	Полное название дня (не более 9 символов)
DY	Сокращенное название дня недели
WW	Неделя года (1–52)
W	Неделя месяца
Q	Квартал(1–4)
YYYY	Год, в виде четырех цифр
YYY|YY|Y	3, 2 или 1 последние цифры года
YEAR	Произношение года
AM (или PM)	Индикатор меридиана (до или после полудня)
HH	Час дня (1–12)
HH24	Час суток (0–23)
MI	Минуты (0–59)
SS	Секунды (0–59)

Также при работе с датами используется функция EXTRACT для извлечения года, месяца или дня. Например:

```
SELECT EXTRACT (YEAR FROM sysdate) FROM dual;
SELECT EXTRACT (MONTH FROM sysdate) FROM dual;
SELECT EXTRACT (DAY FROM sysdate) FROM dual;
```

Можно изменить заданный по умолчанию формат DATE в Oracle с 'DD-MON-YY' на какой-либо другой, используя следующую команду в SQL*Plus:

```
ALTER SESSION SET NLS_DATE_FORMAT = 'новый_формат';
```

Изменение действует только для текущего сеанса SQL*Plus.

Параметрические запросы. В SQL*Plus можно выделять именованную область памяти для хранения некоторой информации. Такая область используется внутри программ PL/SQL и SQL-операторов, однако находится вне программных блоков, поэтому можно по очереди выделять ее разным программным блокам и после выполнения каждого из них выводить ее содержимое. Эта именованная область памяти называется переменной привязки (подстановки) – bind variable. При использовании переменной привязки в запросах ее имени предшествует знак «&» (в iSQLPlus OracleXE11g используется символ двоеточия «:»).

```
SELECT * FROM &table_name WHERE bno=&v_bno;
```

Также практикуется использование двойного знака «&&» для переменной подстановки. Значение введенной после сдвоенного знака «&» переменной подстановки запоминается в SQL*Plus, и в следующий раз при обращении к этой переменной подстановки ее значение вводится автоматически.

Многотабличные запросы с внутренним соединением таблиц. Если необходимо получить информацию более чем из одной таблицы, то можно либо применить подзапрос, либо выполнить соединение таблиц. Для выполнения соединения достаточно в предложении FROM указать имена соединяемых таблиц, а в предложении WHERE – условие соответствия столбцов соединения. Например, составить список всех сотрудников, работающих в минских отделениях.

```
SELECT fname, lname, position, S.tel_no
FROM Branch B, Staff S
WHERE B.bno=S.bno and city = 'Минск';
```

При выполнении запроса СУБД сначала просматривает столбец city с целью фильтрации строк со значениями, отличными от значения «Минск», далее для отфильтрованных строк таблицы находятся значения столбца bno, определяющие минские отделения. После этого просматривается таблица Staff и выявляются строки со значениями в столбце bno, соответствующими идентификатору первого минского офиса. В найденных строках оставляются значения столбцов, указанных после ключевого слова SELECT. Далее эта же таблица просматривается для выявления строк, соответствующих второму минскому офису, и опять заново до последнего найденного соответствия. В итоге формируется таблица результатов с запрошенной информацией. Кроме того, необходимо помнить, что в многотабличных запросах велик риск появления дубликатов строк, избавиться от которых помогает предикат DISTINCT. В рассмотренном примере DISTINCT не используется, так как предполагается, что сотрудники не могут работать одновременно в нескольких отделениях и их телефоны уникальны, иначе предикат был бы необходим.

Отметим некоторые особенности многотабличных запросов. Как видно из примера, в многотабличном запросе часто используются полные имена столбцов, при этом в предложении FROM через пробел могут указываться псевдонимы таблиц, чтобы упростить обращение к столбцам по полному имени, а также обеспечить однозначность ссылок на столбцы. Кроме этого, особый смысл может иметь выбор всех столбцов (SELECT *), например, в Oracle поддерживается следующий синтаксис:

```
SELECT s.*, city
FROM Staff s, Branch b
WHERE b.bno=s.bno;
```

Помимо соединения двух таблиц SQL допускает также соединение трех и более таблиц. Ограничений по количеству соединяемых таблиц ни стандарт, ни разработчики СУБД не предусматривают, однако следует иметь в виду, что при их увеличении в запросе снижается его «читабельность» и скорость выполнения в силу значительного увеличения затрат ресурсов и машинного времени при обработке. Для написания запросов с внутренним соединением можно использовать и стандартный синтаксис операторов внутреннего соединения: JOIN ON, JOIN USING, NATURAL JOIN.

Кроме внутренних соединений используются внешние, например, можно вывести список всех отделений и количество сотрудников в них, включая отделения, где пока никто на работает.

```
SELECT b.bno, count(s.bno)
FROM Branch b, Staff s
WHERE b.bno=s.bno(+)
GROUP BY b.bno
ORDER BY 2;
```

Для запросов с внешним соединением также можно использовать стандартный синтаксис операторов внешнего соединения: FULL JOIN, LEFT JOIN, RIGHT JOIN.

Итоговые запросы. Результирующую таблицу итогового запроса можно рассматривать как некий отчет. Для получения подобных отчетов в запросе на получение итоговой информации требуется указывать предложение GROUP BY и возможное HAVING для отбора групп. Запрос, включающий в себя предложение GROUP BY, называется запросом с группировкой, поскольку он объединяет строки исходных таблиц в группы и для каждой группы строк генерирует одну строку в таблице результатов запроса.

Ограничением при выполнении итоговых запросов является то, что здесь в предложении SELECT могут употребляться лишь столбцы группировки (т. е. те, которые указываются в предложении GROUP BY), строковые константы и статистические функции. Таких функций в SQL пять:

− SUM (имя_столбца) – для вычисления суммы всех значений столбца-аргумента;

− AVG (имя_столбца) – для вычисления среднего значения столбца;

− MIN (имя_столбца) – определяет минимальное значение столбца;

− MAX (имя_столбца) – определяет максимальное значение столбца;

− COUNT (имя_столбца) – подсчитывает число всех определенных значений столбца;

− COUNT (*) – подсчитывает число строк в запросе.

Определить, сколько в среднем получают сотрудники в зависимости от занимаемой ими должности и вывести отсортированный список:

```
SELECT position, AVG(salary)
FROM Staff
GROUP BY position
ORDER BY 2;
```

Условие отбора групп (предложение HAVING). Точно так же, как предложение WHERE используется для отбора отдельных строк, участвующих в запросе, предложение HAVING можно применить для отбора групп строк. Его формат соответствует формату предложения WHERE, т. е. состоит из ключевого слова HAVING, за которым следует условие отбора. Рассмотрим пример: подсчитать количество сотрудников, работающих в каждом из офисов, исключив офисы, в которых работает менее двух человек:

```
SELECT bno, COUNT(sno)
FROM Staff
GROUP BY bno
HAVING COUNT(sno) > 2;
```

Ограничения на условия отбора групп. Предложение HAVING используется для того, чтобы включать и исключать группы строк из результатов запроса, поэтому используемое в нем условие отбора применяется не к отдельным строкам, а к группе в целом, т. е. используется только совместно с GROUP BY. Это значит, что в условие отбора может входить:

− константа;

− статистическая функция, возвращающая одно значение для всех строк, входящих в группу;

− столбец группировки, который по определению имеет одно и то же значение во всех строках группы;

− выражение, включающее в себя перечисленные выше элементы.

Подзапросы. Подзапросом или подчиненным запросом называется запрос, содержащийся в предложении WHERE или HAVING другой инструкции SQL.

Механизм подчиненных запросов позволяет использовать результаты одного запроса в качестве составной части другого.

По типу возвращаемых значений существует три типа подзапросов:

1) скалярный – возвращает единственное значение;

2) строковый – возвращает значения нескольких столбцов таблицы, но в одной строке;

3) табличный – возвращает значения одного и более столбцов таблицы, размещенные более чем в одной строке.

В SQL используются следующие условия отбора в подчиненном запросе:

1. Сравнение с результатом подчиненного запроса. Значение выражения сравнивается с одним значением, которое возвращается подчиненным запросом.

2. Проверка на принадлежность результатам подчиненного запроса. Значение выражения проверяется на равенство одному из множества значений, которые возвращаются подчиненным запросом. Эта проверка напоминает простую проверку на членство во множестве (IN).

3. Проверка на существование. Проверяется наличие строк в таблице результатов подчиненного запроса (EXISTS/NOT EXISTS).

4. Многократное сравнение. Значение выражения сравнивается с каждым из множества значений, которые возвращаются подчиненным запросом (ANY/ALL).

При использовании подзапросов необходимо учитывать ряд особенностей:

− таблица результатов подчиненного запроса обычно состоит из одного столбца;

− в подчиненный запрос не может включаться предложение ORDER BY;

− подчиненный запрос не может быть запросом на объединение нескольких различных инструкций SELECT (т. е. нельзя использовать UNION);

− имена столбцов в подчиненном запросе могут являться ссылками на столбцы таблиц главного запроса.

Рассмотрим несколько примеров. Рассмотренный ранее запрос (список всех сотрудников, работающих в минских отделениях) можно выполнить с помощью подзапроса:

```
SELECT fname, lname, position, tel_no
FROM Staff
WHERE bno IN (SELECT bno FROM Branch WHERE city= 'Минск');
```

С помощью проверки на существование из таблицы Staff можно вывести сотрудников, которые не обслуживают ни один объект недвижимости:

```
SELECT fname, lname FROM Staff
WHERE NOT EXISTS (SELECT * FROM Property_for_rent
                  WHERE Property_for_rent.bno = Staff.bno);
```

Здесь в подзапросе используется ссылка на столбец Staff.bno, который находится во внешнем запросе, поэтому такой прием называется внешней ссыл-

кой в подзапросе, а такие запросы коррелированными, так как они выполняются по особому алгоритму: для каждой строки внешнего запроса выполняется подзапрос, в то время как во всех остальных случаях подзапрос выполняется один раз.

С помощью многократного сравнения можно, например, определить отделения, количество сотрудников которых превышает количество сотрудников любого отделения Минска:

```
SELECT s.bno, count(s.bno)
FROM Staff s, Branch b
WHERE s.bno = b.bno AND city!= 'Минск'
GROUP BY s.bno
HAVING count(s.bno)>= ALL (SELECT count(s1.bno)
      FROM staff s1, branch b1
      WHERE s1.bno = b1.bno AND city= 'Минск'
      GROUP BY s1.bno);
```

Обновление строк таблицы реализуется с помощью оператора UPDATE, форма записи которого приведена ниже. При отсутствии условия модифицируются все строки таблицы для указанного списка столбцов.

```
UPDATE имя_таблицы
SET {(имя_столбца1 [, имя_столбца2] ...) = (подзапрос) |
имя_столбца1=значение1, имя_столбца2={значение2 | (подзапрос)}}
[WHERE условие];
```

Задание к лабораторной работе

1. Выполнить запросы к БД «Предприятие по аренде недвижимости».

Вариант 1:

– Вывести заглавными буквами список фамилий и имен сотрудников с зарплатой от 200 до 300 дол. США.

– Получить список фамилий сотрудников и их телефонов, если они работают в офисах Бреста или Минска.

– Определить и вывести суммарную и среднюю заработную плату сотрудников в зависимости от занимаемой ими должности. Подписать вычисляемые поля как «Суммарная» и «Средняя».

– Вывести должности, по которым суммарная заработная плата больше общей средней зарплаты по отделению.

Вариант 2:

– Вывести в одном поле адреса и телефоны офисов, расположенных в Минске и Гродно.

– Вывести данные о сотрудниках, которые предлагают для аренды 3-комнатные квартиры.

– Вывести итоговый отчет о средней и суммарной заработной плате сотрудников в зависимости от их половой принадлежности. Подписать вычисляемые поля как «Суммарная» и «Средняя».

– Вывести информацию об отделениях, где работает больше женщин, чем мужчин, учитывая, что в отделениях могут быть сотрудники одного пола.

Вариант 3:

– Вывести в одном поле адреса и стоимости всех 3-комнатных квартир, предлагаемых в аренду.

– Получить список арендаторов, осматривавших объекты аренды в 2019 году.

– Определить минимальную и максимальную заработную плату сотрудников в каждом отделении. Подписать вычисляемые поля как «Минимальная» и «Максимальная».

– Вывести отделения, в которых суммарная заработная плата больше средней заработной платы по всему агентству.

Вариант 4:

– Вывести в одном поле фамилии и номера телефонов всех директоров.

– Составить список владельцев всех 3-комнатных квартир.

– Подсчитать количество сотрудников в каждом из отделений. Подписать вычисляемое поле как «Количество».

– Вывести все сведения об отделениях, а также количество сотрудников в них и количество обратившихся арендаторов при условии, что эти количества больше одного.

Вариант 5:

– Вывести в одном поле ФИО и адреса сотрудников, родившихся до мая 1980 года.

– Подсчитать и вывести, сколько сотрудников работает в отделениях в Бресте. Подписать вычисляемое поле как «Количество».

– Подсчитать количество арендаторов, осмотревших 3- и 4-комнатные объекты недвижимости. Вывести в запросе количество комнат для каждого типа объекта и количество арендаторов.

– Вывести информацию о владельцах и количестве объектов у каждого при условии, что они сдают более двух квартир в разных отделениях.

Вариант 6:

– Определить адреса всех квартир с рентной стоимостью не более 300 дол. США.

– Подсчитать и вывести количество менеджеров, работающих во всех отделениях Минска. Подписать вычисляемое поле как «Количество».

– Определить и вывести количество объектов, сдаваемых в аренду, в зависимости от их типа. Подписать вычисляемое поле как «Количество».

– Вывести информацию о сотрудниках, которые работают с бóльшим количеством домов, чем квартир, учесть, что можно работать только с одним типом объектов.

Вариант 7:

– Вывести в одном поле фамилии и домашние телефоны всех потенциальных арендаторов, желающих арендовать дома.

– Вывести телефоны владельцев, дома или квартиры которых осматривались в сентябре 2019 года.

– Определить квартиру и дом с минимальной рентной стоимостью в каждом отделении. Подписать вычисляемое поле как «Дешевый дом/квартира».

– Подсчитать среднюю заработную плату сотрудников каждого отделения и количество обслуживаемых в них объектов.

Вариант 8:

– Вывести заглавными буквами список всех женщин-менеджеров.

– Определить и вывести сотрудников с максимальной зарплатой в отделениях Гродно.

– Определить и вывести количество осмотров объектов по дням недели. Подписать вычисляемое поле как «Количество».

– Вывести информацию об отделении, где предлагаются в аренду 2-комнатные квартиры с максимальной средней стоимостью во всем агентстве.

Вариант 9:

– Определить, сколькими арендаторами и сколько объектов было осмотрено в течение последнего года.

– Создать список сотрудников, предлагающих объекты недвижимости в Минске.

– Определить суммарную рентную стоимость объектов в Минске и объектов в Гродно. Вывести город и сумму.

– Вывести информацию о владельцах неминских квартир, количество квартир у которых превышает количество квартир любого минского владельца.

Вариант 10:

– Вывести фамилии и телефоны арендаторов, в которых встречаются цифры 5 или 7. При выводе поля должны быть фиксированной длины (дополнить, например, точками), поле «Фамилия» выравнивается по левому краю, поле «Телефон» – по правому.

– Определить и вывести количество собственников, объекты которых уже осмотрели и оставили комментарий.

– Вывести информацию об отделениях, в которых работают более трех сотрудников.

– Вывести информацию о сотрудниках того отделения, в котором не обслуживается ни один объект недвижимости.

2. Обновить одной командой информацию о максимальной рентной стоимости объектов, уменьшив стоимость квартир на 5 %, а стоимость домов увеличив на 7 %.

3. Написать запросы по индивидуальной базе данных: условный запрос, итоговый запрос, параметрический запрос, запрос на объединение, запрос с ис-

пользованием условия по полю с типом дата. Варианты заданий приведены в прил. 3. В отчет обязательно включается формулировка запроса, код запроса на SQL и результат его выполнения.

4. Для своего варианта самостоятельно придумать задание и реализовать следующие типы запросов:

– с внутренним соединением таблиц, используя стандартный синтаксис SQL (JOIN…ON, JOIN…USING или NATURAL JOIN), который не применялся в предыдущих запросах;

– с внешним соединением таблиц, используя FULL JOIN, LEFT JOIN или RIGHT JOIN, при этом обязательным является наличие в БД данных, которые будут выводится именно с выбранным оператором внешнего соединения;

– с использованием предиката IN с подзапросом;

– с использованием предиката ANY/ALL с подзапросом;

– с использованием предиката EXISTS/NOT EXISTS с подзапросом.

Контрольные вопросы

1. Какова общая структура запроса на извлечение информации?
2. Какие предикаты могут применяться в предложении WHERE?
3. Как можно устранить дубликаты строк в запросах?
4. Что такое псевдонимы и для чего они используются?
5. Поясните принцип соединения таблиц по условию равенства столбцов.
6. Что такое внешние соединения?
7. Когда выполняется операция декартова произведения двух таблиц?
8. Перечислите особенности синтаксиса итоговых запросов.
9. Для чего используется предложение HAVING?
10. Для чего и где используются подзапросы?
11. Какие предикаты могут использоваться с подзапросами?
12. Как обновить данные одновременно в двух столбцах таблицы?

ЛАБОРАТОРНАЯ РАБОТА №3

СОЗДАНИЕ И ИСПОЛЬЗОВАНИЕ ПРЕДСТАВЛЕНИЙ

Цель работы – научиться создавать и работать с однотабличными, многотабличными, индивидуальными и агрегированными представлениями.

Теоретические сведения

Представление (views) – это хранимый в памяти SQL-запрос, который автоматически выполняется при работе с представлением. Другими словами, представление – это виртуальная таблица, которая сама по себе не существует, но для пользователя выглядит таким образом, как будто она существует. Представление определяется перечнем тех столбцов таблиц и признаками тех их строк, которые хотелось бы в ней увидеть. Представление не поддерживается его собственными физическими хранимыми данными. Вместо этого в каталоге таблиц хранится определение, оговаривающее, из каких столбцов и строк других таблиц оно должно быть сформировано при реализации SQL-предложения на получение данных из представления или на модификацию таких данных. В каталоге БД представление сохраняется в виде того запроса, который его определяет. При трансляции запроса, содержащего обращение к представлению, SQL-машина подставляет вместо него обращение к базовой таблице/таблицам и добавляет те условия, которые определяют представление.

Одним из главных достоинств представлений является то, что они значительно упрощают работу конечного пользователя с данными в БД, так как в запуске одного и того же запроса больше нет необходимости. Достаточно просто создать одно представление и потом уже обращаться к нему с помощью простых запросов. В дальнейшем представления можно объединять с другими таблицами или другими представлениями.

Работая с представлениями, нужно помнить о следующих их свойствах:

– представления добавляют уровень защиты данных (например, можно создать представление для таблицы, где пользователю, выполняющему SELECT над представлением, видны только сведения о заработной плате);

– представления могут скрывать сложность данных, комбинируя нужную информацию из нескольких таблиц;

– представления могут скрывать настоящие имена столбцов, порой трудные для понимания, и показывать более простые имена.

Для создания представления в стандарте SQL используется команда CREATE VIEW, которая имеет следующий формат:

CREATE [OR REPLACE] VIEW [схема.]имя_представления[список имен столбцов]AS SELECT список_полей FROM имя_таблицы;

Как видно, в основе представления лежит SQL-выражение отбора данных, т. е. запрос. Столбцы, указанные в команде SELECT, определяют столбцы (их количество и названия) в созданном представлении.

Список имен столбцов должен быть обязательно определен лишь в следующих случаях:

– когда хотя бы один из столбцов подзапроса не имеет имени (создается с помощью выражения, SQL-функции или константы);

– когда два или более столбцов запроса имеют одно и то же имя; если же список отсутствует, то представление наследует имена столбцов из запроса.

При создании представлений инструкция SELECT не должна содержать:

– конструкцию ORDER BY;

– конструкцию FOR UPDATE, так как она блокирует выбранные строки и не позволяет другим пользователям их обновлять или блокировать, пока включающая эту конструкцию транзакция не будет завершена;

В список столбцов представления может быть включено не более 254 столбцов или выражений.

Каждый включенный в представление столбец всегда доступен для вывода и иногда для изменения (если представление является обновляемым). Для работы с данными, отраженными в представлении, используются те же SQL-команды, что и для работы с таблицами (SELECT, INSERT, UPDATE, DELETE). Однако следует учитывать, что создание или удаление представления не изменяет таблицу, но если в представлении выполняется вставка или удаление строк, то эти операции отражаются в ней.

Если какой-либо столбец не был включен в представление при его создании, то при попытке вставки новой строки в представление Oracle всегда будет вставлять данные в исходные таблицы и присваивать отсутствующим столбцам значение NULL или значение по умолчанию, если такое было установлено для столбца при создании таблицы. Иногда значение NULL может привести к ошибкам (например, если этот столбец является первичным ключом или если на него было наложено ограничение NOT NULL), поэтому следует учитывать это обстоятельство при создании таблицы и представления.

Во всех выражениях выборки представление может быть использовано вместо таблицы без всяких ограничений, т. е. после определения представления к нему можно обращаться с помощью инструкции SELECT как к обычной таблице:

```
SELECT * FROM view_name;
```

Для удаления представления используется команда DROP со следующим форматом:

```
DROP VIEW имя_представления;
```

Сложнее обстоит дело с модификацией представления. Представление обладает свойствами таблицы: в него можно вставлять данные, удалять или модифицировать их. Но тем не менее представление жестко связано с таблицами,

над которыми оно было создано. При вставке новых картежей в представление (а по сути вставки их в таблицу базы данных) анализируются ограничения этой таблицы.

Для простых представлений операции добавления, изменения и удаления можно преобразовать в эквивалентные операции по отношению к исходным таблицам.

По стандарту SQL представления будут обновляемыми (изменяемыми), если выполняются следующие условия:

– должен присутствовать ключ и все столбцы, которые определены как NOT NULL;

– должен отсутствовать предикат DISTINCT, т. е. повторяющиеся строки не исключаются из результата запроса;

– в предложении FROM должна быть задана только одна таблица или представление;

– каждое имя в списке возвращаемых столбцов должно быть ссылкой на простой столбец, не должны содержаться выражения, вычисляемые столбцы и статистические функции, спецификация одного столбца не должна появляться более одного раза;

– в предложении WHERE не должен содержаться подчиненный запрос;

– в запросе не должны присутствовать предложения GROUP BY и HAVING.

Все эти требования стандарта слишком жесткие и не всегда соблюдаются, в том числе и в СУБД Oracle. Так, например, Oracle дает частичную возможность изменения соединенных таблиц.

Все представления можно разделить на однотабличные (основанные только на одной таблице), многотабличные, индивидуальные и агрегированные (сгруппированные).

Основное назначение однотабличных представлений – скрыть от пользователя некоторые столбцы. Эта процедура получила название «маскирование столбцов». Для создания маскирующего представления необходимо в списке столбцов команды SELECT указать только те, к которым пользователю разрешен доступ. Такие представления можно назвать также вертикальными.

Создать представление, показывающее информацию о ФИО и должности служащих:

```
CREATE OR REPLACE VIEW info AS
SELECT fname, lname, position
FROM Staff;
```

Это вертикальное однотабличное представление, т. е. представление, ограничивающее доступ к столбцам исходной таблицы. Это представление не может быть обновляемым, так как в перечень столбцов представления не включен как минимум ключевой столбец, без которого невозможно будет вставить

данные через представление в первоначальную таблицу, за исключением случаев, когда он заполняется с помощью последовательности.

Создать представление, показывающее информацию о служащих, работающих в отделении компании города Минска:

```
CREATE OR REPLACE VIEW Minsk AS
SELECT * FROM Staff
WHERE bno IN (SELECT bno FROM Branch
WHERE city = 'Minsk');
```

Это пример создания простого горизонтального представления, т. е. представления, которое позволяет видеть в исходной таблице Staff не все строки, а только те, которые удовлетворяют условию отбора запроса, лежащего в его основе. Кроме того, это представление будет обновляемым в Oracle, несмотря на то что нарушаются требования стандарта, и через него можно будет добавлять строки в исходную таблицу.

Конечно, на практике при создании представлений обычно требуется разделять таблицу и по вертикали, и по горизонтали – такие представления получили название смешанных.

В отличие от горизонтальных, вертикальных и смешанных представлений каждой строке сгруппированного представления не соответствует какая-то одна строка исходной таблицы. Сгруппированное представление не является просто фильтром исходной таблицы, скрывающим некоторые строки и столбцы. Оно отображает исходную таблицу в виде резюме, поэтому поддержка такой виртуальной таблицы требует от СУБД значительного объема вычислений и отслеживания проблем с обновлением.

Создать представление с информацией о средней заработной плате сотрудников по каждому отделению:

```
CREATE OR REPLACE VIEW average_salary AS
SELECT bno, AVG(salary) Srednya_zarplata
FROM Staff
GROUP BY bno;
```

Полная инструкция создания представления в Oracle имеет вид

```
CREATE [OR REPLACE] [FORCE|NOFORCE] VIEW view-name
AS sql-запрос [WITH CHECK OPTION [CONSTRAINT имя_ограничения]]
[WITH READ ONLY];
```

Фраза OR REPLACE – пересоздает представление, если оно уже существует. Можно использовать эту опцию для изменения определения представления без того, чтобы удалять его, создавать заново и вновь назначать все объектные привилегии, которые были назначены по данному представлению.

Фраза FORCE – создает представление независимо от того, существуют ли базовые таблицы этого представления, и от того, имеет ли владелец схемы, содержащей представление, привилегии по этим таблицам. Необходимо, чтобы оба названных условия были удовлетворены, прежде чем по данному представлению можно будет выдавать любые предложения SELECT, INSERT, UPDATE или DELETE. Перед обращением к приложению его следует также перекомпилировать командой:

ALTER VIEW имя_представления COMPILE;

Фраза NOFORCE (применяется по умолчанию) – создает представление только в том случае, если существуют базовые таблицы этого представления, а владелец схемы, содержащей представление, имеет привилегии по этим таблицам.

Фраза WITH CHECK OPTION – указывает, что вставки и обновления, которые будут осуществляться через этот запрос, должны давать в результате только такие строки, которые могут быть выбраны запросом этого же представления. Другими словами, если представление создается посредством запроса с предложением WHERE, то в представлении будут видны только строки, удовлетворяющие условию отбора. Остальные строки в исходной таблице присутствуют, но в представлении их нет.

Например, представление Minsk содержит только строки таблицы Staff с определенными значениями в столбце bno.

Это представление является обновляемым, следовательно, в него можно добавить информацию о новом служащем посредством инструкции INSERT:

```
INSERT INTO Minsk (sno, fname, lname, address, position, sex, dob, salary, bno)
VALUES ('s129', '…', '…', '…', 'менеджер', 'f', '01.01.81', 300, 1).
```

СУБД добавит новую строку в исходную таблицу Staff, она будет видна также в представлении Minsk. Также без опции WITH CHECK OPTION ничего не препятствует выполнению следующей инструкции:

```
INSERT INTO Minsk (sno, fname, lname, address, position, sex, dob, salary, bno)
VALUES ('s129', '…', '…', '…', 'менеджер', 'f', '01.01.81', 300, 2).
```

После этого в запросе SELECT * FROM Minsk добавленная строка будет отсутствовать. Тот факт, что в результате выполнения инструкции INSERT или UPDATE из представления исчезают строки, в лучшем случае вызывает замешательство.

При указании параметра WITH CHECK OPTION пользователь не сможет вводить, удалять и обновлять информацию таблицы, из которой он не имеет возможности считать информацию через простое представление (создаваемое из данных одной таблицы). Обновляемое представление, использующее несколько связанных таблиц, нельзя создавать с данным параметром, так как оп-

ция WITH CHECK OPTION не может гарантировать контроль, если существует подзапрос в запросе этого представления или любого представления, на котором базируется данное представление.

Фраза CONSTRAINT имя_ограничения – задает имя, которое присваивается ограничению CHECK OPTION. Если этот идентификатор опущен, то Oracle автоматически назначает этому ограничению уникальное имя.

```
CREATE OR REPLACE VIEW Minsk AS
SELECT * FROM Staff
WHERE bno IN (SELECT bno
FROM Branch
WHERE city='Minsk')
WITH CHECK OPTION CONSTRAINT check_Minsk;
```

Когда для представления установлен режим контроля, СУБД автоматически проверяет каждую операцию INSERT или UPDATE, выполняемую над представлением, чтобы удостовериться в том, что полученные в результате строки удовлетворяют условиям отбора в определении представления. Если добавляемая или обновляемая строка не удовлетворяет этим условиям, то выполнение инструкции INSERT или UPDATE завершается ошибкой; другими словами, операция не выполняется.

Фраза WITH READ ONLY – явно запрещает операции DML над любым представлением. Если команды обновления DML (INSERT, UPDATE, DELETE) можно применить к представлению, то говорят, что представление является обновляемым (updatable); в противном случае оно является только читаемым (read-only).

После того как представление создано, можно обратиться к таблице словаря данных USER_VIEWS для просмотра его определения, включающего команду SELECT.

На выдачу текста о представлении могут повлиять значения следующих параметров команды SET среды SQL*Plus: Maxdata; Arraysize; Long.

Текст команды SELECT хранится в поле типа Long, и для просмотра этого текста может понадобиться присвоить параметру ARRAYSIZE какое-либо небольшое значение (например, SET ARRAYSIZE = 1).

Задание к лабораторной работе

1. В учебной базе данных создать следующие представления:

1.1) с информацией об офисах в Бресте;
1.2) с информацией об объектах недвижимости минимальной стоимости;
1.3) с информацией о количестве сделанных осмотров с комментариями;

1.4) со сведениями об арендаторах, желающих арендовать 3-комнатные квартиры в тех же городах (поле city), где они проживают (поле address);

1.5) со сведениями об отделении с максимальным количеством работающих сотрудников;

1.6) с информацией о сотрудниках и объектах, которые они предлагают в аренду в текущем квартале;

1.7) с информацией о владельцах, чьи дома или квартиры осматривались потенциальными арендаторами более двух раз;

1.8) с информацией о собственниках с одинаковыми именами.

2. В индивидуальной БД создать:

– горизонтальное обновляемое представление;

– вертикальное или смешанное необновляемое представление, предназначенное для работы с основной задачей БД (в представлении должны содержаться сведения из главной таблицы, но вместо внешних ключей необходимо использовать связанные данные из родительской таблицы);

3. Проверить обновляемость горизонтального представления с фразой WITH CHECK OPTION при помощи инструкций UPDATE, DELETE или INSERT (привести пример правильной и неправильной инструкции).

4. Создать обновляемое представление для работы с одной из родительских таблиц индивидуальной БД и через него разрешить работу с данными только в рабочие дни (с понедельника по пятницу) и в рабочие часы (с 9:00 до 17:00).

Контрольные вопросы

1. Что такое представления и для чего они используются в базах данных?
2. Какие бывают виды представлений?
3. Какие DDL-инструкции применимы к представлениям?
4. Чем отличаются горизонтальные представления от вертикальных?
5. Для чего используется фраза WITH CHECK OPTION?
6. Что дает использование фразы FORCE?
7. Какие требования предъявляются стандартом к обновляемым представлениям?
8. Почему нельзя обновить представление, если в его основе лежит сгруппированный запрос?
9. Как запретить обновлять представления явно, даже если они и являются обновляемыми?
10. В чем заключаются основные преимущества использования представлений?

ЛАБОРАТОРНАЯ РАБОТА №4

ПРОЦЕДУРЫ, ФУНКЦИИ И ПАКЕТЫ PL/SQL

Цель работы – изучить основные возможности и программные конструкции языка PL/SQL. Приобрести навыки разработки хранимых процедур, функций, пакетов.

Теоретические сведения

Структура программ PL/SQL. Язык PL/SQL (Procedural Language / Structured Query Language) – это язык программирования, представляющий собой процедурное расширение Oracle – версии языка SQL. В первую очередь PL/SQL дает возможность использовать переменные, операторы управления ходом выполнения программы, массивы, курсоры и исключения.

Язык программирования PL/SQL разработан на базе языка третьего поколения Ada. Одним из общих свойств этих языков является их блочная структура. Из Ada заимствованы также обработка исключительных ситуаций, синтаксис объявления процедур и функций, модули (пакеты). Блок может содержать вложенные блоки, называемые иногда подблоками. Из блоков создаются процедуры и функции, из которых в свою очередь строятся приложения. Блок определяет область действия логически связанных объектов. В блоке группируются связанные между собой объявления и выполняются операторы [3].

Блоки PL/SQL имеют следующую структуру:

– раздел заголовка, который начинается ключевыми словами CREATE OR REPLACE и определяет имя блока, необходим только для именованных блоков и отсутствует для анонимных;

– раздел объявлений, который начинается ключевым словом DECLARE и содержит объявления переменных, констант, курсоров, типов и локальных программ, используемых в данном блоке. Может отсутствовать. Слово DECLARE используется только в триггерах и анонимных блоках, во всех остальных программных единицах объявления расположены между заголовком и BEGIN;

– раздел выполнения, который начинается ключевым словом BEGIN и содержит выполняемый код, включающий процедурные и SQL-операторы, заканчивается ключевым словом END. Это основной раздел блока, который также называют выполняемым, он должен содержать по крайней мере один выполняемый оператор. Не обязателен только в спецификациях пакетов и типов.

– раздел исключений, который начинается ключевым словом EXCEPTION, заканчивается END, содержит операторы обработки исключительных ситуаций (ошибок). Может отсутствовать.

Таким образом, блок может быть анонимным или именованным. Анонимные блоки – это неназванные PL/SQL-блоки, которые встроены в приложе-

ние или выдаются в интерактивном режиме. Анонимные блоки не хранятся в базе данных, они передаются в процессор PL/SQL для исполнения во время выполнения приложения. Структура анонимного блока приведена ниже:

```
[DECLARE]
BEGIN
Исполняемые операторы
[EXCEPTION]
END;
/
```

Пример использования:

```
DECLARE
X INTEGER := 8;
Y INTEGER :=5;
BEGIN
X: = X+Y; Y: = Y-3;
DBMS_OUTPUT.PUT_LINE('X ='||X||' Y='||Y);
END;
/
```

Отметим, что любой блок в SQL*Plus завершается символом наклонной черты «/», а оператор присваивания обозначается символом «:=».

Также следует не забывать о необходимости включать серверный вывод результатов работы блоков в SQL*Plus при использовании функций пакета DBMS_OUTPUT командой

```
SET SERVEROUTPUT ON
```

Именованный блок может быть разделен на три подвида:

1. Помеченные блоки – это анонимные блоки с меткой. Они создаются обычно динамически и выполняются только один раз. Метка позволяет ссылаться на переменные блока, которые иначе были бы недоступны. Метка задается перед ключевым словом DECLARE или после ключевого слова END, например:

```
<<outerblock>>
DECLARE
counter INTEGER := 0;
BEGIN
  DECLARE
counter INTEGER := 1;
  BEGIN
    IF counter = outerblock.counter
    THEN
```

```
  DBMS_OUTPUT.PUT_LINE ('X_out ='||outerblock.counter);
ELSE DBMS_OUTPUT.PUT_LINE ('X_int ='||counter);
END IF;
  END;
END outerblock;
/
```

В этом примере при выполнении получим X_int = 1, а если поменять при запуске значение counter INTEGER := 1 во внешнем блоке, то X_out = 1. Также отметим, что в примере без метки блока невозможно было бы различить две переменные с одинаковым именем counter, но на практике лучше присвоить таким переменным разные имена.

2. Подпрограммы – это процедуры и функции, которые могут храниться в БД как автономные объекты, как часть пакета или как метод объектного типа. Подпрограммы обычно не изменяются и выполняются неоднократно, могут быть объявлены в других блоках, для обозначения секции объявлений в них не применяется ключевое слово DECLARE. Независимо, где объявлены, они выполняются явно посредством вызова процедуры или функции.

Процедуры – это именованные PL/SQL-блоки, которые могут принимать входящие параметры, но не возвращают явно никакого значения. Функции – это именованные PL/SQL-блоки, которые могут принимать входящие параметры и всегда возвращают значение. Разница между процедурой и функцией заключается в том, что функция обязательно должна возвращать значение в вызывающую программу.

Кроме того, к подпрограммам в PL/SQL можно отнести пакеты. Пакет – это именованный объект базы данных, который группирует логически связанные типы, программные объекты и подпрограммы PL/SQL. Пакеты обычно состоят из двух частей – спецификации и тела. Спецификация пакета – это интерфейс с приложениями, она объявляет типы, переменные, константы, исключения, курсоры и подпрограммы, доступные для использования в пакете. Тело пакета полностью определяет курсоры и подпрограммы, тем самым реализуя спецификацию пакета (CREATE PACKAGE и CREATE PACKAGE BODY).

3. Триггеры – это именованные блоки, которые ассоциируются с некоторым событием, происходящим в БД, обычно не изменяются и выполняются многократно неявным образом при наступлении соответствующих событий (INSERT, UPDATE, DELETE, CREATE, ALTER, DROP, запуск и останов сеанса работы с БД и др.).

Комментарии и литералы. В программах PL/SQL используются комментарии двух типов:

– однострочный, который начинается двумя символами «--» и действует до конца строки;

– многострочный, который начинается с символов «/*» и заканчивается символами «*/».

Также в программах используются литералы. Литерал – это значение, с которым не связан ни один идентификатор. Литерал может быть представлен:

– числовым значением, например 123, 345.6, 0.54, NULL;

– строковым значением, например, 'это Литерал';

– типом BOOLEAN: TRUE, FALSE, NULL;

– интервальным типом: INTERVAL '04-5' YEAR TO MONTH, INTERVAL '-4' YEAR, INTERVAL '5' MONTH, INTERVAL '5 04:03:02.01' DAY TO SECOND.

Строковый литерал чувствителен к регистру, заключается в одинарные кавычки. Строковый литерал нулевой длины может быть представлен как '' и имеет значение NULL. В случае, когда строковый литерал содержит символ кавычка ('), этот символ в соответствии со стандартом представляется в виде двух последовательных кавычек, например:

```
BEGIN
DBMS_OUTPUT.PUT_LINE ('литерал содержит кавычку '' и имеет длину > 0 ');
END;
/
```

Кроме этого, Oracle позволяет пользователю самостоятельно определять символ, который будет использоваться в качестве кавычек. Выглядеть такое переопределение может, например, так:

```
BEGIN
DBMS_OUTPUT.PUT_LINE (q'[литерал содержит кавычку' и имеет длину > 0]');
END;
/
```

В этом примере в качестве кавычек выступают квадратные скобки.

Типы данных и объявление переменных. Передача информации между PL/SQL и БД осуществляется с помощью переменных. Переменная (variable) – это область памяти, которая может быть считана или присвоена программой. Переменные объявляются в разделе объявлений блока (DECLARE).

Каждая переменная ассоциируется с определенным типом данных. Тип данных определяет вид информации, которая может храниться в данной переменной. Переменные PL/SQL могут иметь тот же тип данных, что и столбцы таблицы базы данных.

Типы данных в PL/SQL сгруппированы в скалярные (scalar), составные (composite), ссылки (reference) и большие объекты (LOB)[ссылка на конспект].

СУБД Oracle предоставляет следующие предопределенные типы данных:

CHAR – хранит строки фиксированной длины. Допустимая длина поля типа CHAR в таблице от 1 до 2000 байт.

VARCHAR2 – хранит строки переменной длины. Максимальная длина поля VARCHAR2 в таблице – 4000 байт. Максимальная длина

PL/SQL-переменной – 32 767 байт. Тип данных VARCHAR – синоним VARCHAR2, но его использования в Oracle стоит избегать.

NCHAR и NVARCHAR2 – являются Unicode-типами, они хранят Unicode-символы.

LONG – может хранить символьные данные переменной длины и содержит до 2 ГБ информации. Вместо LONG следует использовать CLOB или NCLOB.

NUMBER – хранит числа с фиксированной и плавающей запятой (до 38 цифр).

DATE – хранит момент времени (дату и время) в таблице, включая год, месяц, день, часы, минуты и секунды (после полуночи).

TIMESTAMP – расширяет тип данных DATE и хранит год, месяц, день, часы, минуты, секунды и доли секунды.

INTERVAL YEAR TO MONTH – представляет собой промежуток времени и содержит годы и месяцы.

INTERVAL DAY TO SECOND – представляет собой промежуток времени и содержит дни, часы, минуты и секунды.

LOB – включает BLOB, CLOB, NCLOB и BFILE, которые позволяют хранить большие блоки неструктурированных данных (текст, картинки, видео) до 4 ГБ в размере.

BLOB – хранит неструктурированные двоичные данные.

CLOB и NCLOB – хранят символьные данные. CLOB хранит символы в кодировке, установленной в БД, а NCLOB – в Unicode-кодировке.

BFILE – хранит неструктурированные двоичные данные в файлах операционной системы вне БД. BFILE-поле таблицы содержит указатель (locator) на внешний файл.

RAW и LONG RAW – используются для данных, которые никак не должны быть интерпретированы базой данных (например, при перемещении данных между разными системами). Эти типы предназначены для двоичных данных и байтовых строк.

СУБД Oracle использует тип данных ROWID для хранения уникального адреса каждой строки в базе данных. Physical Rowids хранят адреса строк обычных таблиц, кластерных таблиц, табличных партиций (partitions), индексов и индексных партиций. Logical Rowids хранят адреса строк в индекс-организованных таблицах (index-organizedtables).

В программах PL/SQL имеется также возможность определять пользовательские типы, которые могут быть как подтипами стандартных типов, так и более сложными конструкциями, такими как записи, коллекции или объекты. Например:

```
DECLARE
-- подтип без ограничений
SUBTYPE TMyFloat IS NUMBER(15,5);
```

```
varFloat TMyFloat;
-- подтип с ограничением
SUBTYPE TDayInMonth IS BINARY_INTEGER RANGE 1..31;
d TDayInMonth := 1;
BEGIN
NULL;
END;
/
```

Здесь определен собственный подтип TMyFloat на основе типа NUMBER и подтип TdayInMonth, который является целым в диапазоне от 1 до 31. Другими словами, подтипы без ограничений – это альтернативные имена стандартных типов данных.

Объявления переменных. Для объявления переменных в блоках PL/SQL существует специальный раздел. Переменные объявляются по следующим правилам:

имя_переменной тип_данных [CONSTANT] [NOT NULL] [{:=| DEFAULT} начальное_значение]

Например:

```
DECLARE
id INTEGER := 0;
standard INTEGER DEFAULT 500;
```

Если переменной присваивается значение по умолчанию, то при ее объявлении можно также указать, что она не должна принимать значение NULL, воспользовавшись указанием ограничения NOT NULL.

Вложенные блоки определяют область видимости объявленных переменных.

Существует понятие объявления переменной с привязкой, которое обладает рядом преимуществ, когда необходимо присваивать переменной значения из другого источника, например из поля таблицы. Привязывая переменную, можно установить ее тип на основе типа уже определенной структуры данных. Существует два вида привязки:

– скалярная – с помощью атрибута %TYPE;

– привязка к записи – с помощью атрибута %ROWTYPE.

Данные атрибуты используются для объявления переменных, констант и даже определяемых пользователями подтипов и составных типов, соответствующих свойствам столбцов и таблиц баз данных. Использование атрибутов не только упрощает объявление программных конструкций, но и делает программы более удобными для модификации БД. Привязка переменной освобождает от необходимости контролировать изменения в полях базовых таблиц БД, которые используются процедурами и функциями. Например:

```
sal employees.salary%TYPE;
MyFloatType NUMBER(15,5);
SUBTYPE   TMyFloat IS MyFloatType%TYPE;
```

Переменная, объявленная с атрибутом %ROWTYPE, фактически представляет собой запись или строку таблицы и может использоваться с другими составными конструкциями, например:

```
DECLARE
TYPE parts_table IS TABLE OF departments%ROWTYPE;
current_part parts_table;
```

Здесь объявляется вложенная таблица (один из видов коллекций PL/SQL, которые рассматриваются в лабораторной работе №6) parts_table, структура которой повторяет строку таблицы departments, и переменная current_part этого типа.

Запись RECORD – это составной тип данных, который представляет собой группу связанных полей, каждое из которых имеет свое имя и тип. Запись в PL/SQL задается пользователем и определяется как тип:

```
TYPE имя_типа IS RECORD (описание_поля_1 [,описание_поля_1]...);
```

где описание_поля_1 имеет вид

```
имя_поля тип_поля [[NOT NULL] {:= | DEFAULT} выражение]
```

Например:

```
TYPE jobs_type IS RECORD (
job_id          jobs.job_id%Type,
min_salary   NUMBER (6),
max_salary   NUMBER (7)
);
```

В отличие от других сложных типов, таких как VARRAY, NESTED TABLE, OBJECT, запись RECORD не может быть сохранена в БД.

Инициализация полей записи может быть описана при ее объявлении при помощи слова DEFAULT или оператора присваивания, например:

```
TYPE jobs_type IS RECORD (
job_id          jobs.job_id%TYPE := ' ',
min_salary   NUMBER(6) :=0,
max_salary   NUMBER(7) :=0
);
```

Обращение к записи происходит по имени переменной. Ссылка на поле записи происходит по имени переменной и через точку имени поля. При помощи атрибута %ROWTYPE переменная типа запись может быть сопоставлена со строкой любой таблицы в БД.

Переменная с типом запись может быть передана в качестве параметра в процедуру или функцию либо возвращена функцией в качестве результата. Рассмотрим пример работы с записями для стандартной учебной БД Oracle – human resources (HR).

```
DECLARE
TYPE jobs_type IS RECORD (
job_id          VARCHAR2(10),
min_salary   NUMBER(6),
max_salary   NUMBER(7)
);
myjob jobs_type;
BEGIN
SELECT job_id, min_salary, max_salary INTO myjob FROM jobs
WHERE job_id = 'IT_PROG';
DBMS_OUTPUT.put_line (myjob.job_id
|| ':'
|| myjob.min_salary
|| '-'
|| myjob.max_salary);
END;
/
```

Управление ходом выполнения программ. Как и во всех языках программирования, в PL/SQL существуют операторы управления вычислительным процессом, а именно: операторы условного, итерационного и последовательного управления.

Оператор IF. Оператор IF в PL/SQL имеет несколько разновидностей.

1. IF <условие> THEN
<последовательность операторов, когда результат проверки равен TRUE>
END IF;
2. IF <условие> THEN
<последовательность операторов, когда результат проверки равен TRUE>
ELSE
<последовательность операторов, когда результат проверки равен FALSE>
END IF;
3. IF <условие1> THEN
<последовательность операторов, когда результат проверки 1 равен TRUE>
ELSIF <условие 2> THEN
<последовательность операторов, когда результат проверки 2 равен TRUE>
[ELSE <операторы else>]
END IF;

Конструкция IF-THEN-ELSIF в какой-то мере является аналогом оператора CASE.

```
DECLARE
x   NUMBER := 6;
BEGIN
  IF x = 3 THEN
DBMS_OUTPUT.PUT_LINE ('x=3');
  ELSIF x = 5 THEN
DBMS_OUTPUT.PUT_LINE ('x=5');
  ELSIF x = 10THEN
DBMS_OUTPUT.PUT_LINE ('x=10');
  ELSE
DBMS_OUTPUT.PUT_LINE ('x NOT IN (3,5,10) ');
END IF;
END;
/
```

Оператор CASE. Начиная с Oracle9i поддерживаются два типа операторов CASE:

– простой оператор CASE связывает одну или несколько последовательностей операторов PL/SQL с соответствующим значением (последовательность для выполнения выбирается с учетом результатов вычисления выражения, возвращающего указанное значение);

– поисковый оператор CASE выбирает для выполнения одну из последовательностей операторов в зависимости от результатов вычисления списка логических условий (выполняется последовательность операторов, связанная с первым условием, результат проверки которого оказался равным TRUE).

В дополнение к операторам CASE PL/SQL поддерживает CASE-выражения. Выражение CASE очень похоже на оператор CASE, оно позволяет выбрать для вычисления одно или несколько выражений. В итоге получается одно значение, тогда как результатом работы оператора CASE является выполнение последовательности операторов PL/SQL.

Оператор CASE имеет следующий синтаксис:

```
CASE <выражение>
  WHEN <результат 1> THEN <операторы 1>
  WHEN <результат 2> THEN <операторы 2>
 …
  WHEN <результат N> THEN <операторы N>
[ ELSE
<операторы else> ]
END CASE;
```

Предыдущий пример можно переписать в виде

```
DECLARE
x   NUMBER := 6;
BEGIN
  CASE x
    WHEN 3THEN
DBMS_OUTPUT.PUT_LINE ('x=3');
    WHEN 5 THEN
DBMS_OUTPUT.PUT_LINE ('x=5');
    WHEN 10 THEN
DBMS_OUTPUT.PUT_LINE ('x=10');
    ELSE
DBMS_OUTPUT.PUT_LINE ('x not in (3,5,10) ');
END CASE;
END;
/
```

Если в этом примере не задать ключевое слово ELSE, то PL/SQL инициирует исключение

```
ERROR at line 1:
ORA-06592: CASE not found while executing CASE statement
ORA-06512: at line 4
```

В этом заключается основное отличие оператора CASE от IF..ELSIF. В случае оператора IF при отсутствии ELSE и в случае невыполнения условия ничего не происходит и управление передается следующему за END IF оператору.

Синтаксис поискового оператора CASE:

```
CASE
  WHEN <выражение 1> THEN <операторы 1>
  WHEN <выражение 1> THEN <операторы 2>
…
  WHEN <выражение 1> THEN <операторы N>
[ ELSE
<операторы else>]
END CASE;
```

Максимальное количество аргументов в операторе CASE 255, причем каждое выражение «WHEN ... THEN» рассматривается как два аргумента.

Выражения CASE. Выражения CASE работают аналогично операторам CASE, но не для исполняемых операторов, а для выражений. Простое выражение CASE позволяет выбрать для вычислений одно из нескольких выражений на основе заданного скалярного значения:

```
<простое_выражение_case> :=
 CASE <выражение>
WHEN<результат1>THEN<результирующее выражение 1>
WHEN<результат2>THEN<результирующее выражение 2>
...
  ELSE
<результирующее выражение else>
  END;
```

Поисковое выражение CASE:

```
<поисковое_выражение_case> :=
 CASE
WHEN<выражение1>THEN<результирующее выражение 1>
WHEN<выражение2>THEN<результирующее выражение 2>
...
  ELSE
<результирующее выражение else>
  END;
/
```

Пример использования:

```
DECLARE
x   NUMBER :=6;
y   NUMBER :=0;
 BEGIN
y:= CASE x
    WHEN 3  THEN 9
    WHEN 5  THEN 15
    WHEN 10 THEN 50
    ELSE 100
    END;
    DBMS_OUTPUT.PUT_LINE ('x='||x||' y='||y);
END;
/
```

Оператор CASE также можно использовать в SQL-запросах:

```
SELECT job_id, job_title,
CASE WHEN max_salary <5000 THEN 'младший персонал'
        WHEN max_salary BETWEEN 5000 AND 10000 THEN 'АУП'
ELSE 'ВЫСШЕЕ РУКОВОДСТВО'
END
FROM jobs;
```

Циклы. Цикл начинается с зарезервированного слова, должен содержать условие завершения и заканчиваться оператором END LOOP. В PL/SQL имеется несколько вариантом циклов:

1) простые циклы;
2) условные циклы (WHILE);
3) интерактивные циклы (FOR).

Пути выхода из циклов:

– EXIT – безусловный выход из цикла. Используется посредством применения оператора IF.

– EXIT WHEN – выход при выполнении условия.

– GOTO – выход из цикла во внешний контекст.

Синтаксис *простого цикла*:

```
LOOP
<исполняемые операторы>
<EXIT>|<EXIT WHEN>
END LOOP;
```

Простой цикл может завершаться оператором EXIT или EXIT WHEN<условие>, который должен в теле цикла присутствовать обязательно, иначе цикл выполняется бесконечно. Простой цикл применяется:

– когда неизвестно, сколько раз будет выполнен цикл;

– когда необходимо, чтобы он выполнился хотя бы один раз.

Рассмотрим пример:

```
DECLARE
i   INTEGER := 0;
y   INTEGER := 10;
BEGIN
  LOOP
    IF i >= y THEN EXIT;
    END IF;
i:= i + 1;
  END LOOP;
DBMS_OUTPUT.put_line (i);
END;
/
```

Цикл WHILE имеет следующий синтаксис:

```
WHILE условие LOOP
исполняемые_операторы;
END LOOP;
```

Цикл завершается, когда условие принимает значение FALSE или NULL.

Цикл FOR. Цикл FOR может использоваться с числовым счетчиком или с курсором. В случае цикла FOR с числовым счетчиком количество итераций известно до начала выполнения цикла. Общий вид цикла FOR:

```
FOR индекс_цикла IN [REVERSE] начальное_значение .. конечное_значение
    LOOP
    исполняемые_операторы;
    END LOOP;
```

При использовании цикла FOR следует придерживаться правил:

– переменную цикла можно не объявлять – она будет объявлена неявно с типом INTEGER и иметь область действия – тело цикла, поэтому ссылаться на нее вне цикла нельзя;

– внутри цикла нельзя изменять счетчик цикла и границы диапазона;

– используйте REVERSE для обратного счета, когда счетчик уменьшается от начального значения к конечному;

– в цикле нельзя задавать шаг приращения, счетчик всегда увеличивается на 1;

– допускается использовать операторы EXIT / EXIT WHEN.

```
/* первый пример с диапазоном, определяемым переменной*/
DECLARE
m   INTEGER := 3 * 2;
BEGIN
  FOR i IN 1 ..m
  LOOP
DBMS_OUTPUT.put_line ('m=' || m || ' i=' || i);
  END LOOP;
END;
/
/* второй пример с диапазоном, определяемым переменной*/

BEGIN
dbms_output.enable(100);
  FOR i IN reverse 1..10 LOOP
DBMS_OUTPUT.put_line ('i='||i);
END LOOP;
END;
/
```

Циклу можно присвоит имя, воспользовавшись для этого меткой:

```
<<name_for>>
FOR i IN 1..4 LOOP
. . .
END LOOP;
```

Использование меток цикла упрощает сопровождение программы, когда используется длинный цикл со вложенными в него другими циклами.

Оператор GOTO. Оператор GOTO передает управление операторам, следующим за указанной в GOTO меткой.

```
GOTO <<имя метки>>
```

После метки должен быть хотя бы один исполняемый оператор.

```
/* пример для схемы HR */
DECLARE
cntreg INTEGER := 6;
reg     INTEGER;
BEGIN
  SELECT MAX (region_id) INTO reg FROM regions;
  FOR i IN reg + 1 ..reg + cntreg
  LOOP
    IF i> 8 THEN
      GOTO label_exit;
    END IF;
    INSERT INTO regions (region_id, region_name ) VALUES (i, 'TEST');
  END LOOP;
<<label_exit>>
NULL;
END;
/
```

В этом примере показано также использование оператора NULL. Оператор NULL ничего не делает, а просто передает управление следующему за ним оператору, поэтому используется в случае, когда по правилам синтаксиса обязательно присутствие хотя бы одного оператора, но по логике программы не надо выполнять никаких действий. В примере, если не использовать NULL после метки <<label_exit>>, то программа PL/SQL сгенерирует исключение.

Оператор GOTO в языках высокого уровня является объектом критики, поскольку его чрезмерное применение приводит к созданию нечитаемого и неподдерживаемого кода. Тем не менее использование GOTO иногда допустимо, так как может значительно упростить код (например, избавить от необходимости создания вспомогательных переменных и операторов условия).

Ограничения при использовании GOTO:

В PL/SQL на использование операторов GOTO налагаются определенные ограничения:

– нельзя передавать управление программой во внутренний блок, цикл или оператор IF;

– запрещается передавать управление из одной последовательности операторов условного оператора IF в другую;

– нельзя передавать управление из обработчика исключительных ситуаций обратно в текущий блок.

Работа с курсорами и курсорными переменными. Основное назначение языка PL/SQL – создание программ для работы с БД. Программа может взаимодействовать с БД только посредством SQL, используя DML-операторы, курсоры и динамический SQL.

При обработке любого SQL-оператора Oracle выделяет область памяти, называемую контекстной областью (context area). Она содержит информацию, необходимую для начала и завершения обработки SQL-оператора, а именно: число строк, обрабатываемых оператором, указатель на представление этого оператора после синтаксического анализа (parsing) и активный набор (activeset) – набор строк, возвращаемых запросом.

Таким образом, все базовые операторы DML – это курсоры. При выполнении INSERT, UPDATE и других операторов в PGA – глобальной системной области – всегда открывается курсор. Следовательно, курсор – это указатель на контекстную область памяти, с помощью которого программа на PL/SQL может управлять контекстной областью и ее состоянием во время обработки оператора.

Курсор может представлять собой любое допустимое предложение языка SQL. Также курсор является основным базовым «кирпичиком» для построения блоков PL/SQL. Курсоры обеспечивают циклический механизм оперирования наборами данных в БД. Курсор может возвращать одну или несколько строк данных или вообще ни одной.

В программах PL/SQL используется два типа курсоров: явные и неявные. Неявные курсоры, как правило, возвращают одну строку. Если курсор возвращает более одной строки, то следует использовать явные курсоры или курсоры в цикле FOR-LOOP.

Неявные курсоры используются:

– для внесения изменений в таблицы БД с помощью операторов INSERT, UPDATE или DELETE;

– для извлечения данных с помощью оператора SELECT INTO, который возвращает скалярное значение или только одну строку в результирующем множестве запроса.

Неявный курсор вида SELECT INTO позволяет делать выборку данных в переменную или переменные.

```
/*  пример курсора для схемы HR */
DECLARE
```

```
s   VARCHAR2 (500);
BEGIN
  SELECT first_name || ' ' || last_name || ':' || email INTO s
  FROM employees WHERE employee_id = 105;
DBMS_OUTPUT.put_line (s);
END;
/
```

Для работы со строками, извлекаемыми из результирующего множества многострочного запроса, необходимо использовать явный курсор. Типичная последовательность действий при операциях с явными курсорами следующая:

1. Объявление курсора и структуры данных, в которую будут помещены найденные строки.
2. Открытие курсора.
3. Последовательная выборка данных.
4. Закрытие курсора.

Рассмотрим полный синтаксис объявления явного курсора в разделе объявления переменных:

```
CURSOR имя_курсора [(параметр1[, параметр2]...)]
[RETURN возвращаемый_тип] IS SELECT_выражение;
```

где возвращаемый_тип – запись или строка из базы данных.

Примеры объявления явного курсора:

```
DECLARE
-- курсор без параметров
CURSOR emp_cur
  IS
    SELECT employee_id, job_id, salary FROM employees;

  -- курсор с параметрами
  CURSOR emp_cur_par (dep IN NUMBER)
  IS
   SELECT employee_id, job_id, salary FROM employees
    WHERE department_id = dep;

  -- курсор с предложением RETURN
CURSOR emp_cur_return (dep IN NUMBER) RETURN employees%ROWTYPE
  IS SELECT * FROM employees WHERE department_id = dep;
BEGIN
  NULL;
END;
/
```

Открытие курсора выполняется с помощью оператора OPEN:

OPEN имя_курсора [аргумент 1 [, аргумент2 ..]];

При попытке открыть уже открытый курсор генерируется ошибка ORA-06511: PL/SQL: cursor already open.

Для проверки состояния курсора существуют следующие атрибуты:

%ISOPEN – возвращает TRUE, если курсор открыт, иначе FALSE.

%FOUND – возвращает TRUE, если последний FETCH вернул данные, иначе FALSE. Сразу после открытия имеет значение NULL.

%NOT FOUND – альтернатива %FOUND, если последний FETCH не вернул данные, иначе FALSE. Сразу после открытия имеет значение NULL.

%ROWCOUNT – возвращает количество строк, выбранных на данный момент из курсора. Сразу после открытия и до первого оператора FETCH имеет значение 0.

Извлечение данных выполняется при помощи оператора FETCH, имеющего синтаксис:

FETCH имя_курсора INTO переменные_или_курсорная_переменная;

Закрытие курсора производится оператором

CLOSE имя_курсора

Пусть необходимо увеличить заработную плату всем сотрудникам департамента на 10 % и вывести изменения на экран.

```
/* Изменение заработной платы сотруднику, схема HR */
DECLARE
  -- отделIT
vdep departments.department_id%TYPE := 60;
-- % повышения (снижения) заработной платы
pctsal NUMBER (5, 2) := 30;
-- курсор с параметром
  CURSOR listemp_cur (dep IN NUMBER)
  IS
    SELECT emp.employee_id, emp.job_id, emp.salary, jobs.min_salary,
jobs.max_salary FROM employees emp, jobs
      WHERE department_id = dep AND emp.job_id = jobs.job_id;
  -- курсорная переменная
listemp listemp_cur%ROWTYPE;
newsal employees.salary%TYPE;
BEGIN
  OPEN listemp_cur(vDep);
```

```
  FETCH listemp_cur  INTO listemp;
  WHILE listemp_cur%FOUND
LOOP
newsal := listemp.salary * (100 + pctsal) / 100;
    IF newsal < listemp.min_salary
    THEN newsal := listemp.min_salary;
    ELSIF newsal > listemp.max_salary
    THEN newsal:= listemp.max_salary;
    END IF;
DBMS_OUTPUT.put_line (   'Сотрудник '
                      || listemp.employee_id
                      || ' старая заработная плата ='
                      || listemp.salary
                      || ' новая заработная плата ='
|| ' = '
                      || newsal);
    FETCH listemp_cur INTO listemp;
  END LOOP;
  CLOSE listemp_cur;
END;
/
```

В примере обращение к данным, возвращаемым курсором, происходит при помощи курсорной переменной listemp. Курсорная переменная в отличие от обычной представляет собой указатель на рабочую область, где размещаются данные курсора.

Для создания курсорной переменной есть два способа: определение REF CURSOR TYPE и непосредственное объявление переменной в блоке PL/SQL.

При объявлении REF CURSOR TYPE возможно либо жестко прописывать тип, возвращаемый курсором, либо не прописывать тип вообще.

Ссылка на курсор дает возможность не заводить структуры курсора в клиентской программе, а ограничиться в ней выделением памяти только для адреса курсора, в то время как сам курсор будет располагаться целиком в СУБД.

Чтобы завести в PL/SQL переменную-ссылку на курсор, нужно сначала описать ее тип. Это делается в разделе объявлений с помощью предложения TYPE:

TYPE имя_типа_ссылки_на_курсор IS REF CURSOR [RETURN тип_записи];

Если конструкция RETURN присутствует, ссылка на курсор называется строгой; если нет – нестрогой. Нестрогая ссылка на курсор может ссылаться на любой курсор (запрос), а строгая – только на тот, который возвращает результат указанного типа.

Пример объявления типов курсорных переменных – ссылок на курсор:

```
  -- нестрогий указатель на курсор
  TYPE dep_cur_type IS REF CURSOR;

--строгий указатель на курсор с привязкой к таблице employees
TYPE emp_cursor_type IS REF CURSOR
RETURN employees%ROWTYPE;

 -- строгий указатель на курсор с привязкой к записи

TYPE jobs_type IS RECORD (
job_id        VARCHAR2 (10),
min_salary   NUMBER (6),
max_salary   NUMBER (7));

TYPE job_cursor_type IS REF CURSOR
RETURN jobs_type;
```

Курсорная переменная, объявленная как REF CURSOR, может использоваться в качестве параметра процедуры/функции или в качестве значения, возвращаемого функцией.

Объявление курсорной переменной описанных выше типов ссылок на курсоры:

```
v_dep_cur          dep_cur_type;
v_emp_cursor       emp_cursor_type;
v_job_cursor       job_cursor_type;
```

Далее в программе происходит открытие курсора с помощью переменной – ссылки на курсор:

```
OPEN ссылка_на_курсор FOR предложение_SELECT;
```

Команды FETCH и CLOSE используются как обычно, только вместо имени курсора указывается имя переменной – ссылки на курсор.

Использование курсорных циклов FOR. Обработка курсоров с помощью циклов FOR в некоторых случаях значительно упрощает код программы. Цикл FOR с курсором – это цикл, связанный с курсором, явно заданным или представленным в виде оператора SELECT непосредственно в цикле.

Перепишем предыдущий пример с использованием курсоров FOR:

```
-- Изменение заработной платы сотруднику с курсорным циклом FOR
DECLARE
  -- отдел IT
vdep departments.department_id%TYPE := 60;
-- % повышения (снижения) заработной платы
```

```
pctsal NUMBER (5, 2) := 30;
newsal employees.salary%TYPE;
BEGIN
  FOR listemp IN (SELECT emp.employee_id, emp.job_id, emp.salary,
jobs.min_salary, jobs.max_salary FROM employees emp, jobs
 WHERE department_id = vdep AND emp.job_id = jobs.job_id)
  LOOP newsal := listemp.salary * (100 + pctsal) / 100;
    IF newsal < listemp.min_salary  THEN newsal := listemp.min_salary;
    ELSIF newsal > listemp.max_salary THEN newsal := listemp.max_salary;
    END IF;
DBMS_OUTPUT.put_line (   'Сотрудник '
                      || listemp.employee_id
                      || ' старая заработная плата ='
                      || listemp.salary
                      || ' новая заработная плата ='
                      || ' = '|| newsal);
  END LOOP;
END;
/
```

В курсорном цикле FOR автоматически объявляется переменная или запись, с помощью которой можно считывать записи, курсор не нужно явно открывать и закрывать. Закрывается курсор сам, когда из него выбирается последняя строка.

Обработка исключительных ситуаций. В программах PL/SQL подпрограммы обработки ошибок представляют собой обработчики исключительных ситуаций EXCEPTION. Как только инициируется исключение, нормальная работа программы прерывается и управление передается в блок обработки исключений. Поэтому важно помнить, что когда при проектировании программ необходимо перехватить и обработать исключение, не заканчивая на этом работу основной программы, необходимо пользоваться вложенными блоками.

Для того чтобы перехватить исключение, необходимо написать для него обработчик. В одном разделе EXCEPTION может быть написано несколько обработчиков. По структуре они напоминают оператор CASE и имеют следующий синтаксис:

```
EXCEPTION
  WHEN исключение_1 THEN
   обработка_ исключения_1_исполняемые_операторы
  WHEN исключение_2 THEN
   обработка_ исключения_2_исполняемые_операторы
   . . .
  WHEN исключение_n THEN
   обработка_ исключения_n_исполняемые_операторы
```

```
[ WHEN OTHERS THEN
обработка_прочих_исключений_исполняемые_операторы ]
END;
```

Также возможен вариант объединения нескольких исключений в одном обработчике. Например:

```
WHEN NO_DATA_FOUND OR ZERO_DIVIDE THEN
```

Если для инициированного исключения не найдено ни одного обработчика, но при этом присутствует обработчик WHEN OTHERS, то его обработка выполняется именно в этом блоке. Если исключение все же не было обработано, то PL/SQL возвращает исключение в вызвавшую родительскую программу или же в ту среду, из которой была запущена программа. Это может быть SQL*Plus, TOAD или любое другое клиентское приложение.

Исключения бывают двух видов:

- определенные программистом (user-defined exception);
- системные, например, нехватка памяти или нарушение целостности БД.

Исключения, определяемые программистом, объявляются в блоке декларации, там же, где объявляются переменные и курсоры. Например:

```
DECLARE
vjob_id jobs.job_id%TYPE :='0';
exp_badempid EXCEPTION;
BEGIN
NULL;
END;
/
```

Область действия объявленных исключений аналогична области действия переменных и курсоров.

Системные исключения генерируются системой и, как правило, выдаются в формате

```
ORA-номер_ошибки сообщение_об_ошибке
```

Некоторые из системных исключений имеют имена – это так называемые предопределенные исключения. Список таких исключений приведен в прил. 4 (табл. П.4.1). Например, программа распознает исключительную ситуацию NO_DATA_FOUND в случае, если в результирующем множестве оператора SELECT INTO нет строк; исключительную ситуацию TOO_MANY_ROWS – если в результирующем множестве этого оператора более одной строки; исключительную ситуацию DUP_VAL_ON_INDEX (повторяющееся значение в индексе) – если оператор INSERT или UPDATE дублирует ключевое значение, уже находящееся в таблице.

При обработке исключений запоминать номера ошибок бывает затруднительно. Поэтому в PL/SQL имеется механизм связывания номера ошибки с его названием с помощью директивы компилятору – предопределенной прагмы EXCEPTION_INIT, сообщающей компилятору имя исключения, которое ассоциируется с конкретным кодом ошибки Oracle. Это позволяет обращаться к любому внутреннему исключению по имени, написав для него специальный обработчик. Прагма EXCEPTION_INIT указывается в декларативной части блока, подпрограммы или пакета PL/SQL с использованием следующего синтаксиса:

PRAGMA EXCEPTION_INIT (имя_исключения, код_ошибки_Oracle);

где имя_исключения – это имя исключения, ранее уже объявленного в этом блоке.

Пусть программа в качестве аргумента принимает цифровое значение в виде символьной строки и переводит его в числовой формат. Выполним этот блок в SQL*Plus.

```
DECLARE
vals VARCHAR2 (10) := :S;
valn   NUMBER;
BEGIN
DBMS_OUTPUT.PUT_LINE ('строка ='||vals);
valn := TO_NUMBER (vals);
DBMS_OUTPUT.PUT_LINE ('число  ='||valn);
END;
/
```

При выполнении получим приглашение для ввода параметра S и введем, например, число 123.36, в результате получим:

```
строка = 123.36
число = 123.36
Statement processed.
```

Когда пользователь задает аргумент в правильном формате (т. е. пригодном для конвертирования в число), программа отрабатывает нормально. Но стоит ввести неверно заданный аргумент, например, 123/36, то получим

ORA-06502: PL/SQL: numeric or value error: character to number conversion error

Хорошим стилем программирования является обработка подобных ситуаций на этапе написания программного кода, в данном случае воспользуемся прагмой EXCEPTION_INIT.

```
DECLARE
vals   VARCHAR2 (10) := :S;
valn   NUMBER;
expChar_To_number EXCEPTION;
  PRAGMA EXCEPTION_INIT (expChar_To_number, -06502);
BEGIN
DBMS_OUTPUT.put_line ('строка =' || vals);
valn := TO_NUMBER (vals);
DBMS_OUTPUT.PUT_LINE ('число  =' || valn);
EXCEPTION
  WHEN expChar_To_number THEN
DBMS_OUTPUT.PUT_LINE ('ошибка: аргумент ' || vals || ' не является числом');
END;
/
```

Инициирование исключений. Ссылаться на исключения можно несколькими способами.

1. При помощи оператора RAISE:

```
RAISE [имя_пакета].имя_исключения;
```

или просто:

```
RAISE;
```

Первый вариант используется для инициирования объявленных в текущем блоке или в пакете исключений либо для инициирования системных исключений по имени.

Второй вариант используется тогда, когда в блоке EXCEPTION необходимо повторно инициировать исключение.

1. Oracle предоставляет возможность инициировать исключения при помощи процедуры RAISE_APPLICATION_ERROR, которая в отличие от оператора RAISE позволяет связать с номером исключений сообщение об ошибке и не требует заранее определенного исключения:

```
RAISE_APPLICATION_ERROR (номер_ошибки, сообщение_об_ошибке);
```

где номер_ошибки – это целое число в пользовательском диапазоне от –20 999 до – 20 000; сообщение_об_ошибке – строка, длина которой не превышает 2048 символов.

При обработке исключений в ветке WHEN OTHERS обработчику иногда необходимо идентифицировать возникающую ошибку. Для этого имеются встроенные функции SQLCODE и SQLERRM. Функция SQLCODE возвращает номер

текущей ошибки (0 указывает, что в стеке нет ни одной ошибки). Функция SQLERRM возвращает текстовое сообщение о текущей или указанной разработчиком ошибке. Для исключений, определяемых пользователем, SQLCODE возвращает 1, а SQLERRM возвращает «User-defined Exception» (определенное пользователем исключение). Максимальная длина строки, возвращаемой SQLERRM, составляет 512 байт. Значения функций SQLCODE и SQLERRM сначала присваиваются локальным переменным, и только потом эти переменные указываются в SQL-операторе. Например, эти функции часто используются для сохранения в дополнительной таблице сведений об ошибках (протоколирование):

```
CREATE TABLE errors (num number, msg varchar2(256));

DECLARE
b number;
err_num NUMBER;
err_msg CHAR (100);
BEGIN
b := 2/0;
EXCEPTION
WHEN OTHERS THEN  err_num := SQLCODE;
err_msg := SUBSTR(SQLERRM, 1, 100);
INSERT INTO errors VALUES (err_num, err_msg);
END;
/
```

В примере переменная err_msg ограничена 100 символами. Если текст сообщения об ошибке превышает 100 символов, операция err_msg := SQLERRM; сама вызывает стандартную исключительную ситуацию VALUE_ERROR. Во избежание этого используется встроенная функция SUBSTR, обеспечивающая присваивание переменной v_ErrorText не более чем 100 символов.

При обработке ошибок можно не только выдавать приемлемые для пользователя сообщения об ошибках, вести протоколирование, но и просто их игнорировать.

```
DECLARE
vals VARCHAR2 (10) := :S;
valn NUMBER :=0;
BEGIN
BEGIN
   -- смогли конвертировать без ошибок
valn := TO_NUMBER (vals);
EXCEPTION
WHEN OTHERS THEN
-- игнорируем ошибку конвертации
```

```
NULL;
END;
DBMS_OUTPUT.PUT_LINE ('строка =' || vals);
DBMS_OUTPUT.PUT_LINE ('число  =' || valn);
END;
/
```

Получим результат:

```
строка =123/36
число =0
```

Хранимые процедуры и функции. Хранимые процедуры и функции в отличие от анонимных блоков сохраняются в базе данных и наряду с таблицами, представлениями и другими структурными единицами являются самостоятельными объектами БД Oracle. Процедуры и функции сохраняются в откомпилированном виде и по мере их вызова загружаются в разделяемый пул, поддерживаемый СУБД, откуда удаляются по мере заполнения пула в порядке частоты использования кода процедуры или функции. Наиболее редко используемый код удаляется раньше и при очередном вызове снова загружается в пул с диска. Такая организация способствует производительности выполнения вызываемых процедур и функций, поскольку исключает постоянную загрузку с диска исполняемого кода.

Процедуры. Общий синтаксис создания:

```
[CREATE [OR REPLACE]]
PROCEDURE имя_процедуры[(параметр1[, параметр2]...)]
[AUTHID {DEFINER | CURRENT_USER}]
    {IS | AS}
[PRAGMA AUTONOMOUS_TRANSACTION;]
[локальные объявления переменных]
BEGIN
Исполняемые_операторы
[EXCEPTION
обработка_ошибок]
END [имя процедуры];
```

где параметр объявляется как

```
имя_параметра [IN | OUT [NOCOPY] | INOUT [NOCOPY]] тип_данных
[{:= | DEFAULT} выражение]
```

Существуют три режима использования параметра:

IN – параметр только для чтения, не может изменяться в подпрограмме. Как правило, так описываются входные параметры, которые используются внутри модуля как константы. Режим IN применяется по умолчанию.

OUT – параметр только для записи, не может применяться для получения подпрограммой (процедурой или функцией) значения. Как правило, так описываются выходные параметры, с помощью которых модуль возвращает значение.

IN OUT – параметр для чтения и записи. Значения такого параметра можно передавать в подпрограмму и возвращать их обратно вызывающей программе.

Параметры могут передаваться двумя методами:

1) по ссылке – с соответствующим формальным параметром связывается не значение, а указатель на область памяти, где хранится значение фактического параметра;

2) по значению – значение фактического параметра копируется в соответствующий формальный параметр. Если программа завершается без инициирования исключений, то значение формального параметра присваивается фактическому параметру, иначе значение обратно не передается.

Когда в качестве параметров передаются большие структуры данных (списки, объекты и т. д.), объявленные с ключевым словом OUT или IN OUT, то при применении передачи по значению (что является режимом по умолчанию для параметров OUT, IN OUT) происходит копирование этих структур в формальные параметры, что может замедлить работу программы. Для передачи параметров по ссылке используется ключевое слово NOCOPY.

При указании типа параметра дается ссылка на сам тип, без указания размерности, например:

```
PROCEDURE Myproc (param1 IN  NUMBER(2));   -- ошибка

PROCEDURE Myproc (param1 IN NUMBER);   -- правильно
```

Атрибут AUTHID определяет, будет ли процедура выполняться с правами владельца – DEFINER (определяется по умолчанию) – или правами текущего пользователя – CURRENT_USER.

Прагма AUTONOMOUS_TRANSACTION – это инструкция компилятору, которая позволяет выполнить код процедуры в отдельной независимой транзакции и требует обязательного наличия в подпрограмме завершения транзакции оператором COMMIT или ROLLBACK.

В разделе локальных объявлений описываются локальные переменные, константы, курсоры, область действия которых определяется рамками процедуры.

Вызов процедуры осуществляется по ее имени. Значения для параметров процедуры могут передаваться либо по позиции параметра в списке параметров, либо по имени параметра. Список значений размещается в круглых скобках вслед за именем процедуры. Для вызова процедуры можно воспользоваться

командой SQL*Plus EXECUTE имя_процедуры или командой CALL имя_процедуры или выполнить анонимный блок.

```
CREATE OR REPLACE PROCEDURE show_the_date
IS
Today DATE DEFAULT SYSDATE;
BEGIN
--вывести текущую дату
DBMS_OUTPUT.PUT_LINE ('сегодня   ' ||today);
END show_the_date;
/
```

Вызов процедуры с помощью анонимного блока:

```
BEGIN
show_the_date;
END;
/
```

Создаем процедуру для изменения заработной платы одному сотруднику в схеме HR.

```
CREATE OR REPLACE PROCEDURE CHANGE_SALARY (empId IN number, newsal IN  number) IS
maxsal JOBS.MAX_SALARY%Type;
minsal JOBS.MIN_SALARY%Type;
sal employees.salary%TYPE;
v_job VARCHAR2(10);

BEGIN
SELECT job_id, salary INTO v_job, sal
FROM employees WHERE employee_id=empId;

IF sal <> newsal THEN
SELECT max_salary, min_salary INTO maxsal, minsal
FROM jobs WHERE job_id = (SELECT job_id FROM employees
WHERE employee_id=empId);
UPDATE employees SET salary=newsal WHERE employee_id=empId;
COMMIT;
DBMS_OUTPUT.PUT_LINE ('Сотрудник '||empID||': старая заработная плата = '|| sal||', новая заработная плата = '||newsal);
END IF;
END;
/
```

Вызов с передачей параметров по позиции:

```
BEGIN
CHANGE_SALARY (105,8000);
END;
/
```

Вызов с передачей параметров по имени:

```
BEGIN
CHANGE_SALARY (newsal => 9350, empId=>105);
END;
/
```

Процедуры нельзя применять для выполнения запросов, потому что они не возвращают результат.

Функции. Функции – это подпрограммы, которые подсчитывают и возвращают какое-либо значение. Возвращаемое функцией значение принадлежит к определенному типу данных. В отличие от процедур вызов функции всегда является частью оператора, т. е. включается в выражение либо служит в качестве значения по умолчанию. Функции, определенные пользователем, могут использоваться в SQL-выражениях.

Функции играют важную роль в обеспечении модульного подхода программирования. К примеру, реализация отдельного бизнес-правила или формулы должна помещаться в функцию. Любой запрос, возвращающий одну строку, может также объявляться внутри функции для его повторного использования.

```
[CREATE  [OR REPLACE ] ]
FUNCTION имя_функции[ (параметр1 [ , параметр2 ]... ) ]
RETURN тип_возвращаемого_результата
[ AUTHID { DEFINER  | CURRENT_USER } ]
[ PARALLEL_ENABLE ]
  [DETERMINISTIC]   [ PIPELINED  ]
    {IS | AS}
[ PRAGMA AUTONOMOUS_TRANSACTION; ]
[ локальные объявления переменных]
BEGIN
Исполняемые_операторы
[ EXCEPTION
обработка_ошибок]
END [ имя_функции];
```

где PARALLEL_ENABLE – атрибут, используемый для оптимизации, позволяет системе выполнять функцию параллельно в случае, когда она вызывается из оператора SELECT;

DETERMINISTIC – атрибут, используемый для оптимизации, позволяет системе применить сохраненную копию возвращаемого функцией значения, если это возможно;

PIPELINED – атрибут, указывающий, что результат табличной функции должен возвращаться построчно, с помощью оператора PIPE ROW.

тип_возвращаемого_результата – функция, которая может возвращать любые данные, определенные в Oracle.

Функция, которая возвращает количество отработанных сотрудником лет и месяцев для схемы HR:

```
CREATE OR REPLACE FUNCTION get_time_work (
empid employees.employee_id%TYPE,
dateval DATE DEFAULT SYSDATE
)
  RETURN VARCHAR2
IS
  RESULT  INTERVAL YEAR TO MONTH;
d employees.hire_date%TYPE;
BEGIN
  SELECT hire_date   INTO d  FROM employees
   WHERE employee_id = empid;
  RESULT:= (dateval - d) YEAR TO MONTH;
RETURN RESULT;
END;
/
```

Параметр dateval описан ключевым словом DEFAULT, что позволяет при обращении к функции не задавать его значение, если значение по умолчанию устраивает.

Вызов функции может осуществляться в любом месте исполняемого оператора, где возможно использование выражения, следовательно, вызвать функцию можно следующими способами:

– в команде присваивания:

```
DECLARE
time INTERVAL YEAR TO MONTH;
BEGIN
time := get_time_work (108);
DBMS_OUTPUT.PUT_LINE(time);
END;
```

– при задании значения по умолчанию:

```
DECLARE
time INTERVAL YEAR TO MONTH DEFAULT get_time_work (108);
BEGIN
DBMS_OUTPUT.PUT_LINE(time);
END;
```

– в логическом выражении:

```
DECLARE
time INTERVAL YEAR TO MONTH;
BEGIN
IF get_time_work (107) < INTERVAL '10-00' YEAR TO MONTH
THEN
time := get_time_work (107);
DBMS_OUTPUT.PUT_LINE(time);
END IF;
END;
```

– в команде SQL:

```
SELECT employee_id, first_name, get_time_work (employee_id)
FROM employees
WHERE (sysdate - hire_date) YEAR TO MONTH > get_time_work (114);
```

– как аргумент в списке параметров другой программной единицы, например:

```
BEGIN
CHANGE_HIRE_DATE (105, get_time_work (107));
END;
/
```

где CHANGE_HIRE_DATE – гипотетическая процедура для изменения даты приема на работу сотрудника.

Функции, которые в результате своей работы производят изменения в БД, нельзя применять в SQL-запросах.

Просмотреть существующие процедуры и функции можно с помощью запроса к словарю данных:

```
SELECT OBJECT_NAME, OBJECT_TYPE, STATUS
FROM USER_OBJECTS
WHERE OBJECT_TYPE IN ('PROCEDURE', 'FUNCTION');
```

Пакеты. Процедуры и функции можно группировать в пакеты. Пакеты инкапсулируют связанные функциональности в один автономный модуль.

Пакет состоит из двух компонентов:

– спецификация,

– тело.

Спецификация перечисляет все имеющиеся в пакете процедуры, функции, типы и объекты. Можно сделать их доступными для всех пользователей, у которых есть доступ к пакету. Сам код процедур и функций спецификация не содержит.

Тело пакета содержит сам код процедур и функций, которые объявлены в спецификации. Любая процедура или функция, содержащаяся в теле пакета и не упомянутая в спецификации, будет доступна только внутри тела пакета.

Создание спецификации пакета:

```
CREATE [OR REPLACE] PACKAGE имя_ пакета
{IS | AS}
спецификация пакета
END имя_пакета;
```

Создание тела пакета:

```
CREATE [OR REPLACE] PACKAGE BODY имя_ пакета
{IS | AS}
тело пакета
END имя_пакета;
```

Обращение к процедурам и функциям пакета происходит посредством точечной нотации.

Локальные модули. Локальный модуль – это процедура или функция, которая определена в разделе объявления переменных блока PL/SQL. Использование локальных модулей повышает модульность программы, уменьшает объем кода и улучшает читабельность программы. Локальным такой модуль называется по причине того, что он доступен только для использования внутри блока, где он объявлен. Вернемся к процедуре изменения заработной платы и перепишем ее:

```
CREATE OR REPLACE PROCEDURE CHANGE_SALARY2
 (empId IN NUMBER, newsal IN NUMBER)
IS

 TYPE EMP_INF_TYPE IS RECORD (
   job_id employees.job_id%TYPE :=0,
   salary employees.salary%TYPE := 0,
   min_sal employees.salary%TYPE := 0,
   max_sal employees.salary%TYPE := 0);

empInf EMP_INF_TYPE;
```

```
FUNCTION getInf (emp_id in Number)
return EMP_INF_TYPE
IS
result EMP_INF_TYPE;
BEGIN
SELECT emp.job_id, emp.salary, jobs.min_salary, jobs.max_salary INTO
result FROM employees emp, jobs
WHERE employee_id = empId AND emp.job_id = jobs.job_id;
RETURN result;
END getInf;

FUNCTION TestSalary (empInf IN EMP_INF_TYPE, newsal IN Number)
RETURN BOOLEAN
IS
BEGIN
IF newsal < empInf.min_sal OR newsal > empInf.max_sal THEN
RETURN FALSE;
ELSE RETURN TRUE;
END IF;
END TestSalary;

BEGIN
empInf :=getInf (empId);
   IF TestSalary(empInf, newsal) THEN
UPDATE employees SET salary = newsal WHERE employee_id = empId;
COMMIT;
DBMS_OUTPUT.PUT_LINE ('Сотрудник '||empID||':
старая заработная плата = '|| empinf.salary||',
новая заработная плата = '||newsal);
ELSE DBMS_OUTPUT.PUT_LINE('Сумма вне рамок должностного оклада');
   END IF;
END;
/
```

В примере используется две локальные функции: getInf, которая возвращает в переменную типа запись информацию о сотруднике и TestSalary, которая проверяет, входит ли новая заработная плата в допустимые по должности рамки.

Перегрузка модулей. Программы, которые существуют в одной и той же области видимости и имеют одинаковые имена, называются перегруженными.

```
DECLARE
val_s   VARCHAR2 (50) := 'выводим текстовую строку';
val_n   NUMBER := 452.36;
val_d   DATE := SYSDATE;
```

```
  PROCEDURE mywrite (VALUE IN VARCHAR2)
  IS BEGIN
DBMS_OUTPUT.PUT_LINE ('значение переменной ='' || VALUE || ''=');
  END;
  PROCEDURE mywrite (VALUE IN NUMBER)
  IS
s   VARCHAR2 (20) := TO_CHAR (VALUE, '99999.99');
BEGIN
DBMS_OUTPUT.put_line ('значение переменной =' || s);
END;
  PROCEDURE mywrite (VALUE IN DATE)
  IS
s   VARCHAR2 (20):= TO_CHAR (VALUE, 'dd.mm.yyyy hh:mi');
BEGIN
DBMS_OUTPUT.PUT_LINE ('значение переменной =' || s);
END;
BEGIN
mywrite (val_s);
mywrite (val_n);
mywrite (val_d);
END;
/
```

Перегрузка является подходящим решением при необходимости поддержки разных типов данных. Использование перегрузки имеет несколько ограничений. PL/SQL должен иметь возможность отличить друг от друга разные модули, имеющие одно и то же имя. Для этого используют списки параметров и/или сведения о подпрограмме (процедура или функция). Если у перегруженных подпрограмм отличаются только имена параметров, то такие подпрограммы должны вызываться только с использованием метода связывания параметров по имени. Все перегруженные подпрограммы должны быть объявлены в одном и том же блоке PL/SQL.

Задание к лабораторной работе

1. Создать процедуру.
2. Создать функцию.

Варианты заданий для написания процедур и функций приведены в прил. 5. Эти задания при необходимости можно усложнить или предложить другие (согласовать с преподавателем) в соответствии с бизнес-логикой варианта задания.

При создании следует выполнить следующие минимальные требования к синтаксису:

– использовать явный курсор или курсорную переменную, а также атрибуты курсора;

– использовать пакет DBMS_OUTPUT для вывода результатов работы в SQL*Plus;

– предусмотреть секцию обработки исключительных ситуаций, причем обязательно использовать как предустановленные исключительные ситуации, так и собственные (например, стоит контролировать наличие в БД значений, передаваемых в процедуры и функции как параметры);

3. Создать локальную программу, изменив код ранее написанной процедуры или функции.

4. Написать перегруженные программы, используя для этого ранее созданную процедуру или функцию.

5. Объединить все процедуры и функции, в том числе перегруженные, в пакет.

6. Написать анонимный PL/SQL-блок, в котором будут вызовы реализованных функций и процедур пакета с различными характерными значениями параметров для проверки правильности работы основных задач и обработки исключительных ситуаций.

Контрольные вопросы

1. Из каких частей состоит блок PL/SQL?
2. Какие существуют виды входных параметров?
3. Каковы основные типы программ, которые можно использовать в PL/SQL?
4. Какие способы передачи параметров в программу существуют?
5. Каким образом используется CASE в PL/SQL?
6. Чем отличаются друг от друга процедура и функция?
7. Каким образом можно выполнять процедуры и функции в SQL*Plus?
8. Как обрабатываются многострочные запросы в программах PL/SQL?
9. Как и какие атрибуты курсора могут использоваться в программе?
10. В каких случаях используется неявный курсор?
11. В чем отличие курсора от курсорной переменной?
12. Как производится обработка ошибочных ситуаций в программах PL/SQL?
13. Что такое локальные модули?
14. В каких случаях программа называется перегруженной?
15. Для чего используются пакеты?

ЛАБОРАТОРНАЯ РАБОТА №5

ТРИГГЕРЫ

Цель работы – изучить различные виды и особенности создания триггеров и планировщика БД, реализовать с их помощью бизнес-логику.

Теоретические сведения

Триггерами называют процедуры, которые выполняются в ответ на происходящие в базе данных события. Триггеры устанавливаются для проверки введенных в базу данных значений, выполнения аудита изменений, запрета выполнения операций.

Можно выделить следующие группы триггеров:

1. Триггеры DML применяются для проверки внесенных в таблицы изменений, которые запускаются на события: вставки, обновления, удаления данных из таблиц.
2. Триггеры DDL запускаются при выполнении DDL-инструкций, т. е. при создании, модификации, удалении таблиц или других объектов БД.
3. Триггеры уровня базы данных отслеживают события, касающиеся работы самой базы данных, такие как ее останов, запуск, возникновение ошибок Oracle, а также реагируют на такие пользовательские события как logon и logoff.
4. Триггеры INSTEAD OF – замещающие триггеры, запускаются непосредственно перед выполнением вставки, обновления, удаления и определяют, какие операции необходимо выполнить вместо соответствующей операции. Триггеры INSTEAD OF управляют операциями над представлениями (view), но не над таблицами и позволяют преобразовывать необновляемые представления в редактируемые, задавая их поведение в коде триггера.

Триггеры DML. Триггеры DML отслеживают события и операции, производимые над данными. Это наиболее распространенный вид триггеров. DML-триггер может запускаться:

– до выполнения операции – триггер BEFORE – или после выполнения операции – триггер AFTER. Триггеры BEFORE используются в случае, когда надо проверить возможность выполнения операции, определить значения полей;

– один раз для всей инструкции SQL – триггер уровня инструкции – или отдельно для каждой изменяемой записи – триггер уровня записи или строчный триггер (FOR EACH ROW). В этом случае, например, при обновлении 100 строк, триггер будет запускаться и срабатывать 100 раз.

Для манипулирования данными в триггере доступны так называемые псевдозаписи NEW и OLD, которые представляют собой структуры данных, аналогичные типу RECORD – записи PL/SQL. Псевдозапись NEW содержит значения полей после внесения изменений, OLD – значения полей до внесения изменений. Соответственно, псевдозапись NEW доступна в триггерах, обрабатыва-

ющих операции вставки и изменения, а OLD – изменения и удаления. При обращении к полям псевдозаписи в теле триггера всегда записывается двоеточие – :OLD.JOB_ID. Псевдозаписи NEW, OLD в триггере никогда не объявляются, но название псевдозаписи можно изменять при создании триггера, например:

```
CREATE OR REPLACE TRIGGER update_job_history
BEFORE INSERT OR UPDATE
OF job_id, department_id
ON employees
REFERENCING NEW AS NEW1 OLD AS OLD1
FOR EACH ROW
BEGIN
:new1.department_id:=100;
END;
/
```

Триггеры принимают участие в транзакциях. Если в триггере инициируется исключение, то будет выполнен откат соответствующей транзакции. Если триггер сам изменяет данные, то все изменения становятся частью общей транзакции. В триггере нельзя применять операторы управления транзакциями COMMIT, ROOLBACK и SAVEPOINT.

Общий синтаксис создания триггеров:

```
CREATE [OR REPLACE] TRIGGER имя_триггера
{BEFOR |AFTER |
INSERT | DELETE | UPDATE | UPDATE OF COLUMN LIST} ON имя_таблицы
[FOR EACH ROW]
[WHEN () ]
[DECLARE ...]
BEGIN
  ... исполняемые_операторы
  [EXCEPTION
... обработка_исключений]
END [имя_триггера];
```

Триггер, заданный для операции UPDATE, может быть определен для всей таблицы или только для определенных полей –UPDATE OF COLUMN LIST.

Определение WHEN позволяет задать дополнительную логику для исключения ненужных запусков триггера. Предложение WHEN может использоваться только в триггерах уровня записи.

Раздел DECLARE не обязательный, если в триггере не объявляются локальные переменные.

Триггер, использующий последовательность в качестве данных для вставки в ключевое поле таблицы:

```
CREATE OR REPLACE TRIGGER EMPLOYEES_INS
BEFORE INSERT ON employees
FOR EACH ROW
BEGIN
  /* присваиваем код */
  SELECT employees_seq.NEXTVAL INTO :new.employee_id FROM dual;
END employees_INS;
/
```

Триггер EMPLOYEES_INS запускается на событие INSERT для каждой строки. В нем автоматически присваивается значение полю employee_id на основании созданной ранее последовательности employees_seq.

Триггер, изменяющий формат данных при вводе в таблицу:

```
CREATE OR REPLACE TRIGGER EMPLOYEES_UPD
BEFORE INSERT OR UPDATE
ON employees
FOR EACH ROW
BEGIN
-- приводим поля first_name и last_name к нужному формату
:new.first_name:=
UPPER(substr(:new.first_name,1,1))||LOWER(substr(:new.first_name,2));
:new.last_name:=
UPPER(substr(:new.last_name,1,1))||LOWER(substr(:new.last_name,2));
:new.email:=UPPER(:new.email);
END EMPLOYEES_UPD;
/
```

Триггер EMPLOYEES_UPD обрабатывает два события INSERT и UPDATE и приводит к нужному формату поля с фамилией и адресом.

Рассмотрим триггер, который выполняет функцию аудита. Для регистрации всех выполненных операций необходимо создать таблицу useraudit:

```
CREATE TABLE useraudit (
table_name VARCHAR2(30),
oper_name CHAR(1),
pk_key VARCHAR2(200),
column_name VARCHAR2(30),
old_value VARCHAR2(200),
new_value VARCHAR2(200),
```

```
username VARCHAR2(20),
dateoper DATE);
```

Также создадим процедуру, которая будет сохранять данные:

```
CREATE OR REPLACE PROCEDURE setaudit (
vtable_name IN VARCHAR2,
voper_name IN CHAR,
vpk_key IN VARCHAR2,
vcolumn_name IN VARCHAR2,
vold_value IN VARCHAR2,
vnew_value IN VARCHAR2
)
IS
-- PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
  IF vold_value<>vnew_value OR vOper_name IN ('I','D')
  THEN
    INSERT INTO useraudit (table_name, oper_name, pk_key, column_name, old_value, new_value, username, dateoper)
     VALUES (vtable_name, voper_name, vpk_key, vcolumn_name, vold_value, vnew_value, USER, SYSDATE);
-- COMMIT;
END IF;
END;
/
```

В этой процедуре COMMIT закомментирован, так как планируется вызывать ее из триггеров, где использование COMMIT запрещено. Однако, если в процедуре добавить указание на использование автономной транзакции: PRAGMA AUTONOMOUS_TRANSACTION, то в этом случае наличие оператора завершения транзакции обязательно. Если процедуру setaudit объявить, как работающую в автономной транзакции, то она будет регистрировать не только ФАКТ изменений в БД, но и даже ПОПЫТКУ изменения данных, даже если основная транзакция не будет зафиксирована и завершится откатом.

Триггер, который будет запускаться на события изменения данных, имеет следующий синтаксис:

```
CREATE OR REPLACE TRIGGER employees_aud
AFTER INSERT OR UPDATE OR DELETE ON employees
FOR EACH ROW
DECLARE
op   CHAR (1) := 'I';
BEGIN
```

```
  CASE
    WHEN INSERTING THEN
op:= 'I';
setaudit ('EMPLOYEES', op, :NEW.employee_id, 'SALARY', NULL,
:NEW.salary);
setaudit ('EMPLOYEES', op, :NEW.employee_id, 'JOB_ID', NULL,
:NEW.job_id);
setaudit ('EMPLOYEES', op, :NEW.employee_id, 'DEPARTMENT_ID',
NULL, :NEW.department_id);
 WHEN UPDATING('SALARY') OR UPDATING('JOB_ID') OR
 UPDATING('DEPARTMENT_ID') THEN
op:='U';
setaudit ('EMPLOYEES', op, :NEW.employee_id, 'SALARY', :OLD.salary,
:NEW.salary);
setaudit ('EMPLOYEES', op, :NEW.employee_id, 'JOB_ID', :OLD.job_id,
:NEW.job_id);
setaudit ('EMPLOYEES', op, :NEW.employee_id, 'DEPARTMENT_ID',
:OLD.department_id, :NEW.department_id);
    WHEN DELETING THEN
op:='D';
setaudit ('EMPLOYEES', op, :old.employee_id, 'SALARY', :OLD.salary,
            NULL);
setaudit ('EMPLOYEES', op, :old.employee_id, 'JOB_ID', :OLD.job_id,
            NULL);
setaudit ('EMPLOYEES', op, :old.employee_id, 'DEPARTMENT_ID',
:OLD.department_id, NULL);
  ELSE
null;
  END CASE;
END employees_aud;
/
```

Как видно из примера, в Oracle имеется набор функций – предикатов триггера, который позволяет определить, какая операция вызвала триггер. Это предикаты INSERTING, UPDATING, DELETING. Функции возвращают TRUE, если триггер был запущен в ответ на соответствующую операцию. Функция UPDATING имеет перегруженную версию, принимающую в качестве аргумента имя конкретного столбца. Для проверки работы триггера необходимо вставить запись с новым сотрудником, а затем просмотреть таблицу аудита:

```
INSERT INTO employees (first_name, last_name, email, phone_number,
hire_date, job_id, salary, commission_pct, manager_id, department_id)
VALUES ('Иван', 'Иванов', 'ivan', '222.7777', to_date('01.11.2004','dd.mm.yyyy'),
'IT_PROG', 6500, 0, 103, 60);
```

```
SELECT * FROM useraudit;
```

Ошибки при изменении таблицы. Довольно часто при разработке триггеров на изменение данных в таблице приходится сталкиваться с проблемой «мутации» триггеров, а именно с ошибкой Oracle:

```
ORA-04091: table … is mutating. trigger/function may not see it.
```

Эта ошибка возникнет, например, при разработке триггера на обновление таблицы сотрудников, при помощи которой проводится проверка величины заработной платы, чтобы она не превышала среднюю по отделу более чем на 20 %.

Таким образом, мутирующая таблица – это либо таблица, обновляемая в данный момент командами UPDATE, DELETE, INSERT, либо таблица, обновление которой может потребоваться вследствие срабатывания каскадного удаления в подчиненной таблице (ограничение DELETE CASCADE). Для триггеров, принадлежащих изменяемой таблице, эта таблица является мутирующей (к операторным триггерам это понятие не применяется). Кроме того, мутирующей таблицей является любая другая, ссылающаяся на таблицу триггера через ограничение внешнего ключа, что позволяет СУБД запретить чтение несогласованного набора данных.

```
CREATE OR REPLACE TRIGGER employees_sal
  BEFORE UPDATE OF salary
  ON employees
  FOR EACH ROW
DECLARE
avr_sal NUMBER;
-- PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
SELECT avg(salary) INTO avr_sal FROM employees
WHERE department_id = :new.department_id;
--COMMIT;
IF  avr_sal < :new.salary*1.2 THEN
RAISE_APPLICATION_ERROR (-20001, 'заработная плата превышает среднюю по отделу на 20 %');
END IF;
END;
/
```

При попытке обновить заработную плату, например:

```
UPDATE employees SET salary=9000 WHERE employee_id=109;
```

будет получено сообщение об ошибке, однако есть несколько простых приемов, которые могут решить данную проблему, а именно:

1. Если триггер выполняется как автономная транзакция, т. е. объявлен с инструкцией PRAGMA AUTONOMOUS_TRANSACTION и в нем выполняется инструкция COMMIT, тогда в нем можно запрашивать содержимое таблицы, но не изменять его. Таким образом, убрав комментарии в последнем примере, можно достаточно просто решить проблему мутации.

2. Триггер уровня строки не может считывать или записывать данные из таблицы, с которой он связан, но данное ограничение не касается триггеров уровня инструкции, что дает возможность провести необходимые действия.

В более сложных ситуациях, когда невозможно решить проблему указанными способами, используют или комбинацию переменных пакета с двумя триггерами, или составные COMPOUND-триггеры. В первом случае необходимо на уровне пакета объявить глобальную переменную типа ассоциативного массива, в которой можно будет сохранить необходимые данные из мутирующей таблицы с помощью операторного триггера, а затем использовать данные, записанные в пакетной переменной с помощью строчного триггера.

Второй способ предполагает знание синтаксиса COMPOUND-триггера. Появившиеся в версии 11g, эти триггеры имеют в одном блоке обработку всех видов DML-триггеров, могут содержать переменные, которые живут на всем протяжении выполнения оператора, вызвавшего срабатывание триггера, и включают следующие секции:

BEFORE STATEMENT;
AFTER STATEMENT;
BEFORE EACH ROW;
AFTER EACH ROW.

Некоторые правила написания:

– нет отдельной секции инициализации, но для этих целей можно использовать секцию BEFORE STATEMENT;

– нельзя обращаться к псевдозаписям OLD, NEW в секциях уровня оператора (BEFORE STATEMENT и AFTER STATEMENT);

– изменять значения полей псевдозаписи NEW можно только в секции BEFORE EACH ROW;

– исключения, сгенерированные в одной секции, нельзя обрабатывать в другой секции;

– если используется оператор GOTO, он должен указывать на код в той же секции.

Составной триггер можно использовать для написания кода, соответствующего разным моментам времени события и совместного использования переменных, объявленных в секции DECLARE; для отбора строк, предназначенных для загрузки в другую таблиц, – в секции BEFORE EACH ROW или AFTER

EACH ROW, которые затем вставляются в секцию AFTER STATEMENT; для более простого решения проблемы мутирующей таблицы.

Общий синтаксис COMPOUND-триггера:

```
CREATE OR REPLACE TRIGGER имя_триггера
FOR {INSERT|UPDATE|DELETE}[OF имя_столбца, …] ON имя_таблицы
COMPOUND TRIGGER
[секция_объявления_переменных_триггера]
BEFORE STATEMENT
секция_операторного_триггера
BEFORE EACH ROW
секция_строчного_триггера
AFTER EACH ROW
секция_строчного_триггера
AFTER STATEMENT
секция_операторного_триггера
END;
/
```

Рассмотрим пример COMPOUND-триггера:

```
CREATE OR REPLACE TRIGGER sbiu_employees
FOR INSERT OR UPDATE OF salary, job_id ON employees
COMPOUND TRIGGER
TYPE sal_rec_type IS RECORD (min_sal employees.salary%TYPE, max_sal employees.salary%TYPE);
TYPE sal_tab_type IS TABLE of sal_rec_type INDEX BY varchar2(30);
sal_tab sal_tab_type;

BEFORE STATEMENT IS
BEGIN
sal_tab.delete;
FOR rec IN (SELECT job_id, min(salary) min_sal, max(salary) max_sal FROM employees GROUP BY job_id)
LOOP
sal_tab(rec.job_id).min_sal:= rec.min_sal;
sal_tab(rec.job_id).max_sal:= rec.max_sal;
END LOOP;
END BEFORE STATEMENT;

BEFORE EACH ROW IS
BEGIN
IF :NEW.salary NOT BETWEEN sal_tab(:new.job_id).min_sal AND sal_tab(:new.job_id).max_sal
```

```
THEN RAISE_APPLICATION_ERROR (-20505, 'out of range');
END IF;
END BEFORE EACH ROW;
END;
/
```

Триггеры DDL. Триггеры DDL запускаются на события, изменяющие объекты базы данных: таблицы, индексы, триггеры. Синтаксис триггеров следующий:

```
CREATE [OR REPLACE] TRIGGER имя_триггера
{BEFORE | AFTER} {событие_DDL} ON {DATABASE | SCHEMA}
DECLARE
объявление_переменных
BEGIN
 ... код триггера
END;
```

Список событий, на которые реагируют DDL-триггеры, представлен в табл. 2.

Таблица 2

Список событий DDL-триггеров

Событие	Описание
CREATE	Создание объекта
ALTER	Изменение объекта
DROP	Удаление объекта
ANALYZE	Сбор статистики по объекту
ASSOCIATE STATISTICS	Связывание статистики с объектом
AUDIT	Включение средств аудита
NOAUDIT	Отключение средств аудита
COMMENT ON	Создание комментария на объект
DDL	Наступление любого из DDL-событий
DIASSOCIATE STATISTICS	Удаление статистики
GRANT	Назначение прав пользователю
RENAME	Переименование объекта
REVOKE	Отмена прав пользователю
TRUNCATE	Очистка таблицы

Применение триггеров позволяет отслеживать, фиксировать, проверять, запрещать действия с объектами всей базы данных или схемы пользователя. Например:

```
CREATE OR REPLACE TRIGGER drop_trigger
```

```
BEFORE DROP ON SCHEMA
BEGIN
RAISE_APPLICATION_ERROR (num => -20000,
msg => 'Cannot drop object');
END;
/
```

Триггеры событий базы данных. Триггеры могут отслеживать и обрабатывать события базы данных. Триггер имеет следующий синтаксис:

```
CREATE OR REPLACE TRIGGER имя_триггера
{BEFORE | AFTER}
{SERVERERROR | LOGON | LOGOFF | STARTUP | SHUTDOWN |
SUSPEND }ON { DATABASE |SCHEMA }
DECLARE
Объявление_переменных
BEGIN
 . . . код_триггера
END;
/
```

Список событий, на которые реагируют триггеры уровня БД, приведен в табл. 3.

Таблица 3

Список событий триггера уровня БД

Событие	Описание
SERVERERROR	Определяет триггер, срабатывающий в момент, когда Oracle генерирует какую-либо ошибку. Исключения составляют следующие сообщения: ORA-01403: data not found ORA-01422: exact fetch returns more than requested number of rows; ORA-01423: error encountered while checking for extra rows in exact fetch;ORA-01034: ORACLE not available; ORA-04030: out of process memory
LOGON	Определяет триггер на событие LOGON – начало сеанса
LOGOFF	Определяет триггер на событие LOGOFF – закрытия сессии клиента
STARTUP	Определяет триггер на событие STARTUP – старт базы данных
SHUTDOWN	Определяет триггер на событие SHUTDOWN – закрытие базы данных
SUSPEND	Определяет триггер на событие «приостановление инструкции» (появился в версии Oracle 9i)

Необходимо отметить, что системные триггеры уровня БД (ON DATABASE) запускаются только в SQL*Plus из-под учетной записи администратора (as sysdba).

Создадим таблицу и затем системный триггер:

```
CREATE TABLE SYSTEM.AUDITBASE
( nzap NUMBER,
  polz VARCHAR2(20),
  tmin TIMESTAMP,
  oper VARCHAR2(50)
);

CREATE OR REPLACE TRIGGER FIXUSERIN
  AFTER LOGON ON SCHEMA
BEGIN
INSERT INTO SYSTEM.AUDITBASE (NZAP, POLZ, TMIN, OPER)
VALUES (1, USER, SYSDATE, 'User Is Log (off)');
END FIXUSERIN;
/
```

Замещающие триггеры INSTEAD OF создаются для необновляемых представлений (view) и служат для замещения DML-операций своим функционалом; позволяют производить операции вставки/обновления или удаления для необновляемых представлений. Это всегда триггер уровня записи (row level), который имеет доступ к псевдозаписям OLD и NEW, но не может изменять их.

Например, создадим необновляемое представление:

```
CREATE OR REPLACE VIEW my_mgr_view AS
SELECT ( d.department_id || ' ' || d.department_name) "Department",
d.manager_id, e.first_name, e.last_name, e.email, e.hire_date "Hired On",
e.phone_number, e.salary, e.commission_pct,
(e.job_id || ' ' || j.job_title) "Job Class"
FROM departments d
JOIN employees e ON d.manager_id = e.employee_id
JOIN jobs j ON e.job_id = j.job_id
ORDER BY d.department_id;
```

Для того чтобы иметь возможность обновить его, создадим триггер INSTEAD OF:

```
CREATE OR REPLACE TRIGGER update_my_mgr_view
INSTEAD OF UPDATE ON my_mgr_view
FOR EACH ROW
BEGIN
```

```
UPDATE employees SET
last_name = :NEW.last_name,
first_name = :NEW.first_name,
email = :NEW.email,
phone_number = :NEW.phone_number,
salary = :NEW.salary,
commission_pct = :NEW.commission_pct
WHERE employee_id = :OLD.manager_id;
END;
/
```

Порядок активизации триггеров. Если у таблицы имеется несколько типов триггеров, то они активизируются по следующей схеме:

1) выполняется операторный триггер BEFORE (если их несколько, то ничего о порядке их выполнения сказать нельзя);

2) выполняется строковый триггер BEFORE;

3) выполняется активизирующий триггер DML-оператор с последующей проверкой всех ограничений целостности данных;

4) выполняется строковый триггер AFTER с последующей проверкой всех ограничений целостности данных;

5) выполняется операторный триггер AFTER.

Включение/выключение триггеров. Хранящийся в базе данных триггер можно временно отключить, не удаляя его из базы данных. Для этого используется следующая команда:

```
ALTER TRIGGER имя_триггера DISABLE;
```

Включить триггер через некоторый промежуток времени можно, используя команду

```
ALTER TRIGGER имя_триггера ENABLE;
```

Запретить или разрешить запуск всех триггеров, связанных с некоторой таблицей, можно с помощью команды

```
ALTER TABLE имя_таблицы {DISABLE | ENABLE} ALL TRIGGERS;
```

Удаление триггеров. Удаление триггера из базы данных осуществляется с помощью команды

```
DROP TRIGGER имя_триггера;
```

Информацию о триггерах можно получить из представления словаря данных USER_TRIGGERS.

Планировщик задач. Хорошей альтернативой триггерам, позволяющей автоматизировать задания внутри БД Oracle, является Планировщик (Scheduler**)** – специальная программа, которая запускается в определенное время. В СУБД Oracle он реализован с помощью пакета DBMS_SCHEDULER. Пакет DBMS_SCHEDULER позволяет запускать хранимую процедуру или анонимный блок PL/SQL в моменты времени, вычисляемые по указанной пользователем формуле. Он использует следующие основные понятия:

1) Schedule (расписание);
2) Program (программа);
3) Job (плановое задание).

Планировщик можно настроить на однократный запуск (непосредственно после запуска или в определенное время) и на многократный запуск (запуск по расписанию).

Прежде всего для написания планировщика нужно, как правило, дать права на его создание командой: GRANT CREATE JOB TO имя_пользователя;

Рассмотрим пример планировщика для однократного запуска анонимного блока:

```
BEGIN
DBMS_SCHEDULER.CREATE_JOB
( job_name => 'simple_job',
 job_type => 'PLSQL_BLOCK',
 job_action => 'UPDATE employees SET salary=salary+1;',
 enabled => TRUE);
END;
/
```

Здесь параметры: job_name – имя планировщика; job_type – тип запускаемого задания, который может принимать следующие значения: 'STORED_PROCEDURE' – хранимая процедура, 'PLSQL_BLOCK' – анонимный блок; job_action – то, что запускается; enabled – определяет включен или выключен планировщик.

В рассмотренном примере заработная плата каждому сотруднику увеличится на один и планировщик удалится автоматически, что равносильно выполнению обычной команды UPDATE.

Для запуска планировщика однократно в определенное время используется следующий параметр пакета:

```
start_date => SYSTIMESTAMP + INTERVAL '10' SECOND,
```

где start_date – это время запуска, которое в данном примере составляет 10 с от текущего времени.

Для многократного запуска введен параметр repeat_interval, пример использования которого представлен далее:

repeat_interval => 'FREQ = MONTHLY; BYDAY = SUN, -1 SAT',

В данном примере задание будет исполняться ежемесячно по воскресениям и последним субботам месяца. То есть в планировщике есть возможность указывать частоту, интервал и спецификатор запуска задания, например:

FREQ = HOURLY; INTERVAL = 4 – запуск каждые 4 часа;
FREQ = HOURLY; INTERVAL = 4; BYMINUTE = 10; BYSECOND = 30 – запуск каждые 4 часа на 10-й минуте, 30-й секунде;
FREQ = YEARLY; BYYEARDAY = 276 – запуск каждый 276-й день года;
FREQ = YEARLY; BYMONTH = MAR; BYMONTHDAY = 31 – запуск каждое 31-е марта.

Частота – это обязательный компонент календарного выражения, который идентифицируется ключевым словом FREQ. Возможные значения – YEARLY, MONTHLY, DAILY, HOURLY, MINUTELY и SECONDLY.

Интервал повторения идентифицируется ключевым словом INTERVAL и указывает, насколько часто база данных должна повторять задание.

Спецификаторы предоставляют детальную информацию о том, когда должно запускаться задание. Возможные значения – BYMONTH, BYWEEKNO, BYYEARDAY, BYMONTHDAY, BYDAY, BYHOUR, BYMINUTE и BYSECOND. Спецификаторы являются необязательными.

В анонимном блоке можно использовать команды:

```
--отключить планировщик
DBMS_SCHEDULER.DISABLE('simple_job');

--остановить выполняющиеся задания:
DBMS_SCHEDULER.STOP_JOB('simple_job');

--запустить задания
DBMS_SCHEDULER.RUN_JOB('simple_job');

--запустить задания в фоновом режиме
DBMS_SCHEDULER.RUN_JOB('simple_job',
use_current_session => false);

--удалить задания
DBMS_SCHEDULER.DROP_JOB('simple_job');
или
DBMS_SCHEDULER.DROP_JOB ('simple_job', force => TRUE);
```

Также можно изменить параметры планировщика, выполняя последовательно приведенные далее действия:

--отключить планировщик

```
BEGIN
DBMS_SCHEDULER.DISABLE ('simple_job', TRUE);
END;
/

--изменить атрибуты
BEGIN
DBMS_SCHEDULER.SET_ATTRIBUTE (
name => 'simple_job',
attribute => 'repeat_interval',
value => 'FREQ = YEARLY; BYMONTH = MAR; BYMONTHDAY = 31');
DBMS_SCHEDULER.SET_ATTRIBUTE (
name => 'simple_job',
attribute => 'STOP_ON_WINDOW_CLOSE', value => TRUE);
END;
/

--включить планировщик
BEGIN
DBMS_SCHEDULER.ENABLE ('simple_job');
END;
/
```

Информацию об уже созданных планировщиках можно просмотреть в таблице словаря данных USER_SCHEDULER_JOBS.

Кроме того, DBMS_SCHEDULER позволяет скомпоновать задание из независимых элементов: программы и расписания. Оба эти элемента самостоятельны и их можно комбинировать в разных заданиях и изменять, не внося изменений в определения заданий.

Пример создания программы:

```
BEGIN
DBMS_SCHEDULER.CREATE_PROGRAM
(program_name   => 'simple_program',
program_type     => 'STORED_PROCEDURE',
program_action   => 'updatesal',
enabled             => TRUE);
END;
/
```

Список сведений об имеющихся программах для планировщика указан в таблице словаря USER_SCHEDULER_PROGRAMS.

Другими значениями параметра PROGRAM_TYPE могут быть 'PLSQL_BLOCK' и 'EXECUTABLE'.

При наличии у процедуры параметров их количество требуется указать особо:

```
CREATE PROCEDURE salary1 (deer NUMBER ) AS
BEGIN
UPDATE employees SET salary = salary – deer;
END;
/

BEGIN
DBMS_SCHEDULER.CREATE_PROGRAM
( program_name     => 'simple_program1',
 program_type      => 'STORED_PROCEDURE',
 program_action     => 'salary1',
 enabled           => FALSE,
 number_of_arguments => 1);
END;
/
```

Необходимо обратить внимание, что программа создана «отключенной», потому что указать фактические значения параметров во «включенной» программе нельзя, так что в дальнейшем последовательность действий будет следующая:

```
BEGIN
DBMS_SCHEDULER.DEFINE_PROGRAM_ARGUMENT
( program_name      => 'simple_program1',
 argument_position => 1,
 argument_name     => 'DELTA',
 argument_type     => 'NUMBER');
END;
/

BEGIN
DBMS_SCHEDULER.ENABLE ('simple_program1');
END;
```

Пример создания расписания:

```
BEGIN
DBMS_SCHEDULER.CREATE_SCHEDULE
(schedule_name  => 'simple_schedule',
start_date       => SYSTIMESTAMP,
repeat_interval  => 'FREQ=WEEKLY; BYDAY=MON, TUE, WED, THU, FRI',
```

```
end_date        => SYSTIMESTAMP + INTERVAL '1' MONTH);
END;
/
```

Список сведений об имеющихся расписаниях для планировщика находится в таблице словаря USER_SCHEDULER_SCHEDULES.

В общем случае синтаксис указания графика для расписания (параметр REPEAT_INTERVAL) допускает ссылаться на ранее созданные таким же образом расписания.

Из самостоятельно существующих программы и расписания можно составить скомпонованное задание, например:

```
BEGIN
DBMS_SCHEDULER.CREATE_JOB ( job_name => 'compound_job',
program_name  => 'simple_program',
schedule_name => 'simple_schedule',
enabled =>TRUE);
END;
/
```

Задание к лабораторной работе

1. Написать DML-триггер, регистрирующий изменение данных (вставку, обновление, удаление) в одной из таблиц БД. Во вспомогательную таблицу LOG1 записывать, кто, когда (дата и время) и какое именно изменение произвел, для одного из столбцов сохранять старые и новые значения.

2. Написать DDL-триггер, протоколирующий действия пользователей по созданию, изменению и удалению таблиц в схеме во вспомогательную таблицу LOG2 в определенное время и запрещающий эти действия в другое время.

3. Написать системный триггер, добавляющий запись во вспомогательную таблицу LOG3, когда пользователь подключается или отключается. В таблицу логов записывается имя пользователя (USER), тип активности (LOGON или LOGOFF), дата (SYSDATE), количество записей в основной таблице БД.

4. Написать триггеры, реализующие бизнес-логику (ограничения) в заданной вариантом предметной области. Три задания приведены в прил. 6. Количество и тип триггеров (строковый или операторный, выполняется AFTER или BEFORE) определять самостоятельно исходя из сути заданий и имеющейся схемы БД; учесть, что в некоторых вариантах первые два задания могут быть выполнены в рамках одного триггера, а также возможно возникновение мутации, что приведет к совмещению данного пункта лабораторной работы со следующим. Третий пункт задания предполагает использование планировщика задач, который обязательно должен быть настроен на многократный запуск с использованием частоты, интервала и спецификаторов.

5. Самостоятельно или при помощи преподавателя составить задание на триггер, который будет вызывать мутацию таблиц, и решить эту проблему одним из двух способов (при помощи переменных пакета и двух триггеров или при помощи COMPAUND-триггера).

6. Написать триггер INSTEAD OF для работы с необновляемым представлением, созданным после выполнения п. 2 задания к лабораторной работе №3, проверить DML-командами возможность обновления представления до и после включения триггера.

Контрольные вопросы

1. Чем отличаются триггеры от хранимых процедур и функций?
2. Какие типы триггеров существуют?
3. В чем различия строкового и операторного триггеров?
4. Какие псевдозаписи и для чего используются в триггерах?
5. Какие триггеры называют системными?
6. На какие события срабатывают DDL-триггеры?
7. Что такое COMPOUND-триггер и в чем особенность его работы?
8. Для чего используются триггеры INSTEAD OF?
9. От чего зависит последовательность выполнения триггеров?
10. Какие атрибуты есть у триггеров и для чего они применяются?
11. Как можно приостановить работу триггера?
12. Где и как можно посмотреть информацию о триггерах для определенной таблицы?
13. Какие возможности предоставляет пакет DBMS_SCHEDULER?
14. Какими значениями можно определить частоту и интервал запуска планировщика?
15. Что такое расписание и как оно создается?

ЛАБОРАТОРНАЯ РАБОТА №6

ДИНАМИЧЕСКИЙ SQL

Цель работы – изучить возможности пакета DBMS_SQL и встроенного динамического SQL для написания процедур динамической обработки SQL-операторов и блоков PL/SQL, а также научиться применять и управлять такими программными конструкциями, как коллекции.

Теоретические сведения

Коллекции в PL/SQL. Коллекцией в PL/SQL называется упорядоченная группа элементов любого типа, фактически коллекция представляет собой одномерный массив, элементы которого могут иметь как простой, стандартный тип (VARCHAR, NUMBER и т. д.), так и составные типы RECORD, OBJECT.

СУБД Oracle поддерживает три типа коллекций, каждый из которых имеет свои особенности.

1. Ассоциативный массив (таблицы PL/SQL или INDEX BY) – это неограниченные разреженные массивы, состоящие из однородных элементов и индексируемые по номеру элемента. Начиная с версий 9i, допускается индексирование ассоциативных массивов посредством значений с типами VARCHAR2 или PLS_INTEGER. Данный вид коллекций нельзя использовать в качестве типа данных столбца таблицы.

2. Вложенные таблицы – NESTED TABLES – одномерные массивы, состоящие из однородных элементов. Вложенные таблицы можно использовать как в программах PL/SQL, так и в базе данных.

3. Массивы VARRAY – одномерные массивы, состоящие из однородных элементов, размер которых ограничен, и они не могут быть разреженными. Массивы VARRAY можно использовать в качестве типа данных для столбцов таблиц.

Объявление коллекций. Объявление типов коллекций выполняется аналогично объявлению типа данных оператором TYPE.

Объявление ассоциативного массива:

```
TYPE имя_типа_таблицы IS TABLE OF тип_данных [NOT NULL]
INDEXBY [BINARY_INTEGER |
подтип_типа BINARY_INTEGER | VARCHAR2 (максимальный_индекс)];
```

Объявление вложенной таблицы:

```
/* так объявляются типы для использования в базе данных */
```

CREATE [OR REPLACE] TYPE имя_типа AS | IS
TABLE OF тип_данных_элемента [NOTNULL];

Для использования в программах PL/SQL можно не регистрировать в базе данных тип, а объявлять его непосредственно в программах PL/SQL. В этом случае слово CREATE опускается.

Объявление массива VARRAY:

CREATE [OR REPLACE] TYPE имя_типа IS
VARRAY (максимальный_индекс) OF тип_данных_элемента [NOT NULL];

Объявление переменных с типом коллекция выполняется аналогично объявлению любой другой переменной:

имя_переменной тип_переменной [NOT NULL] [DEFAULT | := значение_по_умолчанию];

Операции с коллекциями. Для оперирования коллекциями имеются встроенные функции, которые называются методами коллекций (табл. 4).

Таблица 4

Методы коллекций

Метод	Описание	Ассоциативный массив	Вложенная таблица	Массив переменной длины
COUNT	Функция, возвращающая количество элементов в коллекции	+	+	+
DELETE	Процедура, удаляющая элементы коллекции: DELETE(i) – один элемент; DELETE(i,j) – с i-го по j-й элементы; DELETE – все элементы коллекции	+	+	+
EXISTS	Проверяет наличие заданного элемента в коллекции	+	+	+
EXTEND	Добавляет элементы в коллекцию	–	+	+
FIRST, LAST	Возвращают наименьший и наибольший индексы коллекций	+	+	+
LIMIT	Возвращает максимальное количество элементов	–	–	+
PRIOR, NEXT	Возвращают индекс предыдущего и следующего элемента коллекции	+	+	+
TRIM	Удаляет n последних элементов коллекции	–	+	+

Для начала работы необходимо коллекцию инициализировать. Это можно сделать: 1) явным образом, используя конструктор; 2) путем выборки элементов из базы данных в коллекцию; 3) присваивая ей содержимое другой переменной-коллекции.

Для примера определим следующие типы данных:

```
/*тип для хранения информации о присутствии сотрудника на работе */

CREATE OR REPLACE TYPE maska_type IS VARRAY (31) OF NUMBER;
```

Инициализировать коллекцию с типом maska_type можно следующим образом:

```
DECLARE
 -- инициализация maska1 с помощью конструктора
maska1 maska_type := maska_type(1,1,0,0,1,1,1,1,1,1,1,0,1,1);
maska2 maska_type; -- пока коллекция не инициализирована
 maska3 maska_type; -- пока коллекция не инициализирована
BEGIN
 -- инициализация maska2 присвоением
 maska2:=maska1; -- инициализация maska1
 -- инициализация maska3 путем выборки элементов из базы данных
SELECT CAST (MULTISET (SELECT rownum FROM all_tables WHERE rownum <=31) as maska_type) INTO maska3 FROM DUAL;
END;
/
```

Присваивание значений элементам коллекции производится стандартным оператором присваивания. Ассоциативные массивы позволяют присваивать значения произвольным элементам коллекций. В отличие от них массивы VARRAY и вложенные таблицы индексируются автоматически PL/SQL. При этом прежде чем присвоить значение строке вложенной таблицы или VARRAY, необходимо расширить коллекцию при помощи оператора EXTEND.

```
/*объект для хранения данных о персоне*/

CREATE OR REPLACE TYPE inf_type AS OBJECT (
first_name   VARCHAR2 (20),    dater DATE);
/

/*вложенная таблица для хранения информации о детях сотрудников */

CREATE OR REPLACE TYPE lst_inf_type AS TABLE OF inf_type;
/
```

```
DECLARE
 lst_child lst_inf_type := lst_inf_type();
BEGIN
 DBMS_OUTPUT.PUT_LINE('count in lst_child ='||lst_child.count);
 lst_child.EXTEND;
 lst_child (lst_child.LAST):=
   inf_type ('Николай',TO_DATE('05.06.1996', 'dd.mm.yyyy'));
 lst_child.EXTEND;
 lst_child (lst_child.last):=
  inf_type ('Ольга',TO_DATE('28.11.2000', 'dd.mm.yyyy'));
DBMS_OUTPUT.PUT_LINE ('count in lst_child ='||lst_child.count);
FOR i IN lst_child.FIRST..lst_child.LAST
LOOP
DBMS_OUTPUT.PUT_LINE (i||': '|| lst_child(i).FIRST_NAME||': '||
    lst_child(i).DATER);
END LOOP;
END;
/
```

Псевдофункции коллекций. Псевдофункции коллекций позволяют работать с коллекциями как таблицами и с таблицами как коллекциями. PL/SQL предлагает четыре псевдофункции (табл. 5).

Таблица 5

Псевдофункции коллекций

Функция	Описание
THE	Представляет значение одного столбца таблицы с типом коллекции в виде виртуальной таблицы. На сегодняшний момент считается устаревшей и не рекомендуется к использованию
CAST	Преобразует коллекцию одного типа в коллекцию другого типа, например массив VARRAY во вложенную таблицу
MULTISET	Представляет таблицу базы данных в виде коллекции. Используя функции CAST и MULTISET, можно получить из запроса набор строк и преобразовать его в коллекцию
TABLE	Представляет коллекцию в виде таблицы базы данных, что позволяет выполнять запросы к коллекции как к таблице базы данных

Пример использования функций с коллекциями:

```
DECLARE
lst_child   lst_inf_type := lst_inf_type ();
BEGIN
  DBMS_OUTPUT.put_line ('count in lst_child =' || lst_child.COUNT);
```

```
  lst_child.EXTEND;
  lst_child (lst_child.LAST) :=
          inf_type ('Анна', TO_DATE ('08.09.2003', 'dd.mm.yyyy'));
  lst_child.EXTEND;
  lst_child (lst_child.LAST)    :=
        inf_type ('Николай', TO_DATE ('05.06.1996', 'dd.mm.yyyy'));
  lst_child.EXTEND;
  lst_child (lst_child.LAST)    :=
          inf_type ('Ольга', TO_DATE ('28.11.2000', 'dd.mm.yyyy'));
  DBMS_OUTPUT.put_line ('Список детей ');

  FOR i IN lst_child.FIRST .. lst_child.LAST
  LOOP
    DBMS_OUTPUT.put_line (   i
                          || ':'
                          || lst_child (i).first_name
                          || ':'
                          || TO_CHAR (lst_child (i).dater, 'dd.mm.yyyy'));
  END LOOP;

  DBMS_OUTPUT.put_line ('Дети до пяти лет:');

  FOR cd IN (SELECT   first_name, TO_CHAR (dater, 'dd.mm.yyyy') dater
               FROM TABLE (lst_child)
              WHERE dater >= TO_DATE ('01.01.2000', 'dd.mm.yyyy')
           ORDER BY dater)
  LOOP
    DBMS_OUTPUT.put_line (cd.first_name || ':' || cd.dater);
END LOOP;
END;
/
```

Функция TABLE позволяет использовать мощный инструмент SQL-запросов применительно к коллекциям, что в отличие от последовательной обработки значительно удобнее.

В следующем примере показано, как можно использовать функции CAST, MULTISET для выборки данных из таблиц в коллекции. Здесь используется тот же тип lst_inf_type, для того чтобы работать со списком сотрудников. Заполнив коллекцию из базы данных, на ее основе получим несколько отчетов. Использование коллекции сокращает количество обращений к БД, особенно если не только выдаются отчеты, но и выполняется многократное обновление информации. Например:

```
DECLARE
 lst_empl   lst_inf_type := lst_inf_type ();
n INTEGER;
BEGIN
   -- увеличили размер буфера вывода
dbms_output.enable(100000);

SELECT CAST (MULTISET (SELECT first_name, hire_date
FROM employees ORDER BY first_name) AS lst_inf_type)
  INTO lst_empl FROM dual;

DBMS_OUTPUT.put_line ('Список сотрудников: ');


FOR i IN lst_empl.FIRST .. lst_empl.LAST
  LOOP
    DBMS_OUTPUT.put_line (   i
                          || ':'
                          || RPAD (lst_empl (i).first_name, 20, ' ')
                          || ':'
                          || TO_CHAR (lst_empl (i).dater, 'dd.mm.yyyy'));
END LOOP;

  /* получим список сотрудников, проработавших более десяти лет*/
  n := 0;
  DBMS_OUTPUT.put_line ('Список сотрудников, проработавших пять и
более лет: ');
FOR cd IN (SELECT   first_name, TO_CHAR (lc.dater, 'dd.mm.yyyy') dater
    FROM TABLE (lst_empl) lc
    WHERE TRUNC (MONTHS_BETWEEN (SYSDATE, dater) / 12) >= 5
    ORDER BY lc.dater)
  LOOP
    n    := n + 1;
    DBMS_OUTPUT.put_line (   n
                          || '.'
                          || RPAD (cd.first_name, 20, ' ')
                          || ':'
                          || cd.dater);
  END LOOP;
 DBMS_OUTPUT.put_line ('Список сотрудников, принятых в текущем году');
n := 0;
  FOR cd IN (SELECT   first_name, TO_CHAR (dater, 'dd.mm.yyyy') dater
FROM TABLE (lst_empl)
            WHERE EXTRACT (YEAR FROM SYSDATE) = EXTRACT
(YEAR FROM dater) ORDER BY dater)
```

```
  LOOP
    n := n + 1;
    DBMS_OUTPUT.put_line (n
                          || '.'
                          || RPAD (cd.first_name, 20, ' ')
                          || ':'
                          || cd.dater);
  END LOOP;
END;
/
```

Также можно осуществлять выборку данных из коллекции:

```
CREATE OR REPLACE TYPE a_data IS TABLE OF VARCHAR2(30);
/

DECLARE
        v_data a_data := a_data( 'Dmitry', 'Sergei', 'Yulya' );
BEGIN
-- "column_value" – это имя поля, возвращаемого table(cast(collection))
        FOR c1 IN (
SELECT column_value name FROM TABLE (CAST( v_data AS a_data)) )
        LOOP
        DBMS_OUTPUT.PUT_LINE ('Looking for values: ' || c1.name);
        END LOOP;
END;
/
```

Использование BULK COLLECT. Использование оператора BULK COLLECT может значительно ускорить выборку данных из таблицы в коллекцию, особенно при большом количестве строк. Включив его в явный или неявный курсор, программист указывает ядру SQL на необходимость перед передачей управления PL/SQL выполнить пакетное связывание данных из множества строк с заданными в запросе коллекциями, что уменьшает переключения между ядром SQL и PL/SQL. Так, например, если коллекция имеет 100 элементов, использование BULK COLLECT позволит выполнить операцию, функционально эквивалентную выполнению 100 операторов SELECT, но при этом исключит 99 лишних переключений. Синтаксис этого предложения следующий:

```
BULK COLLECT INTO имя_коллекции [, имя_коллекции] …
```

Пример использования:

```
DECLARE
```

```
  TYPE empl_inf_type is TABLE of EMPLOYEES%ROWTYPE;
  cEmp empl_inf_type;
BEGIN
   SELECT *
   BULK COLLECT into  cEmp
   FROM EMPLOYEES;
DBMS_OUTPUT.PUT_LINE ('Список сотрудников : ');
FOR i IN cEmp.FIRST .. cEmp.LAST
   LOOP
     DBMS_OUTPUT.PUT_LINE (cEmp(i).Employee_id
                    || '.'
                    || RPAD(cEmp(i).first_name,20,' ')
                    || ':'
                    || cEmp(i).Salary);
END LOOP;
END;
/
```

Использование FORALL. Предложение FORALL предназначено для выполнения операторов обновления базы данных: INSERT, UPDATE, DELETE. Синтаксис оператора:

```
FORALL индекс_записи IN начальное_значение ... конечное значение
  Sql_инструкция
```

Рассмотрим пример использования оператора FORALL. Сначала создадим таблицу на основе справочника ALL_TAB_COLUMNS:

```
CREATE TABLE tmp_forall as
SELECT OWNER, table_name, column_name, data_type
FROM ALL_TAB_COLUMNS WHERE rownum=0;
```

Количество записей в ALL_TAB_COLUMNS довольно большое. Сравним производительность СУБД при использовании FORALL и без него:

```
DECLARE
TYPE LstString_type IS TABLE OF VARCHAR2(50);
cOwner LstString_type;
ctable_name LstString_type;
ccolumn_name LstString_type;
cdata_type LstString_type;
t1 Number(15);
t2 Number(15);
t3 Number(15);
```

```
BEGIN
SELECT OWNER, table_name, column_name, data_type
BULK COLLECT INTO cOwner, ctable_name, ccolumn_name, cdata_type
 FROM ALL_TAB_COLUMNS;
  t1 := dbms_utility.get_time;
  DBMS_OUTPUT.PUT_LINE ('количество записей ='||cowner.Count);
  FOR i IN cowner.FIRST .. cowner.LAST
LOOP
  INSERT INTO tmp_forall
  VALUES (cOWNER(i), ctable_name(i), ccolumn_name(i), cdata_type(i));
  END LOOP;
  t2 := dbms_utility.get_time;
  FORALL i IN cowner.FIRST .. cowner.LAST
    INSERT INTO tmp_forall
 VALUES (cOWNER(i), ctable_name(i), ccolumn_name(i), cdata_type(i));
  t3 := dbms_utility.get_time;
  DBMS_OUTPUT.PUT_LINE ('time FOR = '||to_char(t2-t1));
  DBMS_OUTPUT.PUT_LINE ('time FORALL = '||to_char(t3-t2));
END;
/
```

Получили результат:

```
количество записей =9382
time FOR    =174
time FORALL =5
```

Понятие динамического SQL. Все программы, рассматриваемые ранее, являются статическими. Это означает, что структура SQL-операторов известна уже во время компиляции программы (механизм раннего связывания). Динамический SQL позволяет выполнять SQL-операторы, создавая их динамически, во время выполнения программы (механизм позднего связывания), а потом производить их грамматический разбор и обработку. Возможно, программа генерирует запросы по ходу работы на основе введенных пользователем условий; возможно, это специализированная программа загрузки данных. Утилита SQL*Plus – это пример рода программы, схожего с любым другим средством выполнения произвольных запросов или генерации отчетов. Утилита SQL*Plus позволяет выполнить любой SQL-оператор и показать результаты его выполнения, хотя при ее компиляции операторы, которые пишет пользователь, определенно не были известны.

В обычной практике разработки приложений на PL/SQL встречаются ситуации, когда необходимо выполнить в модуле PL/SQL запрос, текст которого будет известен только к моменту его выполнения, не определен на стадии раз-

работки программы и будет меняться во время выполнения. Например, процедура, реализующая выборку данных из таблицы, имя которой задается в качестве параметра процедуры, или процедура, запускающая на выполнение другую процедуру, имя которой храниться в таблице базы данных.

Динамический SQL позволяет сконструировать и хранить код в приложении в виде символьной строки, выполнить команды с различными столбцами или условиями, содержащие и не содержащие связанные переменные, написать и выполнить в коде PL/SQL команды DDL, DCL и команды управления сеансом.

Для обработки таких ситуаций существует несколько механизмов:

– пакет DBMS_SQL;

– внутренний динамический SQL (пакет NDS), основой которого является оператор EXECUTE IMMEDIATE.

И пакет DBMS_SQL, и оператор EXECUTE IMMEDIATE позволяют выполнять не только запросы SELECT, но и анонимные блоки PL/SQL, процедуры, а также операторы DDL, такие как создание, удаление таблиц БД, триггеров и других объектов, что невозможно сделать напрямую, написав команду в блоке PL/SQL.

Пакет DBMS_SQL. Алгоритм выполнения операторов с помощью DBMS_SQL следующий:

1. Преобразование SQL-оператора в строку символов.

2. Грамматический разбор строки символов с помощью DBMS_SQL.PARSE.

3. Привязка всех входных переменных с помощью DBMS_SQL.BIND_VARIABLE.

4. Если выполняемый оператор – это оператор DML (UPDATE, DELETE, INSERT), выполнение его с помощью DBMS_SQL.EXECUTE с последующим считыванием выходных переменных привязки с помощью DBMS_SQL.VARIABLE_VALUE (если нужно).

5. Если оператор является оператором выборки (*select*) – описание выходных переменных с помощью DBMS_SQL.DEFINE_COLUMN.

6. Выполнение запроса на выборку с помощью DBMS_SQL.EXECUTE и выборка результатов при помощи DBMS_SQL.FETCH_ROWS и DBMS_SQL.COLUMN_VALUE.

Обработка операторов DML посредством DBMS_SQL. Для обработки операторов UPDATE, DELETE, INSERT средствами модуля DBMS_SQL необходимо последовательно выполнить следующие действия:

1. Открыть курсор. Осуществляется посредством вызова процедуры OPEN_CURSOR, описание которой в модуле выглядит следующим образом:

```
OPEN_CURSOR return INTEGER;
```

Параметры в данной процедуре отсутствуют.

Каждый вызов возвращает целое число, представляющее собой идентификационный номер курсора. Этот номер используется в последующих вызовах курсора. В границах одного курсора можно по очереди обрабатывать несколько SQL-операторов или выполнять один и тот же оператор несколько раз.

2. Выполнить грамматический разбор оператора. При выполнении грамматического разбора оператор направляется на сервер БД. Сервер проверяет его синтаксис и семантику и возвращает ошибку (устанавливая исключительную ситуацию), если нарушены требования грамматики. Кроме того, во время разбора определяется план выполнения оператора. Осуществляется грамматический разбор посредством вызова процедуры DBMS_SQL.PARSE, описание которой в модуле имеет следующий вид:

```
PROCEDURE PARSE (c in INTEGER, statement in VARCHAR2,
                 language_flag in INTEGER);
```

где c – идентификационный номер курсора, который предварительно должен быть открыт посредством OPEN_CURSOR;

statement – оператор, грамматический разбор которого выполняется;

language_flag – указывает, как трактовать оператор, при этом значение NATIVE – это режим, установленный для той базы данных, с которой выполнено соединение.

3. Выполнить привязку входных переменных. При выполнении этой операции заполнители, указанные в операторе, связываются с фактическими переменными. Имена заполнителей обычно предваряют символом двоеточия. Процедура BIND_VARIABLE выполняет привязку и объявление имен заполнителей. Размер и тип данных фактических переменных также устанавливается BIND_VARIABLE посредством набора переопределенных вызовов:

```
PROCEDURE BIND_VARIABLE (c in INTEGER, name in VARCHAR2,
                         value in  number);
PROCEDURE BIND_VARIABLE (c in INTEGER,  name in VARCHAR2,
                         value in VARCHAR2);
PROCEDURE BIND_VARIABLE (c in INTEGER, namein VARCHAR2,
                         value in VARCHAR2, out_value_size in INTEGER);
```

где name – это имя заполнителя, с которым будет связана переменная;

value – реальные данные, которые будут привязываться (тип и размер этой переменной также считываются); при необходимости данные, содержащиеся в этой переменной, будут преобразованы;

out_value_size – параметр, задаваемый при привязке переменных VARCHAR2 и CHAR; если он указан, то это максимальный ожидаемый размер значения в байтах, если не указан, то используется размер указанный в параметре value.

4. Выполнить оператор посредством функции EXECUTE. Описание ее в модуле выглядит следующим образом:

```
FUNCTION EXECUTE (c in INTEGER) return INTEGER;
```

где c – идентификатор предварительно открытого курсора.

Функция EXECUTE возвращает число отработанных строк (в этом смысле возвращаемое значение аналогично курсорному атрибуту %ROWCOUNT). Следует учесть, что возвращаемое значение не определено для операторов выборки, а также и то, что EXECUTE вызывается из выражений программ.

5. Закрыть курсор. Закрытие курсора осуществляется посредством вызова процедуры CLOSE_CURSOR, описание которой выглядит следующим образом:

```
PROCEDURE CLOSE_CURSOR (c in out INTEGER);
```

Передаваемое процедуре значение должно быть достоверным идентификатором курсора. После вызова фактический параметр устанавливается в NULL, что свидетельствует о закрытии курсора.

Приведем пример динамического использования операторов DML:

```
CREATE OR REPLACE PROCEDURE update_address (p_lname IN staff.lname%TYPE,
        p_fname IN staff.fname%TYPE,
        p_newaddress IN staff.address%TYPE,
        p_rowsupdated OUT INTEGER) IS
v_cursor_id INTEGER;
v_updatestmt VARCHAR2(100);
BEGIN
        v_cursor_id := DBMS_SQL.OPEN_CURSOR;
        v_updatestmt := 'UPDATE staff SET address=:addr
        WHERE fname =: fname AND lname= :lname';
DBMS_SQL.PARSE (v_cursor_id, v_updatestmt, DBMS_SQL.NATIVE);
        DBMS_SQL.BIND_VARIABLE (v_cursor_id, ':addr', p_newaddress);
        DBMS_SQL.BIND_VARIABLE (v_cursor_id, ':fname', p_fname);
        DBMS_SQL.BIND_VARIABLE (v_cursor_id, ':lname', p_lname);
        p_rowsupdated := DBMS_SQL.EXECUTE (v_cursor_id);
        DBMS_SQL.CLOSE_CURSOR (v_cursor_id);
EXCEPTION
        WHEN OTHERS THEN DBMS_SQL.CLOSE_CURSOR(v_cursor_id);
        RAISE;
END update_address;
/
```

Обработка запросов на извлечение информации производится путем последовательного выполнения всех перечисленных далее действий:

1. Открытие курсора (OPEN_CURSOR).
2. Выполнение грамматического разбора (PARSE).
3. Выполнение привязки всех входных переменных (BIND_VARIABLE).
4. Описание элементов списка выбора (DEFINE_COLUMN).
5. Исполнение запроса (EXECUTE).
6. Считывание строк (FETCH).
7. Запись результатов в переменные (COLUMN_VALUE).
8. Закрытие курсора (CLOSE_CURSOR).

В отличие от динамической обработки DML-операторов, обработка инструкций SELECT включает дополнительно описание элементов списка выбора, считывание строк и запись результатов в переменные PL/SQL.

Процесс определения элементов списка выбора напоминает привязку входных переменных, за исключением того, что элементы списка выбора должны быть не привязаны, а только определены. В процедуре DEFINE_COLUMN указываются типы и размер переменных, в которые считываются элементы списка выбора. Каждый элемент при этом преобразуется в тип соответствующей переменной. Описание элементов списка выбора производится посредством процедур DEFINE_COLUMN модуля DBMS_SQL:

```
PROCEDURE DEFINE_COLUMN (c in INTEGER, position in INTEGER,
                         column in number);
PROCEDURE DEFINE_COLUMN (c in INTEGER, position in INTEGER,
                         column in VARCHAR2, column_size in INTEGER);
```

Для переменных *VARCHAR2* нужно обязательно указывать параметр column_size, поскольку система поддержки PL/SQL должна знать максимальный размер этих переменных во время выполнения программы, так как в отличие от типов *number, date* данные этих типов не имеют фиксированной длины, заранее известной компилятору. Описание параметров процедуры приведено в табл. 6.

Таблица 6

Параметры процедуры DEFINE_COLUMN

Параметр	Тип	Назначение
c	INTEGER	Идентификатор курсора
position	INTEGER	Позиция пункта списка выбора
column	NUMBER, VARCHAR2	Переменная, определяющая тип и размер выходной переменной. Имя переменной не играет особой роли, однако тип и размер важны. Однако как в DEFINE_COLUMN, так и в COLUMN_VALUE обычно используются одни и те же переменные

Окончание табл. 6

Параметр	Тип	Назначение
column_size	INTEGER	Максимальный ожидаемый размер данных. Обязателен для тех типов, длина которых неизвестна заранее системе поддержки PL/SQL

После выполнения запроса строки набора необходимо считать в буфер посредством вызова функции FETCH_ROWS. Эта функция описана в модуле DBMS_SQL следующим образом:

```
FUNCTION FETCH_ROWS (cin INTEGER) return INTEGER;
```

FETCH_ROWS возвращает число считываемых строк. FETCH_ROWS и COLUMN_VALUE вызываются в цикле несколько раз до тех пор, пока FETCH_ROWS не возвратит 0.

После успешно выполненного считывания строк производится запись результатов в переменные PL/SQL посредством процедуры COLUMN_VALUE. Если в выборке не были возвращены строки (что указывается возвратом 0), COLUMN_VALUE устанавливает для выходной переменной NULL-значение. Ниже приведено описание этой процедуры в модуле DBMS_SQL, а описание ее параметров – в табл. 7:

```
PROCEDURE COLUMN_VALUE (c in INTEGER, position in INTEGER,
                        value out number);
PROCEDURE COLUMN_VALUE (c in INTEGER, position in INTEGER,
value out number, column_error out number, actual_length out number);
PROCEDURE COLUMN_VALUE (c in INTEGER, position in INTEGER,
                        value out VARCHAR2);
PROCEDURE COLUMN_VALUE (c in INTEGER, positionin INTEGER,
 value out VARCHAR2, column_error out number, actual_length out number);
```

Таблица 7

Параметры процедуры COLUMN_VALUE

Параметр	Тип	Предназначение
c	INTEGER	Идентификатор курсора
position	INTEGER	Относительная позиция в списке выбора. Как и в DEFINE_COLUMN, позиция первого элемента списка =1
value	NUMBER, VARCHAR2	Выходная переменная. Если тип этого параметра отличается от типа, указанного в DEFINE_COLUMN, то возникает ошибка, что соответствует исключительной ситуации DBMS_SQL.INCONSISTENT_TYPES

Окончание табл. 7

Параметр	Тип	Предназначение
column_error	NUMBER	Код ошибки столбца. Выдается в виде отрицательного числа. Ошибка будет устанавливать исключительную ситуацию, а column_error позволяет определить, какой из столбцов стал причиной конкретной ошибки. Если столбец был успешно прочитан, то column_error = 0
actual_length	NUMBER	Если указан, то в данной переменной будет находиться исходный размер столбца (размер столбца перед его считыванием). Это удобно в случае, когда размер переменной недостаточен и значение усекается (это также приводит к ошибке)

Процедура создания таблицы с помощью пакета DBMS_SQL.

```
CREATE OR REPLACE PROCEDURE create_temp_dept (tname IN OUT VARCHAR2) AS cur INTEGER; --хранит идентификатор (ID) курсора
        ret INTEGER; -- хранит возвращаемое по вызову значение
        str VARCHAR2(250); -- хранит команды
BEGIN -- генерация временной таблицы по имени с использованием заранее
-- заданного имени и возврат значения функции
-- DBMS_SESSION.UNIQUE_SESSION_ID
tname:= 'dept ' || dbms_session.unique_session_id;
-- генерация команды CREATE TABLE по заранее заданному тексту
-- и переменной tname
str := 'CREATE TABLE'||' '||tname
|| ' (deptno INTEGER,' || 'dname VARCHAR2(14),' || 'loc VARCHAR2(13))';
-- Динамически формируемое DDL-предложение
cur:= DBMS_SQL.OPEN_CURSOR;
DBMS_SQL.PARSE(cur, str, dbms_sql.v7);
ret:= DBMS_SQL.EXECUTE(cur);
DBMS_SQL.CLOSE_CURSOR(cur);
END;
/
```

Выполним анонимный блок:

```
DECLARE
f varchar2(20);
BEGIN
f:='name_tab';
create_temp_dept (f);
END;
/
```

В результате появится таблица, например, с именем DEPT10145ACDA0002, где следующие за DEPT символы представляют собой уникальный номер сессии пользователя. Чтобы создать таблицу со своим именем ('name_tab'), нужно закомментировать строку tname := 'dept1' || dbms_session.unique_session_id и снова выполнить анонимный блок.

Создадим процедуру print_any_table, которая выводит в виде отчета содержимое таблицы, имя которой передается в качестве параметра.

```
CREATE OR REPLACE PROCEDURE print_any_table (
vowner  IN   VARCHAR2,
vtable_name   IN   VARCHAR2)
IS
sqltext     VARCHAR2 (2000) := 'select ';
x  CHAR (1):= ' ';
table_cursor   NUMBER;
TYPE col_inf_type IS RECORD (name VARCHAR2 (30),
data_type VARCHAR2 (30),value VARCHAR2 (1000));

TYPE list_col_type IS TABLE OF col_inf_type;
s VARCHAR2 (250);
list_col list_col_type := list_col_type ();
cnt INTEGER;
BEGIN
  DBMS_OUTPUT.ENABLE (100000);
  FOR c IN (SELECT column_name, data_type FROM all_tab_columns
          WHERE owner = UPPER (vowner)
          AND table_name = UPPER (vtable_name))
  LOOP
sqltext:=
sqltext || x || 'to_char(' || c.column_name || ') AS ' || c.column_name;
x:= ',';
list_col.EXTEND;
list_col (list_col.LAST).NAME := c.column_name;
list_col (list_col.LAST).NAME := c.data_type;
  END LOOP;
sqltext := sqltext || ' from ' || vowner || '.' || vtable_name;
table_cursor := DBMS_SQL.open_cursor;
DBMS_SQL.parse (table_cursor, sqltext, DBMS_SQL.native);

  FOR i IN list_col.FIRST ..list_col.LAST
  LOOP
DBMS_SQL.DEFINE_COLUMN (table_cursor, i, list_col (i).VALUE, 1000);
  END LOOP;
```

```
cnt := DBMS_SQL.EXECUTE (table_cursor);
DBMS_OUTPUT.put_line ('Таблица'
                      || vtable_name
                      || ' содержит '
                      || cnt
                      || ' строк.');
cnt := 0;
  LOOP
    IF DBMS_SQL.fetch_rows (table_cursor) = 0THEN
EXIT;
    END IF;
cnt := cnt + 1;
s := cnt || '.';
    FOR i IN list_col.FIRST ..list_col.LAST
    LOOP
DBMS_SQL.column_value (table_cursor, i, list_col (i).VALUE);
s := s || ' ' || list_col (i).VALUE;
    END LOOP;
DBMS_OUTPUT.put_line (SUBSTR (s, 1, 255));
  END LOOP;
DBMS_SQL.close_cursor (table_cursor);
END;
/
```

Встроенный SQL (Native dynamic SQL, NDS). Встроенный динамический SQL впервые появился в Oracle 8i и является составной частью самого языка. Вследствие этого он значительно проще в применении и быстрее, чем модуль DBMS_SQL. Он позволяет декларативно выполнять динамический SQL в среде PL/SQL при условии, что структура команды неизвестна только до момента выполнения. Большинство действий можно выполнить с помощью одного оператора – EXECUTE IMMEDIATE, а остальные – с помощью операторов OPEN FOR, FETCH, CLOSE.

При работе со встроенным динамическим SQL для выполнения предложений используется следующая конструкция:

EXECUTE IMMEDIATE *оператор_SQL*
[INTO{ *переменная1* [, *переменная2*] ... | *запись* }]
[USING [IN | OUT | IN OUT] *связанный_аргумент1* [,
[IN | OUT | IN OUT] *связанный_аргумент2*] ...];
[{RETURNING | RETURN} INTO *результат1, . . . ,результатN*]

где INTO – определяет приемник результата;

USING – определяет типы и порядок аргументов, используемых для передачи значений в запрос и приема данных результата. Замена аргументов на их значения происходит позиционно;

оператор_SQL – любой оператор SQL или PL/SQL-блок;

переменная1, переменная2 или запись – переменные PL/SQL, в которые необходимо выбрать данные (столбцы одной строки результатов оператора SELECT);

связываемый_аргумент1, связываемый_аргумент2 – набор переменных PL/SQL, используемых для передачи входных данных/результатов;

результат1,…, результатN – набор PL/SQL-переменных, используемых для размещения результатов, возвращаемых конструкцией RETURN.

Ограничения на передаваемые аргументы:

– Нельзя передавать значения типа boolean, таблицы index-by, записи.

– Нельзя передавать NULL как литерал (только переменную со значением NULL или функцию, которая возвращает NULL).

DML-команда в EXECUTE IMMEDIATE выполняется в контексте текущей транзакции.

DDL-команда в EXECUTE IMMEDIATE выполнит COMMIT всех измененных данных.

Рассмотрим примеры использования, встроенного SQL в процедурах и функциях:

1. Для создания, изменения или удаления таблицы в коде PL/SQL:

```
EXECUTE IMMEDIATE 'DROP TABLE temp_table';
```

2. Для изменения заработной платы сотрудникам определенного отдела, указываемого только при выполнении:

```
EXECUTE IMMEDIATE  'UPDATE emp SET salary=salary*:num WHERE depno=:did' USING v_num,  v_did;
```

3. Для получения информации о количестве записей в неизвестной заранее таблице:

```
EXECUTE IMMEDIATE 'SELECT count(*) FROM ' || v_name INTO v_cnt;
```

4. Для получения информации об определенном сотруднике:

```
EXECUTE IMMEDIATE 'SELECT name, post FROM ' || tabname ||
'WHERE tabno=:id'  INTO  vname, vpost USING  vid;
```

5. Для выполнения запроса, подсчитывающего количество строк в любой таблице базы данных, к которой пользователь имеет доступ:

```
CREATE OR REPLACE
```

```
FUNCTION get_row_cnts(p_tname in VARCHAR2) return NUMBER  AS
l_cnt number;
BEGIN
EXECUTE IMMEDIATE 'SELECT count(*) FROM ' || p_tname INTO l_cnt;
RETURN l_cnt;
END;
 /
```

Для работы с запросами, порождающими множественные результаты, существуют три конструкции:

OPEN{ *курсорная_переменная* | *:хост_переменная*}
FOR *предложение_SQL*
[USING *связанный_аргумент1* [, *связанный_аргумент2*] ...];

Оператор OPEN исполняет запрос, позиционирует курсор на первую запись и устанавливает %ROWCONT в 0.

FETCH{ *курсорная_переменная* | *:хост_переменная*}
INTO{ *переменная1*[, *переменная2*] ... | *запись* };

Оператор FETCH извлекает очередную строку (запись) из курсора и увеличивает значение %ROWCONT на 1; если строка считана, устанавливает %FOUND в TRUE; если курсор исчерпан, устанавливает %NOTFOUND в TRUE.

CLOSE{ *курсорная_переменная* | *:хост_переменная*};

Оператор CLOSE закрывает открытый курсор.

Рассмотрим приведенные конструкции на примере. С помощью курсорных переменных и динамического SQL реализуем обобщенную процедуру запроса к таблице в зависимости от входных данных и возвращения результирующего множества клиенту для дальнейшей обработки.

```
CREATE OR REPLACE PACKAGE my_pkg AS
type refcursor_type IS REF CURSOR;
PROCEDURE get_emps (p_ename IN VARCHAR2 DEFAULT NULL,
p_deptno IN VARCHAR2 DEFAULT NULL,
p_cursor IN OUT refcursor_type);
END;
/

CREATE OR REPLACE PACKAGE BODY my_pkg AS
PROCEDURE get_emps (p_ename IN VARCHAR2 DEFAULT NULL,
p_deptno IN VARCHAR2 DEFAULT NULL,
```

```
p_cursor IN OUT refcursor_type) IS
l_query LONG;
l_bind VARCHAR2(30);
BEGIN
l_query := 'SELECT first_name, job_id FROM emp';
IF (p_ename IS NOT NULL)
THEN l_query := l_query || ' WHERE first_name LIKE(:x)';
l_bind := '%' || UPPER(p_ename) || '%';
ELSIF (p_deptno IS NOT NULL)
THEN l_query := l_query || ' WHERE job_id = TO_NUMBER(:x)';
l_bind := p_deptno;
ELSE RAISE_APPLICATION_ERROR (-20001,'Missing search criteria');
END IF;
OPEN p_cursor FOR l_query USING l_bind;
END;
END;
 /
```

Для выполнения процедуры в SQL*Plus можно использовать команду EXEC или анонимный блок:

```
VARIABLE C REFCURSOR
SET AUTOPRINT ON
BEGIN
my_pkg.get_emps (p_ename=>'a', p_cursor=>:C);
END;
/
```

Приведем еще пример – анонимный блок, в котором создается и заполняется данными таблица:

```
DECLARE
val1 NUMBER(2) :=10;
val2 VARCHAR2(5) := 'test';
BEGIN
EXECUTE IMMEDIATE
'CREATE TABLE test_dyn_sql (value1 NUMBER(2), value2 VARCHAR2(5))';
EXECUTE IMMEDIATE
'INSERT INTO test_dyn_sql (value1, value2) VALUES (:1, :2)'
USING val1, val2;
COMMIT;
END;
/
```

При выполнении динамической SQL-инструкции (это DML- или DDL-строки, не оканчивающиеся точкой с запятой) параметр подстановки необходимо задать для каждой позиции, даже если их имена повторяются.

При выполнении динамического блока PL/SQL параметр подстановки необходимо указывать для каждого уникального формального параметра.

```
CREATE OR REPLACE PROCEDURE updnumval (
coljn IN VARCHAR2,
startjn IN DATE,
endjn IN DATE,
valjn IN NUMBER) IS
dml_str VARCHAR2(32767) := 'UPDATE emp SET ' || coljn || ' =  :val
WHERE hiredate BETWEEN: lodate AND: hidate AND: val IS NOT NULL';
BEGIN
EXECUTE IMMEDIATE dml_str
USING valjn, startjn, endjn, valjn;
END;
/

CREATE OR REPLACE PROCEDURE updnurwal (
coljn IN VARCHAR2,
startjn IN DATE,
endjn IN DATE,
val IN NUMBER) IS
dml_str VARCHAR2(32767) := 'BEGIN
UPDATE emp SET ' || coljn || ' - :val
WHERE hiredate BETWEEN startjn AND endjn
AND :val IS NOT NULL;
END;';
BEGIN
EXECUTE IMMEDIATE dml_str
USING val, startjn, endjn;
END;
/
```

По сравнению с пакетом DBMS_SQL использование EXECUTE IMMEDIATE значительно проще и понятнее. Однако в отличие от EXECUTE IMMEDIATE пакет DBMS_SQL позволяет выполнять выражения динамического SQL, в которых неизвестны число параметров и набор столбцов, возвращаемых запросом или, другими словами, пакет необходимо использовать для динамического SQL с заранее неизвестным количеством входных или выходных переменных.

Задание к лабораторной работе

1. Написать с помощью пакета DBMS_SQL динамическую процедуру или функцию, в которой заранее неизвестен текст команды SELECT. Предусмотреть возможность вывода разных результатов, в зависимости от количества передаваемых параметров.

2. Написать, используя встроенный динамический SQL, процедуру создания в БД нового объекта (представления или таблицы) на основе существующей таблицы. Имя нового объекта должно формироваться динамически и проверяться на существование в словаре данных. В качестве входных параметров указать тип нового объекта, исходную таблицу, столбцы и количество строк, которые будут использоваться в запросе.

3. Выполнить, используя динамический SQL, одно из предложенных заданий:

3.1. Создать процедуру, которая принимает в качестве параметров имя таблицы и имена четырех полей в этой таблице. Первое поле она интерпретирует как ФИО, разбивает его на составляющие и заполняет три оставшихся поля. Если значение первого поля не может быть правильно проинтерпретировано как ФИО (отсутствует отчество, имя и отчество или в строке встречаются недопустимые символы), она помещает в первое поле из трех оставшихся значения ключа (ROWID) этой записи, а во втором и третьем выводит соответствующее сообщение об ошибке строчными и прописными буквами.

3.2. Создать процедуру, которая принимает в качестве параметра имя таблицы и имя поля в этой таблице. Процедура подсчитывает и выводит на экран статистику по этой таблице: количество записей, имя поля, количество различных значений поля, количество null-значений.

3.3. Создать процедуру, которая принимает в качестве параметра имя таблицы. Процедура выводит на экран или в новую таблицу аудита информацию обо всех таблицах, связанных с указанной таблицей по внешнему ключу и принадлежащих пользователю, от имени которого запускается эта процедура. Эта информация включает в себя имя связанной таблицы, общее количество записей в ней и количество различных значений внешнего ключа.

3.4. Создать процедуру, которая делает копию указанной таблицы, добавляя в нее поле «идентификатор» с типом number(n). Параметры: имя исходной таблицы, имя результирующей таблицы, имя добавляемого поля, размер N. Процедура копирует структуру и данные исходной таблицы, проставляя в качестве значения поля «идентификатор» целые числа, начиная с 1.

3.5. Создать процедуру, которая принимает в качестве параметров имя таблицы и имена двух полей в этой таблице. Первое поле имеет символьный тип с маской 'DD.MM.YYYY' и интерпретируется как дата. Во второе поле эта дата проставляется в формате 'YYYY/MM/DD'. Если дата неправильная, то второму полю присваивается значение null.

3.6. Создать процедуру, которая выводит на экран информацию о количестве записей во всех таблицах, принадлежащих пользователю, от имени которого запускается эта процедура.

3.7. Создать функцию, которая принимает в качестве параметра имя таблицы и имя поля в этой таблице и возвращает среднее арифметическое по этому полю. В том случае, если тип поля не позволяет посчитать среднее арифметическое, функция должна возвращать null.

3.8. Создать процедуру, которая принимает в качестве параметра имя таблицы и имена двух полей в этой таблице и добавляет содержание первого поля к содержанию второго. Если поле_2 пустое, то просто копировать поле_1 в поле_2 и наоборот.

3.9. Написать процедуру, которая выведет на экран отчет по указанным полям заданной таблицы, причем по некоторым из них может происходить группировка или сортировка. Также в качестве параметра должен задаваться разделитель столбцов в отчете (по умолчанию «|»).

4. Написать программу, которая позволит для двух указанных в параметрах таблиц существующей БД определить, есть ли между ними связь «один ко многим». Если связь есть, то на основе родительской таблицы создать новую, в которой будут присутствовать все поля старой и одно новое поле с типом коллекции, в котором при переносе данных помещаются все связанные записи из дочерней таблицы.

Контрольные вопросы

1. Что такое коллекция? Какие виды коллекций используются в PL/SQL?

2. В чем состоит основное отличие вложенной таблицы от ассоциативного массива?

3. В чем основное отличие коллекции типа VARRAY от других составных типов?

4. Что значит разреженность? Какие типы коллекций могут быть разреженными?

5. Назовите основные методы коллекций.

6. Для чего используются функции CAST и TABLE?

7. Что означает термин «массовое связывание»? Какие операторы используются для его реализации?

8. В чем необходимость использования динамического SQL в Oracle?

9. Какие способы динамического SQL существуют и для чего они применяются?

10. Каков порядок обработки динамических запросов на извлечение информации при использовании пакета DBMS_SQL?

11. Какие подпрограммы пакета DBMS_SQL не выполняются при обработке DML-операторов?

12. В чем отличие пакета NDS от DBMS_SQL?

ПРИЛОЖЕНИЕ 1

ВСТРОЕННЫЕ ФУНКЦИИ СУБД ORACLE

Таблица П.1.1

Функции для работы со строковыми переменными

Название функции	Действие, выполняемое функцией
ASCII	Возвращает код ASCII левого символа строки
ASCIISTR	Преобразует строку любого набора символов к ASCII строке, используя набор символов базы данных
CHR	По коду ASCII возвращает символ
CONCAT	Позволяет соединить две строки вместе
INITCAP	Устанавливает первый символ каждого слова в верхний регистр, а остальные – в нижний регистр
INSTR	Возвращает n-е вхождение подстроки в строку
LENGTH	Возвращает длину строки
LOWER	Переводит все символы строки в нижний регистр
LPAD	Добавляет с левой части строки определенный набор символов (при ненулевом string1)
LTRIM	Удаляет пробелы в начале строки
NCHR	Возвращает по коду символ Unicode
REGEXP INSTR	Определяет номер первого символа вхождения REGEXP шаблона в строку
REGEXP LIKE	Выбирает из таблицы все строки, соответствующие заданному шаблону регулярного выражения REGEXP
REGEXP REPLACE	Заменяет шаблон регулярного выражения REGEXP в строке на заданную строку
REGEXP SUBSTR	Выделяет из строки заданный REGEXP-шаблон
REGEXP_COUNT	Определяет количество вхождений REGEXP-шаблона в строку
REPLACE	Заменяет вхождения подстроки на указанное значение
RPAD	Дополняет с правой части строки определенный набор символов (при ненулевом string1)
RTRIM	Удаляет пробелы в конце строки
SOUNDEX	Возвращает код звучания строки
SUBSTR	Возвращает для строки подстроку указанной длины с заданного символа
TRANSLATE	Заменяет последовательность символов в строке другим набором символов (заменяет один символ один раз)
TRIM	Удаляет пробелы в строке слева и справа
VSIZE	Возвращает длину в байтах для внутреннего представления выражения
UPPER	Переводит все символы строки в верхний регистр

Таблица П.1.2

Функции для работы с числами

Название функции	Действие, выполняемое функцией
ABS (число)	Вычисляет абсолютное значение числа
ACOS	Вычисляет арккосинус
ASIN	Вычисляет арксинус
ATAN	Вычисляет арктангенс
ATAN2	Вычисляет арктангенс с учетом квадратов
BITAND	Возвращает результат побитовой операции И (AND), выполняемой над двумя числовыми значениями
CEIL (число с дробной частью)	Выполняет округление вверх до ближайшего большего целого
COS (радианы)	Вычисляет косинус угла, указанного в радианах
COSH	Вычисляет гиперболический косинус
EXP (число)	Возвращает результат возведения в указанную степень экспоненты. Если число = 1, то получим само значение exp
FLOOR (число с дробной частью)	Выполняет округление вниз до ближайшего целого
LN (число)	Вычисляет натуральный логарифм числа
LOG (основание, число)	Вычисляет логарифм, т. е. в какую степень нужно возвести основание, чтобы получить указанное число. Например: log(10,100) равен 2
MOD (делимое, делитель)	Возвращает остаток от деления
POWER (число, степень)	Возводит число в степень
ROUND (значение, число знаков для округления)	Выполняет округление с заданной точностью
SIGN (число)	Определяет знак числа, возвращает 1 или 0, или –1
SIN (радианы)	Вычисляет синус угла, указанного в радианах
SINH	Вычисляет гиперболический синус
SQRT (число)	Извлекает квадратный корень из неотрицательного числа
TAN (радианы)	Возвращает тангенс угла, указанного в радианах
TANH	Вычисляет гиперболический тангенс
TRUNC (число)	Возвращает целую часть числа

Таблица П.1.3

Основные функции для работы с датой и временем

Название функции	Действие, выполняемое функцией
ADD_MONTHS	Возвращает дату плюс n месяцев
CURRENT_DATE	Возвращает текущую дату в часовом поясе текущей сессии SQL (как установлено с помощью команды ALTER SESSION)
CURRENT_TIME	Возвращает текущее время в часовом поясе текущей сессии SQL
CURRENT_TIMESTAMP	Возвращает текущую дату и время в часовом поясе текущей сессии SQL
DBTIMEZONE	Возвращает смещение часового пояса базы данных в формате '[+|-]TZH:TZM' или название области часового пояса
EXTRACT	Извлекает значение из даты или значения интервала
FROM_TZ	Преобразует значение TIMESTAMP в TIMESTAMP со значением часового пояса
LAST_DAY	Возвращает последний день месяца на основе значения даты
LOCALTIMESTAMP	Возвращает текущую дату и время в часовом поясе из текущей сессии SQL
MONTHS_BETWEEN	Возвращает количество месяцев между date1 и date2
NEW_TIME	Возвращает дату и время часового пояса zone2 для даты и времени часового пояса zone1, заданных DATE
NEXT_DAY	Возвращает первый день недели, который больше DATE
NUMTODSINTERVAL	Преобразует числовое значение в значение типа INTERVAL DAY TO SECOND
NUMTOYMINTERVAL	Преобразует числовое значение в значение типа INTERVAL YEAR TO MONTH
ROUND	Возвращает дату, округленную до определенной единицы измерения
SESSIONTIMEZONE	Возвращает смещение часового пояса текущей сессии в формате: '[+ | -] TZH: TZM') или наименование региона часового пояса

Окончание табл. П.1.3

Название функции	Действие, выполняемое функцией
SYSDATE	Возвращает текущую системную дату и время на локальной базе данных
SYSTIMESTAMP	Возвращает текущую системную дату и время (в том числе доли секунды и часового пояса) на локальной базе данных
TO_DSINTERVAL	Преобразует строку в интервал DAY TO SECONDS
TO_DATE	Преобразует строку в дату
TO_TIMESTAMP	Преобразует строку в значение TIMESTAMP
TO_TIMESTAMP_TZ	Преобразует строку в значение типа TIMESTAMP с TIMEZONE (часовым поясом)
TO YMINTERVAL	Преобразует строку в значение типа NTERVAL YEAR TO MONTH
TRUNC	Возвращает дату, усеченную к определенной единице измерения (шаблону)
TZ_OFFSET	Возвращает смещение часового пояса из значения

ПРИЛОЖЕНИЕ 2

ВАРИАНТЫ ЗАДАНИЙ ДЛЯ СОЗДАНИЯ ИНДИВИДУАЛЬНОЙ БД

База создается на основе ERD, разработанной в предыдущем семестре. Общее количество полей в базе – не менее 20. Общее количество записей – не менее 30.

Вариант 1. Создайте БД отдела кадров университета. Ориентировочные таблицы-составляющие: «Сотрудники», **«Контракты»**, «Кафедры», «Должности».

Вариант 2. Создайте БД галантерейного магазина. Ориентировочные таблицы-составляющие: «Партии товара», «Поставщики», **«Продажи»**, «Продавцы».

Вариант 3. Создайте БД отдела доставки почтового отделения. Ориентировочные таблицы-составляющие: «Подписчики», «Периодика», **«Подписки»,** «Почтальоны».

Вариант 4. Создайте БД универмага. Ориентировочные таблицы-составляющие: «Товары», «Отделы», **«Продажи»**, «Продавцы».

Вариант 5. Создайте БД штатного расписания предприятия. Ориентировочные таблицы-составляющие: «Отделы», «Должности», «Тип производства», «Сотрудники», **«Штатное расписание».**

Вариант 6. Создайте БД фирмы по производству пиломатериалов. Ориентировочные таблицы-составляющие: «Изделия», «Сырье», **«Продажи»**, «Поставщики».

Вариант 7. Создайте БД программы выпуска деталей литейного цеха. Ориентировочные таблицы-составляющие: «Детали», «Материал», **«Технологические процессы»**, «Рабочие».

Вариант 8. Создайте БД инфекционного отделения городской больницы. Ориентировочные таблицы-составляющие: «Койко-место», «Больные», «Диагнозы», **«Госпитализация»**, «Врачи».

Вариант 9. Создайте БД процессов обработки партий деталей. Ориентировочные таблицы-составляющие: «Детали», «Оборудование», **«Технологические карты»**, «Процессы», «Бригады».

Вариант 10. Создайте БД фирмы по оптовой реализации бытовой техники. Ориентировочные таблицы-составляющие: «Продукция», «Клиенты», **«Заказы»**, «Производители».

Вариант 11. Создайте БД оборудования НИИ. Ориентировочные таблицы-составляющие: «Оборудование», «Исследовательские работы», **«Акты проведения работ»**, «Руководители работ».

Вариант 12. Создайте БД фирмы по производству столярных работ. Ориентировочные таблицы-составляющие: «Продукция», «Клиенты», **«Договора»,** «Услуги», «Материалы».

Вариант 13. Создайте БД агентства недвижимости. Ориентировочные таблицы-составляющие: «Объекты недвижимости», «Типы объектов», «Типы сделок», «Покупатели», «Продавцы», **«Сделки».**

Вариант 14. Создайте БД фирмы-распространителя программного обеспечения. Ориентировочные таблицы-составляющие: «Программное обеспечение», «Клиенты», **«Производимые работы»**, «Производители ПО», «Гарантийные сроки».

Вариант 15. Создайте БД инструментального склада. Ориентировочные таблицы-составляющие: «Обслуживаемое оборудование», «Инструмент», **«Технологические карты»**, «Рабочие».

Вариант 16. Создайте БД страховой фирмы. Ориентировочные таблицы-составляющие: «Виды страховок», «Клиенты», «Объекты страховки», **«Страховая деятельность»**, «Агенты».

Вариант 17. Создайте БД подъемно-транспортного оборудования машиностроительного предприятия. Ориентировочные таблицы-составляющие: «Техника», «Производимые работы», «Виды ремонта», «Рабочие», **«Учет работ».**

Вариант 18. Создайте БД музея. Ориентировочные таблицы-составляющие: «Экспонаты», «Авторы», «Виды экспозиции», «Жанры», **«Проведение выставки».**

Вариант 19. Создайте БД НИИ. Ориентировочные таблицы-составляющие: «Сотрудники», «Научно-исследовательские разработки», **«Составление штатного расписания»**, «Отделы», «Должности».

Вариант 20. Создайте БД студии видеозаписи. Ориентировочные таблицы-составляющие: «Режиссеры», «Актеры», «Фильмы», «Жанры», **«Продажи».**

Вариант 21. Создайте БД деканата. Ориентировочные таблицы-составляющие: «Список курсов и групп», «Список студентов», **«Сдача сессии»**, «Предметы», «Преподаватели».

Вариант 22. Создайте БД автозаправочной станции. Ориентировочные таблицы-составляющие: «Горюче-смазочные материалы (ГСМ)», «Поставщики», **«Накладные»**, «Рабочие АЗС».

Вариант 23. Создайте БД выставки продукции. Ориентировочные таблицы-составляющие: «Продукция», «Предприятие», «Выставочное место», **«Организация выставки».**

Вариант 24. Создайте БД полиграфической фирмы. Ориентировочные таблицы-составляющие: «Материалы», «Техника», **«Контракты»**, «Заказчики», «Перечень работ».

Вариант 25. Создайте БД фирмы-поставщика медицинской техники. Ориентировочные таблицы-составляющие: «Ассортимент», «Заказчики», **«Контракты»**, «Производители».

Вариант 26. Создайте БД планово-финансового отдела. Ориентировочные таблицы-составляющие: «Участки», «Работники», **«Наряды»**, «Мероприятия», «Должности».

Вариант 27. Создайте БД строительной фирмы. Ориентировочные таблицы-составляющие: «Объекты», «Этапы выполнения», «Стройматериалы», «Бригады», **«Строительство».**

Вариант 28. Создайте БД опытного цеха. Ориентировочные таблицы-составляющие: «Производимые изделия», «Оборудование», **«Технологические карты»,** «Рабочие», «Инвентарь».

Вариант 29. Создайте БД механообрабатывающего цеха. Ориентировочные таблицы-составляющие: «Детали», «Материалы», «Оснастка», **«Работа цеха»**, «Рабочие».

Вариант 30. Создайте БД фирмы по установке и обслуживанию локальной вычислительной сети (ЛВС). Ориентировочные таблицы-составляющие: «Сетевые конфигурации», «Клиенты», **«Договора»**, «Оборудование», «Исполнители».

Вариант 31. Создайте БД фирмы по продаже стройматериалов. Ориентировочные таблицы-составляющие: «Сырье», «Продукция», «Заказчики», **«Контракты».**

Вариант 32. Создайте БД ветеринарной станции. Ориентировочные таблицы-составляющие: «Ветеринарные работы», «Обслуживаемые хозяйства», «Медикаменты», «Ветеринары», **«Проведенные работы».**

Вариант 33. Создайте БД фирмы по проведению буровых работ. Ориентировочные таблицы-составляющие: «Скважины», «Клиенты», «Бригады», «Оборудование», **«Контракты».**

Вариант 34. Создайте БД загрузки аудиторий. Ориентировочные таблицы-составляющие: «Аудитории», «Учебные дисциплины», «Преподаватели», «Кафедры», «Группы», **«Загрузка аудиторий».**

Вариант 35. Создайте БД трикотажной фабрики. Ориентировочные таблицы-составляющие: «Сырье», «Изделия», **«Сбыт изделий»**, «Покупатели», «Размеры».

Вариант 36. Создайте БД ателье головных уборов. Ориентировочные таблицы-составляющие: «Изделия», «Клиенты», «Материалы», «Размеры», **«Квитанции».**

Вариант 37. Создайте БД гостиницы. Ориентировочные таблицы-составляющие: «Номера», «Клиенты», «Обслуживающий персонал», **«Счета»,** «Услуги».

Вариант 38. Создайте БД жилищного коммунального хозяйства. Ориентировочные таблицы-составляющие: «Специалисты», «Жилищный фонд», «Мероприятия», **«Заказы».**

Вариант 39. Создайте БД стоматологической поликлиники. Ориентировочные таблицы-составляющие: «Врачи», «Пациенты», **«Обслуживание»,** «Услуги».

Вариант 40. Создайте БД сборочного процесса. Ориентировочные таблицы-составляющие: «Комплектующие», «Изделия», **«Сборка»**, «Технологические процессы».

Вариант 41. Создайте БД мебельной фабрики. Ориентировочные таблицы-составляющие: «Изделия», «Материалы», «Заказчики», «Поставщики», **«Контракты»**.

Вариант 42. Создайте БД кабельного завода. Ориентировочные таблицы-составляющие: «Сырье», «Продукция», «Технологические процессы», **«Производство»**, «Рабочие».

Вариант 43. Создайте БД механизированной колонны. Ориентировочные таблицы-составляющие: «Техника», «Выполняемые работы», **«Путевки»**, «Водители».

Вариант 44. Создайте БД санатория. Ориентировочные таблицы-составляющие: «Оздоровительные программы», «Отдыхающие», «Врачи», **«Обслуживание»**, «Заболевания».

Вариант 45. Создайте БД геолого-разведочной экспедиции. Ориентировочные таблицы-составляющие: «Регионы», «Карты», **«Экспедиции»**, «Геологи».

Вариант 46. Создайте БД фирмы по автоматизации производства. Ориентировочные таблицы-составляющие: «Каталог устройств» «Программное обеспечение», «Клиенты», «Персонал», «Услуги», **«Договора».**

ПРИЛОЖЕНИЕ 3

ВАРИАНТЫ ЗАДАНИЙ ДЛЯ НАПИСАНИЯ ЗАПРОСОВ

Вариант 1. Создайте запросы: «Доценты» (условная выборка); «Сводка количества работающих на каждой должности» (итоговый запрос); «Сотрудники, нуждающиеся в продлении контракта» (параметрический запрос); «Общий список сотрудников и кафедр с количеством контрактов по каждой позиции» (запрос на объединение); «Количество сотрудников на кафедрах по годам» (запрос по полю с типом дата).

Вариант 2. Создайте запросы: «Залежавшийся товар» (условная выборка); «Рейтинг спроса по фирмам» (итоговый запрос); «Поставщики партий заданного объема» (параметрический запрос); «Общий список продавцов и поставщиков с суммами продаж каждого» (запрос на объединение); «Количество продаж каждого вида товаров по месяцам текущего года» (запрос по полю с типом дата).

Вариант 3. Создайте запросы: «Подписчики газеты «Вечерний Минск» (условная выборка); «Количество подписок на каждое издание» (итоговый запрос); «Подписки дешевле заданной стоимости» (параметрический запрос); «Общий список подписчиков и почтальонов с количеством подписок у каждого» (запрос на объединение); «Суммы подписок по кварталам» (запрос по полю с типом дата).

Вариант 4. Создайте запросы: «Список отделов, реализующих парфюмерию» (условная выборка); «Сводка продаж по отделам» (итоговый запрос); «Список продукции в заданном отделе» (параметрический запрос); «Общий список отделов и продавцов с указанием сумм продаж по каждой позиции» (запрос на объединение); «Суммы продаж по неделям года» (запрос по полю с типом дата).

Вариант 5. Создайте запросы: «Список сотрудников, не занятых в основном производстве» (условная выборка); «Сводка по заработной плате каждого отдела» (итоговый запрос); «Список сотрудников заданной должности» (параметрический запрос); «Общий список отделов и должностей с количеством сотрудников по каждой позиции» (запрос на объединение); «Количество сотрудников штатного расписания по месяцам (выводить название месяца)» (запрос по полю с типом дата).

Вариант 6. Создайте запросы: «Реализованные изделия за последнюю неделю» (условная выборка); «Сводка расхода сырья» (итоговый запрос); «Продажи изделий заданной породы древесины» (параметрический запрос); «Общий список изделий и сырья с указанием количества по каждой позиции» (запрос на объединение); «Суммы продаж товаров по неделям текущего месяца» (запрос по полю с типом дата).

Вариант 7. Создайте запросы: «Чугунные отливки большого объема» (условная выборка); «Расход материала в плановом периоде» (итоговый запрос); «Процессы выплавки деталей заданных габаритов» (параметрический

запрос); «Общий список деталей и материалов с указанием количества по каждой позиции» (запрос на объединение); «Количество сделанных деталей с 9:00 до 12:00» (запрос по полю с типом дата).

Вариант 8. Создайте запросы: «Больные-пенсионеры» (условная выборка); «Количество больных в каждой палате» (итоговый запрос); «Палаты больных с заданной температурой» (параметрический запрос); «Общий список врачей с количеством обслуженных больных и больных с количеством дней пребывания» (запрос на объединение); «Количество заболевших по годам» (запрос по полю с типом дата).

Вариант 9. Создайте запросы: «Список оборудования для высококачественной обработки деталей» (условная выборка); «Загруженность оборудования» (итоговый запрос); «Партии, проходящие заданную операцию» (параметрический запрос); «Общий список деталей и оборудования с количеством использований в картах» (запрос на объединение); «Количество деталей по месяцам текущего года» (запрос по полю с типом дата).

Вариант 10. Создайте запросы: «Крупнейшие партии» (условная выборка), «Сводка по технике, закупленной каждым клиентом» (итоговый запрос); «Продукция, реализованная в заданный период времени» (параметрический запрос); «Общий список клиентов и производителей с количеством продаж» (запрос на объединение); «Количество проданных товаров фирм-производителей по кварталам» (запрос по полю с типом дата).

Вариант 11. Создайте запросы: «Наиболее загруженное оборудование» (условная выборка); «Количество часов наработки» (итоговый запрос); «Проведение работ на заданной единице оборудования» (параметрический запрос); «Общий список оборудования и работ с количеством использований в картах» (запрос на объединение); «Количество используемого оборудования по месяцам» (запрос по полю с типом дата).

Вариант 12. Создайте запросы: «Постоянные клиенты фирмы» (условная выборка); «Прибыль по каждому виду продукции/услуги» (итоговый запрос); «Продукция/услуги, реализованные на заданную сумму» (параметрический запрос); «Общий список услуг и продукции с количеством договоров по каждой позиции» (запрос на объединение); «Количество договоров по неделям года» (запрос по полю с типом дата).

Вариант 13. Создайте запросы: «Список объектов, предлагаемых к продаже» (условная выборка); «Сальдо по видам объектов» (итоговый запрос); «Объекты заданной стоимости» (параметрический запрос); «Общий список покупателей и продавцов с количеством сделок» (запрос на объединение); «Количество сделок по районам и по годам» (запрос по полю с типом дата).

Вариант 14. Создайте запросы: «Список клиентов, программному обеспечению которых предстоит обновление» (условная выборка); «Сводка реализованного программного обеспечения» (итоговый запрос); «Список клиентов, купивших продукцию заданного вида» (параметрический запрос); «Общий список ПО с количеством продаж и производителей с количеством выпускаемого

ПО» (запрос на объединение); «Количество дней от покупки ПО до его обновления» (запрос по полю с типом дата).

Вариант 15. Создайте запросы: «Инструмент с высокой степенью используемости» (условная выборка); «Количество единиц инструмента для каждого оборудования» (итоговый запрос); «Список инструмента, подверженного повышенному износу» (параметрический запрос); «Общий список видов оборудования и инструментов с указанием количества» (запрос на объединение); «Количество задействованного оборудования по месяцам и видам оборудования» (запрос по полю с типом дата).

Вариант 16. Создайте запросы: «Клиенты, застраховавшие свою жизнь за последний месяц» (условная выборка); «Сводка полученных/выплаченных сумм страховок по клиентам» (итоговый запрос); «Объекты, застрахованные на заданную сумму» (параметрический запрос); «Общий список клиентов и агентов с количеством договоров у каждого» (запрос на объединение); «Заключенные договора по кварталам за два последних года» (запрос по полю с типом дата).

Вариант 17. Создайте запросы: «Работа техники высокой грузоподъемности» (условная выборка); «Сводка часов простоя единиц оборудования во внеплановом ремонте» (итоговый запрос); «Список техники с заданным коэффициентом загрузки» (параметрический запрос); «Общий список рабочих с количеством работ и техники с количеством ремонтов» (запрос на объединение); «Количество поломок по годам» (запрос по полю с типом дата).

Вариант 18. Создайте запросы: «Малоизвестные экспонаты» (условная выборка); «Выставлено работ по авторам» (итоговый запрос); «Экспозиции работ повышенной ценности» (параметрический запрос); «Общий список авторов с количеством работ и экспонатов с количеством экспозиций» (запрос на объединение); «Количество выставок по кварталам» (запрос по полю с типом дата).

Вариант 19. Создайте запросы: «Сотрудники, задействованные в научно-исследовательских разработках» (условная выборка); «Выплаченная заработная плата (по отделам)» (итоговый запрос); «Список ответственных по научно-исследовательским разработкам» (параметрический запрос); «Общий список разработок и сотрудников с суммами затрат» (запрос на объединение); «Количество завершенных научных работ по месяцам (выводить название месяца)» (запрос по полю с типом дата).

Вариант 20. Создайте запросы: «Оскар» (условная выборка); «Рейтинг продаж по актерам» (итоговый запрос); «Фильмы с заданным актерским дуэтом» (параметрический запрос); «Общий список режиссеров и актеров с указанием количества фильмов, где они участвуют» (запрос на объединение); «Суммы продаж по неделям текущего года» (запрос по полю с типом дата).

Вариант 21. Создайте запросы: «Неуспевающие студенты» (условная выборка); «Средний балл по предметам» (итоговый запрос); «Студенты, имеющие хорошие оценки по заданному предмету» (параметрический запрос); «Общий список студентов и преподавателей с количеством изучаемых или преподавае-

мых предметов» (запрос на объединение); «Средний балл групп по годам обучения» (запрос по полю с типом дата).

Вариант 22. Создайте запросы: «Закупки дизельного топлива» (условная выборка); «Затраты по видам продукции» (итоговый запрос); «Поставщики дешевой смазки» (параметрический запрос); «Общий список поставщиков и ГСМ с количеством накладных» (запрос на объединение); «Количество накладных рабочего по месяцам текущего года» (запрос по полю с типом дата).

Вариант 23. Создайте запросы: «Предприятия, арендовавшие наибольшие выставочные площади» (условная выборка); «Сводка по продукции/предприятиям» (итоговый запрос); «Продукция заданного предприятия» (параметрический запрос); «Общий список продукции на выставках с указанием количества и предприятий с количеством выставок» (запрос на объединение); «Количество выставок и общее количество товаров по годам» (запрос по полю с типом дата).

Вариант 24. Создайте запросы: «Материалы заказов малых объемов» (условная выборка); «Сводка расходов материалов» (итоговый запрос); «Заказы на заданную сумму» (параметрический запрос); «Общий список материалов и техники с количеством используемых в заказах» (запрос на объединение); «Количество заказчиков по кварталам» (запрос по полю с типом дата).

Вариант 25. Создайте запросы: «Последние поступления» (условная выборка); «Рейтинг заказчиков по общим суммам контрактов» (итоговый запрос); «Клиенты, заказавшие заданный вид изделия» (параметрический запрос); «Общий список производителей с количеством изделий и заказчиков с количеством контрактов» (запрос на объединение); «Количество проданного ассортимента по месяцам текущего года» (запрос по полю с типом дата).

Вариант 26. Создайте запросы: «Опасные работы» (условная выборка); «Начисление заработной платы по участкам» (итоговый запрос); «Списки работников по заданным датам/участкам» (параметрический запрос); «Общий список работников и участков с количеством нарядов по каждой позиции» (запрос на объединение); «Количество нарядов по дням недели (выводить названия)» (запрос по полю с типом дата).

Вариант 27. Создайте запросы: «Текущие этапы работы» (условная выборка); «Сроки строительства объектов» (итоговый запрос); «Объекты заданного процента завершенности» (параметрический запрос); «Общий список объектов и стройматериалов с указанием количества этапов» (запрос на объединение); «Количество построенных объектов по кварталам» (запрос по полю с типом дата).

Вариант 28. Создайте запросы: «Изделия/карты фрезерной обработки» (условная выборка); «Количество изделий по каждому оборудованию» (итоговый запрос); «Карты изделий заданных габаритов» (параметрический запрос); «Общий список изделий и оборудования с количеством карт, в которых они используются» (запрос на объединение); «Количество единиц занятого оборудования по часам за текущий день» (запрос по полю с типом дата).

Вариант 29. Создайте запросы: «Оснастка для обработки деталей высокой твердости» (условная выборка); «Расход материалов по видам деталей»

(итоговый запрос); «Детали заданного материала» (параметрический запрос); «Общий список материалов и оснастки с общим количеством по каждому виду» (запрос на объединение); «Количество рабочих по дням недели» (запрос по полю с типом дата).

Вариант 30. Создайте запросы: «Список клиентов, с которыми необходимо перезаключать договора» (условная выборка); «Протяженность кабеля у каждого клиента» (итоговый запрос); «Список ЛВС с заданной конфигурацией» (параметрический запрос); «Общий список клиентов и исполнителей с количеством договоров по каждому» (запрос на объединение); «Количество договоров по сетевым конфигурациям в год» (запрос по полю с типом дата).

Вариант 31. Создайте запросы: «Продажа кирпича» (условная выборка); «Сводка сумм контрактов по каждому заказчику» (итоговый запрос); «Список продукции, заказанной в заданный период» (параметрический запрос); «Общий список продукции и сырья с указанием количества в наличии» (запрос на объединение), «Количество контрактов в год по заказчикам» (запрос по полю с типом дата).

Вариант 32. Создайте запросы: «Список хозяйств, запланированных на ближайшую неделю» (условная выборка); «Объемы проведенных работ в каждом хозяйстве» (итоговый запрос); «Выполнение плана по хозяйствам на заданное число» (параметрический запрос); «Общий список обслуживаемых хозяйств и ветеринаров с количеством обслуживаемых животных» (запрос на объединение); «Количество животных по дням текущей недели» (запрос по полю с типом дата).

Вариант 33. Создайте запросы: «Контракты на бурение глубинных скважин» (условная выборка); «Количество контрактов на каждый вид работ» (итоговый запрос); «Контракты с заданным клиентом» (параметрический запрос); «Общий список клиентов и бригадиров с количеством контрактов у каждого» (запрос на объединение); «Общий метраж скважин за последние 10 недель» (запрос по полю с типом дата).

Вариант 34. Создайте запросы: «Поточные (лекционные) аудитории» (условная выборка); «Рейтинг дисциплин по загрузке аудиторий» (итоговый запрос); «Аудитории для заданной дисциплины» (параметрический запрос); «Общий список кафедр и групп с указанием количества студентов» (запрос на объединение); «Количество занятий групп по дням недели» (запрос по полю с типом дата).

Вариант 35. Создайте запросы: «Сбыт изделий, содержащих искусственные волокна» (условная выборка); «Рейтинг себестоимости изделий» (итоговый запрос); «Картина сбыта изделий заданного размера» (параметрический запрос); «Общий список используемого сырья и изделий с указанием количества в наличии» (запрос на объединение); «Квитанции за последние пять дней» (запрос по полю с типом дата).

Вариант 36. Создайте запросы: «Заказчики зимнего ассортимента» (условная выборка); «Популярность моделей» (итоговый запрос); «Модели

шляп заданной стоимости» (параметрический запрос); «Общий список изделий и материалов с количеством в наличии» (запрос на объединение); «Количество изделий по годам» (запрос по полю с типом дата).

Вариант 37. Создайте запросы: «Богатые клиенты» (условная выборка); «Рейтинг загрузки номеров» (итоговый запрос); «Счета указанного клиента» (параметрический запрос); «Общий список клиентов с количеством номеров проживания и персонала с количеством номеров обслуживания» (запрос на объединение); «Общий доход по месяцам за текущий год» (запрос по полю с типом дата).

Вариант 38. Создайте запросы: «Деятельность, запланированная на ближайшую неделю» (условная выборка); «Рейтинг работников по количеству проведенных мероприятий» (итоговый запрос); «Мероприятия, проведенные заданным работником» (параметрический запрос); «Общий список специалистов и мероприятий с указанием количества заказов» (запрос на объединение); «Количество проведенных мероприятий и задействованных специалистов по месяцам» (запрос по полю с типом дата).

Вариант 39. Создайте запросы: «Протезирование» (условная выборка); «Рейтинг услуг» (итоговый запрос); «Пациенты заданного врача» (параметрический запрос); «Общий список врачей и пациентов с количеством оказанных услуг» (запрос на объединение); «Количество пациентов по дням текущего месяца» (запрос по полю с типом дата).

Вариант 40. Создайте запросы: «Изделия, требующие микросборки» (условная выборка); «Количество комплектующих в каждом изделии» (итоговый запрос); «Информация по изделиям заданного технологического процесса» (параметрический запрос); «Общий список комплектующих и изделий с количеством в наличии по каждой позиции» (запрос на объединение); «Количество изделий по кварталам» (запрос по полю с типом дата).

Вариант 41. Создайте запросы: «Контракты по изготовлению корпусной мебели» (условная выборка); «Рейтинг продукции» (итоговый запрос); «Последние контракты» (параметрический запрос); «Общий список заказчиков и поставщиков с количеством контрактов у каждого» (запрос на объединение); «Количество изделий в месяц (выводить название месяца)» (запрос по полю с типом дата).

Вариант 42. Создайте запросы: «Монтажные провода» (условная выборка); «Затраты сырья по видам продукции» (итоговый запрос); «Исходные материалы для производства проводников заданного типа» (параметрический запрос); «Общий список сырья и продукции с количеством по каждой позиции» (запрос на объединение); «Общий расход материалов по годам» (запрос по полю с типом дата).

Вариант 43. Создайте запросы: «Экскаваторщики» (условная выборка); «Количество отработанных персоналом часов за неделю» (итоговый запрос); «Наряды в заданный промежуток времени» (параметрический запрос); «Общий список водителей и техники с количеством нарядов по каждой позиции»

(запрос на объединение); «Количество водителей техники по неделям года» (запрос по полю с типом дата).

Вариант 44. Создайте запросы: «Программы по желудочно-кишечным заболеваниям» (условная выборка); «Сводка отдыхающих за плановый период» (итоговый запрос); «Отдыхающие, завершающие курс оздоровления» (параметрический запрос); «Общий список врачей и отдыхающих с количеством программ у каждого» (запрос на объединение); «Количество заболеваний по месяцам» (запрос по полю с типом дата).

Вариант 45. Создайте запросы: «Экспедиции по Беларуси» (условная выборка); «Протяженность маршрутов по регионам» (итоговый запрос); «Перечень карт заданного региона» (параметрический запрос); «Общий список регионов и геологов с количеством экспедиций по каждой позиции» (запрос на объединение); «Количество карт по кварталам текущего года» (запрос по полю с типом дата).

Вариант 46. Создайте запросы: «Услуги, оказанные в текущем месяце» (условная выборка); «Сводка доходов по виду услуг за отчетный период» (итоговый запрос); «Договора по установке программного обеспечения» (параметрический запрос); «Общий список клиентов и персонала с количеством договоров каждого» (запрос на объединение); «Количество договоров по годам» (запрос по полю с типом дата).

ПРИЛОЖЕНИЕ 4

Таблица П.4.1

ИСКЛЮЧИТЕЛЬНЫЕ СИТУАЦИИ ORACLE

Exception	Oracle Error	SQLCODE Value
ACCESS_INTO_NULL	ORA-06530	–6530
CASE_NOT_FOUND	ORA-06592	–6592
COLLECTION_IS_NULL	ORA-06531	–6531
CURSOR_ALREADY_OPEN	ORA-06511	–6511
DUP_VAL_ON_INDEX	ORA-00001	–1
INVALID_CURSOR	ORA-01001	–1001
INVALID_NUMBER	ORA-01722	–1722
LOGIN_DENIED	ORA-01017	–1017
NO_DATA_FOUND	ORA-01403	+100
NOT_LOGGED_ON	ORA-01012	–1012
PROGRAM_ERROR	ORA-06501	–6501
ROWTYPE_MISMATCH	ORA-06504	–6504
SELF_IS_NULL	ORA-30625	–30625
STORAGE_ERROR	ORA-06500	–6500
SUBSCRIPT_BEYOND_COUNT	ORA-06533	–6533
SUBSCRIPT_OUTSIDE_LIMIT	ORA-06532	–6532
SYS_INVALID_ROWID	ORA-01410	–1410
TIMEOUT_ON_RESOURCE	ORA-00051	–51
TOO_MANY_ROWS	ORA-01422	–1422
VALUE_ERROR	ORA-06502	–6502
ZERO_DIVIDE	ORA-01476	–1476

ВАРИАНТЫ ЗАДАНИЙ ДЛЯ НАПИСАНИЯ ПРОЦЕДУР И ФУНКЦИЙ

Вариант 1. Написать процедуру, которая повышает заработную плату сотрудников:

– до размера в 1,5 прожиточного минимума, если оклад меньше прожиточного минимума;

– на 5 %, если оклад больше, чем 1,5 прожиточного минимума.

Входной параметр – прожиточный минимум. Выводить количество обновленных записей.

Создать функцию, которая определяет, работает ли тот или иной сотрудник на указанной кафедре. Функция возвращает или значение FALSE и в параметре OUT соответствующий комментарий, или – TRUE и в параметре OUT – номер телефона сотрудника.

Вариант 2. Написать процедуру, которая формирует заказ на товары, которых осталось меньше необходимого запаса и выводит его на экран, подсчитывая общий итог. Величину минимального запаса передавать во входном параметре.

Создать функцию, возвращающую пустую строку или строку «закончился срок реализации», если дата поставки плюс срок реализации партии товара больше текущей даты и времени. Поля даты поставки и срока реализации должны присутствовать в таблицах БД.

Вариант 3. Создать процедуру, копирующую строки с информацией о подписках за текущий месяц, во вспомогательную таблицу. Все коды в таблице подписок должны быть заменены на реальные данные из связанных таблиц.

Написать функцию, которая выводит информацию о подписчиках, которые оформили количество подписок за указанный год больше, чем указано во входном параметре. Возвращать общее количество подписчиков.

Вариант 4. Создать процедуру, распечатывающую список товаров с их общим количеством продаж, старой и новой ценой (для товаров, которые не продавались в течение месяца снизить цену на 10 %) по отделам:

Отдел 1			
товар1	количество	старая_цена	новая_цена
товар2	количество	старая_цена	новая_цена
Отдел 2			
товар1	количество	старая_цена	новая_цена
товар2	количество	старая_цена	новая_цена

Написать функцию, которая возвращает общее количество продавцов, работающих в отделах. Вывести также количество продавцов по отделам и продавцов, которые еще не участвовали в продажах.

Вариант 5. Создать процедуру расчета общего количества отработанных часов в виде

Отдел 1	
фамилия1	количество часов
фамилияN	количество часов
Отдел 2	
фамилия1	количество часов
фамилияN	количество часов

Написать функцию, определяющую, являются ли сотрудники указанного отдела пенсионерами с учетом разного возраста выхода на пенсию мужчин и женщин. В результате работы выводить ФИО сотрудников отдела с пометками «пенсионер» или «не достигнут пенсионный возраст», а также возвращать общее количество пенсионеров.

Вариант 6. Создать процедуру, переносящую в архив информацию о товарах, которые реализованы ранее текущего месяца. Архив должен содержать не значения ключей, а реальные названия изделий и сырья.

Написать функцию, которая возвращает пустую строку или строку «остатки», если заданного товара осталось меньше 10 единиц, а также выводит перечень товаров с оставшимся количеством.

Вариант 7. Создать хранимую процедуру, которая выводит количество сделанных деталей заданного наименования и в случае отсутствия заказов на определенную деталь в течение года удаляет ее из исходной таблицы.

Написать функцию, которая возвращает общее количество деталей и выводит список использованных материалов с указанием количества.

Вариант 8. Создать процедуру, выводящую список палат с указанием количества коек и статуса палаты:

- «пустая», если в палате никто не лежит;
- «свободных мест нет», если палата заполнена;
- «мужская», если в палате лежат только мужчины;
- «женская», если в палате лежат только женщины;
- «смешанная» во всех остальных случаях.

Написать функцию, возвращающую строку «больше месяца», если со времени поступления пациента прошло более одного месяца. В качестве параметра использовать диагноз и заменять для такого пациента его лечащего врача на того, у кого меньше всего больных.

Вариант 9. Создать процедуру, которая копирует строки с информацией об оборудовании указанного вида во вспомогательную таблицу и подсчитывает количество деталей, произведенных на данном оборудовании.

Создать функцию, возвращающую количество рабочих в бригаде. Входной параметр функции – id бригады. Если в бригаде менее трех человек, то обновить количество до пяти человек.

Вариант 10. Написать процедуру, которая изменяет домашний адрес клиента по указанной в качестве параметра фамилии. В случае обнаружения однофамильцев выводить их количество и уточняющие данные.

Создать функцию, подсчитывающую общее количество реализованной продукции заданного производителя. Если какой-либо продукции реализовано менее 1 единицы в год, то удалить информацию об этой продукции во вспомогательную таблицу.

Вариант 11. Создать процедуру, обеспечивающую вывод информации обо всем оборудовании, участвующем в указанной ключевым словом работе.

Написать функцию, которая возвращает строку «не руководит работами», если нет ни одного акта в настоящий момент у переданного в качестве параметра сотрудника.

Вариант 12. Создать процедуру изменения стоимости услуги. Входные параметры – id услуги и новая стоимость. Вывести отчет о том какая именно услуга подорожала или подешевела и на сколько процентов.

Написать функцию, которая определяет количество заключенных договоров за указанный период. В качестве параметра передать начальную и конечную даты периода.

Вариант 13. Написать процедуру изменения мобильного номера продавца по указанной в качестве параметра фамилии. Контролировать, чтобы повторно не был введен тот же номер.

Создать функцию, подсчитывающую количество сделок, совершенных потенциальными покупателями за текущий день. В вызывающую среду возвращать объекты недвижимости, участвующие в этих сделках.

Вариант 14. Создать процедуру изменения стоимости ПО. Входные параметры – производитель программного продукта и новая стоимость. Вывести количество обновленных записей.

Написать функцию, подсчитывающую количество клиентов, которым предстоит обновить указанное ПО в ближайшие два месяца.

Вариант 15. Написать процедуру, создающую отчет о движении инструментов за указанный в днях период:

Отчет за период с … по …

1. инструмент1 поступление (ед. изм.) количество используемого
2. инструмент 2 поступление (ед. изм.) количество используемого

…

Создать функцию, которая возвращает пустую строку или строку «остатки», если инструмента на складе осталось меньше 100 единиц, а также выводит инструмент с минимальным количеством в остатке.

Вариант 16. Создать процедуру, увеличивающую на заданный процент заработную плату агентам, которые заключили наибольшее количество страховых договоров.

Написать функцию, возвращающую количество клиентов, у которых срок страхования подходит к концу. Вывести список клиентов и дату окончания контракта.

Вариант 17. Написать процедуру, которая за указанный период определяет для указанного вида техники количество работ и выводит перечень выполненных работ. В качестве параметров передать идентификатор техники, начальную и конечную даты периода. Результаты занести в специальную таблицу.

Создать функцию, которая возвращает количество оборудования, находящегося в ремонте и выводит информацию о сроках его окончания.

Вариант 18. Написать процедуру, выводящую список экспонатов автора, у которого имеется не менее пяти работ. Если таких авторов несколько, то вывести их фамилии и количество экспонатов.

Создать функцию, которая возвращает количество выставок в заданном месяце (году).

Вариант 19. Создать процедуру, «переводящую» сотрудников заданного отдела в другие отделы этого же города. В каждый из отделов переводить приблизительно одинаковое по отношению к среднему значению количество сотрудников. Расформированный отдел удалить.

Написать функцию, подсчитывающую количество сотрудников, работающих в заданном отделе.

Вариант 20. Написать процедуру, выводящую список фильмов, в которых режиссер является одновременно исполнителем одной из главных ролей, с указанием фамилии режиссера и роли, которую он сыграл.

Создать функцию, которая возвращает продолжительность фильма в виде строки «X ч. Y м.» по значению числового поля «Продолжительность».

Вариант 21. Написать функцию, которая возвращает время начала и завершения экзамена или консультации. Использовать два входных параметра: дата сдачи и тип (0 – экзамен, 1 – консультация). Продолжительность экзамена – 5 ч, консультации – 2 ч.

Создать процедуру, которая выводит расписание сессии. Результат должен выглядеть приблизительно так:

Факультет_1
группа1
дисциплина 1 преподователь конс. (дата, время, ауд.) экз. (дата, время, ауд.)
дисциплина N преподаватель конс. (дата, время, ауд.) экз. (дата, время, ауд.)
группа2
дисциплина 1 преподователь конс. (дата, время, ауд.) экз. (дата, время, ауд.)
дисциплина N преподаватель конс. (дата, время, ауд.) экз. (дата, время, ауд.)
…
Факультет_2
…

Вариант 22. Создать процедуру, обеспечивающую удаление поставщика из таблицы по указанному названию и городу. Накладные удаленного поставщика перенести во вспомогательную таблицу.

Написать функцию, которая возвращает наиболее востребованный за последний месяц вид ГСМ.

Вариант 23. Написать процедуру, которая принимает в качестве параметра номер выставочного места и выводит список всех предприятий или товаров, принимающих участие в выставке.

Создать функцию, возвращающую количество дней, прошедших между датой последней выставки и сегодняшней датой. Если первая дата больше второй, функция возвращает –1.

Вариант 24. Написать процедуру, выводящую информацию о контрактах, в которых сумма заказа лежит в диапазоне ±50 дол. США от введенного значения. Если контрактов с такой суммой не имеется, должно выводиться соответствующее сообщение.

Создать функцию, подсчитывающую количество заключенных контрактов за текущий год. В вызывающую среду возвращать общую сумму этих контрактов в параметре OUT.

Вариант 25. Создать процедуру изменения стоимости медицинской техники. Входные параметры – id медицинской техники и новая стоимость. Подсчитать и вывести, на сколько процентов изменилась стоимость.

Написать функцию, подсчитывающую количество и сумму контрактов медицинской техники заданного производителя. Если указанного производителя нет, то добавить его.

Вариант 26. Создать процедуру, копирующую строки с информацией об участках, нарядов на которые в указанном месяце не было, во вспомогательную таблицу. Вывести количество таких участков.

Создать функцию, возвращающую количество работников, которые получили наименьшее количество нарядов до указанной в параметре даты. Вывести более подробную информацию об этих работниках.

Вариант 27. Создать процедуру, копирующую строки с информацией о строительстве в текущем месяце во вспомогательную таблицу. Подсчитать количество извлеченных строк.

Создать функцию, подсчитывающую, сколько этапов выполнено по каждому объекту. Вернуть количество объектов, по которым завершены все этапы.

Вариант 28. Написать в технологических картах процедуру замены на текущий месяц заболевшего рабочего на другого, который меньше других загружен.

Создать функцию, возвращающую количество произведенных изделий в текущем месяце. Входной параметр функции – идентификатор изделия.

Вариант 29. Написать процедуру определения занятости указанного рабочего за текущий год в процентах от общего количества выполненной работы.

Создать функцию, определяющую количество материалов, в среднем расходующихся в месяц в цехе.

Вариант 30. Создать процедуру, обеспечивающую удаление договоров, заключенных на фамилию клиента, указанную в качестве параметра.

Написать функцию, которая за указанный период определяет количество заключенных договоров. В качестве параметров передавать начальную и конечную даты периода, контролировать правильность введенных значений.

Вариант 31. Создать процедуру, которая заменяет сырье в еще не выполненных контрактах и пересчитывает их общую стоимость. В качестве параметров передавать новое сырье и сырье, которое надо заменить.

Написать функцию, которая определяет сумму подписанных контрактов за указанный месяц. В качестве параметра передать название месяца в текстовом виде.

Вариант 32. Создать процедуру, которая при передаче в качестве параметра, например, 1, заполняет таблицу проведенных работ профилактическими осмотрами всех хозяйств (в день не более двух хозяйств), а при получении в качестве параметра, например, 2, подсчитывает количество работ за текущий месяц.

Написать функцию, которая определяет, работает тот или иной ветеринар на ветеринарной станции (путем возвращения TRUE или FALSE), а в параметре OUT по выходе из функции записывает его номер телефона.

Вариант 33. Создать процедуру, обеспечивающую удаление клиента из таблицы по указанным имени и фамилии, а все его договора перемещает в архив.

Написать функцию, которая определяет наиболее загруженную бригаду и, если есть незанятая на текущий момент времени бригада, то распределяет задачи между ними. Функция возвращает ФИО бригадира загруженной бригады, а также выводит сообщение о том, какие проведены перестановки.

Вариант 34. Создать процедуру, обеспечивающую вывод информации о всех аудиториях, которые свободны в указанный в качестве параметра день.

Написать функцию, которая подсчитывает количество часов занятий в неделю для указанной группы.

Вариант 35. Создать процедуру изменения стоимости изделий. Входные параметры – идентификатор изделия и новая стоимость.

Создать функцию, возвращающую идентификатор изделия, имеющего наибольший спрос.

Вариант 36. Создать процедуру, обеспечивающую удаление клиентов из таблицы, если они не обращались в ателье более года.

Создать функцию, возвращающую название изделия, имеющего наименьший спрос.

Вариант 37. Написать процедуру переноса данных об услугах, оказанных выбывшим клиентам, в архив (вспомогательную таблицу). Если в один день клиенту было оказано несколько одинаковых услуг, в архивную таблицу добавлять их одной строкой, указывая количество.

Создать функцию, возвращающую количество дней, прожитых постояльцем в гостинице на основании двух дат (день приезда и день отъезда считать как один день). Если второй параметр не определен, считать до текущей даты.

Вариант 38. Создать процедуру, переносящую информацию о заказах, поступивших в указанный день, во вспомогательную таблицу, подсчитывать их общее количество.

Написать функцию, которая подсчитывает среднее количество работ по переданной в качестве параметра должности специалиста.

Вариант 39. Создать процедуру изменения стоимости услуги. Входные параметры – идентификатор услуги и новая стоимость. Определить насколько (в процентах) изменилась стоимость услуги.

Создать функцию, которая возвращает ФИО врачей, у которых нет пациентов на указанную в параметре дату; если указан выходной день, то сообщать об этом.

Вариант 40. Создать процедуру, подсчитывающее количество комплектующих в заданном изделии. Выводить изделие и количество комплектующих в процентах.

Написать функцию, возвращающую количество изделий, сборка которых на данный момент не завершена.

Вариант 41. Написать процедуру изменения адреса поставщика по указанному названию. Адрес не должен повторяться. Старый адрес записывать во вспомогательную таблицу с указанием текущей даты.

Написать функцию, которая за указанный период определяет количество подписанных контрактов. В качестве параметра передать начальную и конечную даты периода.

Вариант 42. Создать процедуру, которая проверяет, выполняется ли план производства за месяц. Значение плана передавать в процедуру в виде общего количества деталей.

Написать функцию, которая вычисляет общее время технологических процессов для указанного типа кабеля.

Вариант 43. Создать процедуру, которая обеспечивает вывод информации обо всей технике, которая не работает на текущую дату и подсчитывает время простоя.

Написать функцию, которая определяет, работал ли указанный водитель на указанной технике, а в параметре OUT по выходе из функции записывает количество раз.

Вариант 44. Создать процедуру, переносящую информацию об отдыхающих с указанным в качестве аргумента заболеванием во вспомогательную таблицу.

Создать функцию, возвращающую количество дней, прожитых отдыхающим в санатории, на основании двух дат (день приезда и день отъезда считать как один день). Если второй параметр не определен, считать до текущей даты.

Вариант 45. Создать процедуру, переносящую информацию о геологах, участвовавших в определенной экспедиции, во вспомогательную таблицу. Аргументом является название экспедиции.

Написать функцию, которая определяет, сколько экспедиций было по указанной карте (региону).

Вариант 46. Создать процедуру, которая подсчитывает количество договоров у персонала и на основании этой информации вычисляет их заработную плату. Результаты вычисления занести в новую таблицу.

Написать функцию, которая возвращает процент скидки клиента в зависимости от суммы предыдущих договоров (предусмотреть не менее трех позиций).

ПРИЛОЖЕНИЕ 6

ВАРИАНТЫ ЗАДАНИЙ ДЛЯ НАПИСАНИЯ ТРИГГЕРОВ

Вариант 1

1) Запретить сотрудникам одновременно работать более чем на 1,5 ставки. Если при вставке или обновлении контракта размер ставки превышает 1,5, то операция не выполняется, а в сообщении указывается, сколько составляет превышение.

2) Следить, чтобы руководить кафедрой мог только один сотрудник с научной степенью, а количество сотрудников кафедры не превышало максимально возможное, которое задается при описании кафедры.

3) Отслеживать за месяц срок окончания контрактов сотрудников.

Вариант 2

1) Реализовать политику премирования продавцов в зависимости от количества или суммы продаж в месяц.

2) Управлять количеством товара, имеющегося в наличии при осуществлении продажи (вычитать количество купленного, запрещать продажи с недостающим количеством или предлагать купить остаток и т. д.), рассчитывать общую стоимость каждой покупки с учетом наценки магазина.

3) В определенное время сообщать о наличии продавцов, которые не продали за день (неделю, месяц) ни одного товара.

Вариант 3

1) Осуществлять контроль загруженности почтальонов, например, по количеству одновременно обслуживаемых подписчиков.

2) Осуществлять расчет стоимости подписки, при этом предусматривать, чтобы нельзя было подписаться менее чем на месяц и более чем на год, а дата оформления подписки должна быть не позднее 10 дней до ее начала.

3) Каждый месяц перемещать подписки с законченным сроком в архив.

Вариант 4

1) Расформировать отдел, перевести продавцов в другие отделы так, чтобы среднее количество продавцов в них было приблизительно одинаковым.

2) Отслеживать наличие товара в необходимом количестве, изменять это количество при покупке или возврате, рассчитывать общую сумму продажи.

3) Осуществлять «скидочную» политику на товары, которые не были проданы в течение месяца (года). Сохранять во вспомогательной таблице старую цену и дату ее изменения.

Вариант 5

1) Не превышать заданное количество сотрудников в отделах, предлагать (выводить) свободные места по другим отделам, а информацию о «лишних» сохранять во вспомогательной таблице.

2) Не допускать в штатном расписании наличие записей о сотрудниках, работающих более чем на 1,5 ставки одновременно.

3) Начислять на определенную дату заработную плату всем работающим сотрудникам.

Вариант 6

1) Реализовать ценовую политику при продаже пиломатериалов в зависимости от их количества, рассчитывать общую стоимость продажи с учетом количества товаров и стоимости сырья.

2) При недостаточном количестве сырья или изделий сохранять информацию о несостоявшейся продаже во вспомогательной таблице «Отложенная продажа».

3) Автоматически пополнять наличие товара на определенное количество раз в неделю и проверять возможность выполнения отложенных продаж.

Вариант 7

1) Контролировать расход материалов при создании деталей, не создавать запись в технологической карте при недостаточном количестве необходимого материала.

2) Увольнять рабочего, только если все процессы с его участием завершены, переносить их во вспомогательную таблицу, указывая ФИО работника и дату его увольнения.

3) В определенный день составлять список материалов, которые заканчиваются (например, количество меньше, чем нужно на производство минимальной партии деталей).

Вариант 8

1) Отслеживать количество пациентов, закрепленных за врачом одновременно, не допускать превышения установленного значения, которое зависит от категории и стажа специалиста.

2) При госпитализации контролировать занятость койко-мест, тип палат (мужская, женская) и срок нахождения пациента в больнице (не более 14 дней).

3) Каждый день обновлять информацию о количестве пациентов в отделении и количестве свободных мест по палатам.

Вариант 9

1) Отслеживать наличие руководителя и количество человек в бригаде, чтобы оно было меньше заданного в триггере минимального значения и не превышало заданный максимум.

2) Отслеживать, чтобы на одной единице оборудования в день не выполнялось больше технологических процессов, чем это установлено в таблице с описанием оборудования.

3) Обновлять максимальное и минимальное количество партий деталей, произведенных в день (месяц).

Вариант 10

1) Отслеживать количество единиц товаров на складе при совершении оптовых продаж. Выдавать сообщение о необходимости пополнения склада, если оставшееся количество товара меньше, чем его среднее значение по продажам за последний месяц.

2) Организовать слежение за минимальным оптовым количеством товара в зависимости от его вида и производителя.

3) Создать таблицу-отчет, которая должна будет обновляться один раз в неделю или в месяц и содержать данные о производителе, количестве проданных единиц товаров, сумме продаж.

Вариант 11

1) Отслеживать загруженность оборудования по часам в актах работ, не допускать одновременную работу на одном оборудовании, работу в ночные часы и работу без перерывов.

2) Отслеживать, чтобы руководитель исследовательских работ был закреплен не более чем за тремя работами в месяц.

3) Архивировать информацию об оборудовании, подлежащем списанию, во вспомогательную таблицу, где указывать общее количество актов, количество отработанных часов, дату начала и конца работ на каждой единице оборудования.

Вариант 12

1) Рассчитывать общую сумму стоимости по договору, делая скидку клиентам в зависимости от количества или общей суммы их заказов.

2) Контролировать количество материала и продукции, имеющихся в наличии при заключении договора, не допускать одновременного оказания одних и тех же услуг нескольким клиентам.

3) Ежедневно вести учет договоров, по которым истечение сроков исполнения произойдет менее чем через три дня.

Вариант 13

1) Рассчитывать сумму, которая должна быть уплачена риэлтору в зависимости от типа сделки, и включать ее в общую сумму оплаты.

2) Помечать специальным символом недвижимость, по которой была совершена сделка, или переносить ее из основной таблицы в архив.

3) Каждый день обновлять таблицу-отчет, содержащую тип недвижимости, тип сделки, количество объектов, сумму сделок, дату.

Вариант 14

1) Автоматически формировать дату окончания обслуживания клиента в соответствии с гарантийными сроками на выбранное ПО; не допускать записей о гарантийных ремонтах после окончания гарантийного обслуживания.

2) При удалении поставщика информация о нем переносится в архив вместе с описанием его ПО и количеством заключенных сделок на покупку. Если есть сделки, по которым на момент удаления не завершено гарантийное обслуживание, удаление запретить.

3) Регулярно обновлять список постоянных клиентов (статус присваивается при покупке товаров на определенную сумму и аннулируется, если в течение года нет обращений).

Вариант 15

1) При оформлении технологической карты контролировать рабочее время: понедельник – пятница с 8:00 до 17:00 (обед с 12:30 до 13:30) для оборудования и для рабочих, не допускать одновременной работы (график можно формировать самостоятельно, но не более 40 ч в неделю для каждого рабочего).

2) Следить за количеством инструмента на складе, его износом и соответствием определенному оборудованию при формировании карт.

3) Пополнять склад инструментов, если его количество достигает установленных минимальных значений.

Вариант 16

1) При заполнении информации о договоре автоматически вносить текущую дату как начальную, если она не указана агентом, и дату окончания страховки в зависимости от типа и срока страхования, рассчитывать общую стоимость и первоначальный взнос.

2) Следить за суммами выплат, начислять пеню за просроченные выплаты.

3) Переносить в архив старые договора, указывая сумму страховки и выплаты по страховым случаям, если они были.

Вариант 17

1) Техника, находящаяся в ремонте, не должна быть задействована в других работах; на одной единице техники нельзя работать одновременно.

2) Предусмотреть, чтобы каждая единица техники проходила плановый ремонт в соответствии с установленным графиком.

3) Обновлять информацию во вспомогательной таблице о часах работы, планового и внепланового ремонта по каждой единице оборудования.

Вариант 18

1) Запретить выставлять один экспонат на две выставки одновременно; выставлять экспонат, находящийся на реставрации.

2) Контролировать количество экспонатов на каждой выставке.

3) Если выставка закончилась, то во вспомогательной таблице обновлять «коэффициент выставляемости» по каждому экспонату и по каждому автору.

Вариант 19

1) Рассчитывать заработную плату сотрудников в зависимости от оклада и процента участия в разработках.

2) Отслеживать, чтобы сотрудник одновременно работал не более чем на 1,5 ставки и совмещал не более трех должностей, также он не может работать одновременно более чем над тремя разработками и более чем в двух отделах.

3) Во вспомогательную таблицу регулярно вносить информацию о ведущихся в отделе разработках с указанием количества участвующих сотрудников, датах начала и окончания.

Вариант 20

1) Отслеживать, чтобы в БД было не более трех фильмов одного режиссера или актера с одинаковым годом выпуска.

2) Предусмотреть расчет рейтинговых баллов актеров в зависимости от количества или сумм продаж фильма в месяц.

3) Архивировать неповторяющуюся информацию о фильмах, продажи которых не возобновлялись в течение года, при этом рассчитывать и сохранять количество лет и месяцев в продаже (интервальный тип), общую сумму продаж.

Вариант 21

1) При формировании дат сдачи сессии контролировать занятость студентов и преподавателей, количество дней между экзаменами и количество экзаменов в сессию.

2) Проверять, сданы ли студентом необходимые зачеты; не допускать студента до следующего экзамена, если количество неудовлетворительных оценок в текущую сессию достигло двух.

3) После сдачи последнего экзамена изменить в дополнительной таблице статус студента: переведен (не переведен) на курс №__ и установить процент повышения (понижения) стипендии.

Вариант 22

1) При заполнении накладной устанавливать текущую дату, рассчитывать общую стоимость, проверять количество топлива в наличии.

2) Пополнять количество каждого вида топлива на складе, но не превышать максимально возможное общее количество на станции.

3) Заносить в дополнительную таблицу сведения об объеме топлива и общей сумме по каждому поставщику.

Вариант 23

1) При вставке данных в таблицу «Организация выставки» проверять, не занято ли уже выставочное место, запрещать отдавать более 20 % мест на выставке одному предприятию.

2) Контролировать даты проведения выставок: количество дней выставки не менее трех, время между выставками не менее двух дней и т. д.

3) В дополнительной таблице обновлять информацию о количестве выставок по месяцам.

Вариант 24

1) При изменении цен на материалы пересчитывать стоимость услуг, для которых они необходимы.

2) Не заключать в месяц более определенного количества контрактов, делать скидку заказчику при повторном обращении, рассчитывать автоматически общую стоимость контракта.

3) Вести в дополнительной таблице учет занятости оборудования по дням недели.

Вариант 25

1) При заключении контракта проверять и изменять наличие медтехники на складе.

2) Рассчитывать стоимость контракта с учетом скидки для постоянных клиентов.

3) Вести общий учет количества проданных товаров по поставщикам.

Вариант 26

1) При формировании наряда проверять занятость работников и проведение других мероприятий на участке в одно время.

2) Контролировать количество работников по должностям и их общее количество.

3) Рассчитывать заработную плату работников по итогам месяца.

Вариант 27

1) Отслеживать последовательное выполнение этапов по каждому объекту строительства с учетом сроков на выполнение.

2) Контролировать наличие материалов, занятость бригад и сроки строительства; не допускать одновременного строительства более трех объектов.

3) Если последний этап строительства завершен, то сохранить данные в дополнительной таблице «Построенные объекты», где указывать данные об объекте и количество дней, потраченных на строительство.

Вариант 28

1) При составлении технологических карт контролировать занятость оборудования и рабочих, не загружать оборудование больше 8 ч в день.

2) В зависимости от количества изготавливаемых деталей устанавливать время на их изготовление в технологической карте.

3) Вести подсчет деталей, изготавливаемых на каждой единице оборудования в месяц.

Вариант 29

1) Вести учет рабочего времени в цеху (время начала и конца работы в соответствии с установленным графиком, для рабочего не более 8–12 ч в день).

2) При изготовлении деталей контролировать количество материалов и оснастки на складе, квалификацию рабочего.

3) Вести учет изготовленных деталей на складе.

Вариант 30

1) Для выбранной конфигурации установить и контролировать минимальные требования к оборудованию при заключении контракта.

2) Отслеживать чтобы у одного исполнителя не было одновременно двух договоров и более пяти договоров в день всего.

3) При удалении клиента архивировать в дополнительную таблицу информацию о нем и статистику по договорам.

Вариант 31

1) Контролировать имеющиеся в наличии количества сырья и продукции при заключении контракта.

2) Вести скидочную политику для постоянных клиентов фирмы.

3) Протоколировать количество проданного сырья и стройматериалов.

Вариант 32

1) Рассчитывать цену обслуживания с учетом стоимости работы и медикаментов: делать скидку 10 %, если хозяйство обслуживается чаще двух раз в месяц.

2) Контролировать расход медикаментов на проведенные работы.

3) В конце месяца автоматически формировать в проведенных работах записи о профилактических осмотрах обслуживаемых хозяйств.

Вариант 33

1) Отслеживать, чтобы одна бригада работала одновременно только с одной скважиной.

2) Контролировать по параметрам соответствие скважины и оборудования для ее бурения при заключении контракта.

3) Списывать устаревшее оборудование и архивировать общий метраж пробуренных им скважин, дату списания.

Вариант 34

1) Контролировать занятость аудиторий, группы и преподавателей при любом изменении в таблице загрузки аудиторий.

2) Разрешить назначать пару только с 8:00 до 22:00 и учитывать соответствие типа аудиторий и проводимых занятий.

3) Считать общее количество часов занятий по кафедрам за семестр в дополнительной таблице.

Вариант 35

1) При создании договора на сбыт проверять наличие изделий на складе. Если изделий не хватает, то сообщать об этом и разрешать вставку только при условии, что дата поставки не раньше, чем через две недели от текущей даты. Если изделий достаточно, то изменять количество товара на складе.

2) Организовать систему скидок для постоянных клиентов при расчете стоимости.

3) В дополнительной таблице обновлять информацию об изменениях в ассортименте фабрики.

Вариант 36

1) Рассчитывать стоимость квитанции и контролировать расход материалов на заказанные изделия.

2) Не брать заказов больше, чем можно выполнить в ателье за месяц с учетом времени пошива на изделие.

3) Вести статистику о количестве заказов по кварталам.

Вариант 37

1) Контролировать занятость номера и не заселять нового клиента в день выселения предыдущего; рассчитывать стоимость проживания.

2) Отслеживать, чтобы за одним представителем персонала не было закреплено более пяти номеров одновременно.

3) В дополнительной таблице фиксировать количество проживающих в каждом номере и общую сумму оплаты за месяц.

Вариант 38

1) Контролировать соответствие мероприятия квалификации выбираемого специалиста, а также зону обслуживания.

2) При оформлении заказа следить, чтобы работник был свободен на дату исполнения; если он занят, то выводить сообщение о дате, когда он свободен.

3) При увольнении сотрудника разрешать переносить его в архив только с датой завершения последнего заказа (не ранее).

Вариант 39

1) Запрещать запись клиентов в одно и то же время к одному и тому же врачу.

2) Предусмотреть систему изменения цен на услуги в рамках заданного интервала значений и времени.

3) Уволить врача, учитывая дату последнего обслуживания, и сохранить информацию о количестве обслуженных пациентов, количестве оказанных услуг и общей сумме оплат.

Вариант 40

1) Контролировать последовательность и количество операций для сборки любого изделия.

2) Следить за наличием комплектующих и не собирать одновременно более трех изделий.

3) Фиксировать время обновления количества комплектующих в дополнительной таблице.

Вариант 41

1) При заключении контракта контролировать количество изделий и материалов в наличии, сроки изготовления, рассчитывать цену контракта.

2) Контролировать возможность изготовления изделий только из определенных материалов.

3) Если материал отсутствует на складе, то в дополнительной таблице записывать заказ на него, дату, требуемое количество и возможного поставщика.

Вариант 42

1) При производстве контролировать наличие материалов, занятость рабочих и даты изготовления.

2) Контролировать соответствие технологических процессов, их количества и последовательности определенным видам кабеля.

3) Записывать во вспомогательную таблицу данные о произведенной продукции за день.

Вариант 43

1) При формировании путевки контролировать занятость техники и водителей, делать перерывы между работами по установленному графику.

2) График работы водителей должен быть с 8:00 до 18:00 по схеме «два через два» и на определенной технике.

3) Контролировать расход топлива и километраж, пройденный каждой единицей техники в дополнительной таблице.

Вариант 44

1) Контролировать количество одновременно отдыхающих в соответствии с количеством мест в санатории, а также количество отдыхающих, закрепленных за одним врачом.

2) Создать систему соответствий и противопоказаний между заболеваниями и оздоровительными программами, следить за правильностью назначений.

3) В отдельной таблице обновлять сведения о количестве дней, проведенных в санатории каждым отдыхающим.

Вариант 45

1) Контролировать, чтобы в экспедиции было не менее трех геологов, и все они не были задействованы в других экспедициях в данное время. При отсутствии свободных геологов данные о предполагаемой экспедиции заносить во вспомогательную таблицу.

2) Подбирать карты для выбранного маршрута с учетом региона.

3) Ежедневно проверять вспомогательную таблицу с предполагаемыми экспедициями и при наличии освободившихся от работы или новых геологов переносить данные в основную.

Вариант 46

1) Делать скидку клиенту при повторном обращении.

2) При составлении договора контролировать занятость персонала, стоимость работ, даты оказания услуги.

3) Ежедневно переносить во вспомогательную таблицу записи о договорах, которые должны быть завершены на текущей неделе.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Зудилова, Т. В. SQL и PL/SQL для разработчиков СУБД ORACLE / Т. В. Зудилова, С. Е. Иванов, С. Э. Хоружников. – СПб. : НИУ ИТМО, 2012. – 74 с.

2. Бондаренко, С. П. Практикум по курсу «Модели данных и СУБД» : учеб. пособие / С. П. Бондаренко, А. Н. Исаченко. – Минск : БГУ, 2005. – 103 с.

3. Капанов, Н. А. Базы данных САПР. Лабораторный практикум / Н. А. Капанов. – Минск : БГУИР, 2006. – 47 с.

4. Коннолли, Т. Базы данных: проектирование, реализация и сопровождение. Теория и практика / Т. Коннолли, К. Бегг. – 3 изд., перераб. – М. : Вильямс, 2003. – 1440 с.

5. Гринвальд, Р. Oracle. Справочник / Р. Гринвальд, Д. К. Крейнс ; пер. с англ. – СПб. : Символ-Плюс, 2005. – 976 с.

6. Скотт, У. Oracle Database 10g. Программирование на языке PL/SQL / У. Скотт, Р. Хардман, М. МакЛафлин. – М. : Лори, 2007. – 816 с.

7. Кригель, А. SQL. Библия пользователя. Язык запросов SQL / А. Кригель, Б. Трухнов. – 2-е изд. – М. : Диалектика, 2009. – 752 с.

8. Кайт, Т. Oracle для профессионалов. В 2 т. / Т. Кайт ; пер. с англ. – СПб. : ООО «ДиаСофтЮП», 2005. – Т. 1 – 672 с. ; – Т. 2 – 816 с.

9. Кайт, Т. Эффективное проектирование приложений / Т. Кайт ; пер. с англ. – М. : Лори, 2006. – 656 с.

10. Кайт, Т. Oracle для профессионалов. Архитектура, методики программирования и особенности версий 9i, 10g и 11g / Т. Кайт ; пер. с англ. – М. : Лори, 2011. – 842 с.

11. Кристофер, А. Oracle PL/SQL. Как писать мощные и гибкие программы на PL/SQL / А. Кристофер. – М. : Лори, 2005. – 369 с.

12. Фейерштейн, С. Oracle PL/SQL для профессионалов / С. Фейерштейн, Б. Прибыл. – 3-е изд. – СПб. : Питер, 2004. – 941 с.

13. Гринвальд, Р. Oracle 11g. Основы / Р. Гринвальд, Р. Стаковьяк, Дж. Стерн. – 4-е изд. – СПб. : Символ-Плюс, 2009. – 1016 с.

14. Прайс, Дж. Oracle Database 11g. Операторы SQL и программы PL/SQL / Дж. Прайс. – СПб. : Лори, 2014. – 690 с.

Св. план 2019, поз. 23

Учебное издание

Крупская Марина Александровна

ИНФОРМАЦИОННОЕ ОБЕСПЕЧЕНИЕ СИСТЕМ УПРАВЛЕНИЯ. ЛАБОРАТОРНЫЙ ПРАКТИКУМ

УЧЕБНО-МЕТОДИЧЕСКОЕ ПОСОБИЕ

Редактор *М. А. Зайцева*
Корректор *Е. Н. Батурчик*
Компьютерная правка, оригинал-макет *Е. Г. Бабичева*

Подписано в печать 02.09.2019. Формат 60×84 1/16. Бумага офсетная. Гарнитура «Таймс».
Отпечатано на ризографе. Усл. печ. л. 8,72. Уч.-изд. л. 9,3. Тираж 70 экз. Заказ 107.

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