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

Кафедра систем управления

Н. А. Капанов

БАЗЫ ДАННЫХ САПР

Лабораторный практикум

для студентов специальности
I-53 01 07 «Информационные технологии и управление
в технических системах»

Минск 2006

УДК 004.65
ББК 32.973-018.2
К 20

Р е ц е н з е н т :
доцент кафедры ИТАС БГУИР,
канд. техн. наук О. В. Герман

Капанов, Н. А.

К 20 Базы данных САПР : лаб. практикум для студ. спец. I-53 01 07 «Информационные технологии и управление в технических системах» / Н. А. Капанов. – Мн. : БГУИР, 2006. – 47 с. : ил.

ISBN 985-488-055-9

Цель настоящего издания – оказание помощи студентам специальности «Информационные технологии и управление в технических системах» всех форм обучения при выполнении ими лабораторных работ по курсу «Базы данных САПР».

Данный материал содержит теоретические сведения, необходимые для практических занятий, с целью отметить характерные особенности построения и функционирования информационных систем с базами данных и систематизировать справочный материал, что может быть полезно при разработке информационных систем.

УДК 004.65
ББК 32.973-018.2

ISBN 985-488-055-9

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Назначение и возможности баз данных

Базы данных давно стали неотъемлемой частью нашей научно-производственной деятельности. Что же такое *базы данных* и почему теперь они нашли столь широкое применение? В литературе можно найти различные определения этого понятия, однако по существу все источники определяют базу данных как централизованный набор логически связанных данных, т. е. *база данных – это единое, большое хранилище данных, которое используется одновременно многими пользователями из разных подразделений организации (централизованность) и полностью удовлетворяет их информационные нужды (логическая связанность).* Вместо разрозненных файлов с избыточными данными все данные в базе собраны вместе с минимальной долей избыточности. База данных не принадлежит какому-либо единственному отделу, а является общим корпоративным ресурсом, причем в ней хранятся не только рабочие данные, но и их описания. По этой причине *базу данных* еще называют *набором интегрированных записей с самоописанием*. Такой подход позволяет отделить структуру данных от приложений, их обслуживающих.

Очевидно, что для поддержки баз данных, а именно обеспечения связанности данных в ней, а также их однозначности и целостности существует специальное программное обеспечение, называемое *системой управления базой данных*. Такое программное обеспечение взаимодействует с прикладными программами пользователя и позволяет определять базу данных, т.е. предоставлять пользователю средства указания типа данных и их структуры, а также средства задания ограничений для хранимой в базе информации. Помимо этого система управления базой данных (СУБД) позволяет вставлять, удалять, обновлять и извлекать из базы данных необходимую пользователю информацию, включая и ту, для получения которой необходима обработка перекрестных связей между данными и поставленных ограничений.

Преимуществами СУБД по сравнению с традиционными, ныне устаревшими файловыми системами являются:

1. Контроль за избыточностью данных.
2. Соблюдение непротиворечивости данных как следствие централизации и минимальной избыточности данных в базе.
3. Совместное использование данных, поддержка целостности за счет предоставления возможности задания ограничений на данные при создании базы.
4. Повышенная безопасность.
5. Повышение эффективности с ростом масштабов организации, т. е. рациональное использование бюджета.
6. Улучшение показателей производительности.

Лабораторная работа № 1

Простые SQL-запросы

Цель работы – изучить назначение простых запросов SQL, возможностей задания, а также ограничений задания запросов для безошибочной их обработки СУБД.

- ***Теоретические сведения***

Описание учебной базы данных

Для выполнения приложений баз данных необходимо помимо знаний специфичных интерфейсов и навыков их программирования представлять себе структуру самой базы данных. Под *структурой базы данных* здесь понимается *набор именованных и взаимосвязанных таблиц, их атрибутов, а также возможных именованных ограничений на значения атрибутов вышеуказанных таблиц.* Поэтому знание структуры базы данных предполагает представление о предназначении каждой из таблиц базы данных, о типе данных и предназначении каждого из атрибутов (столбцов) таблиц, знание возможных ограничений на значения столбцов (условий уникальности, условий на значение, допустимость значений типа NULL и т. д.), а также зачастую именованных ограничений ссылочной целостности [1].

Особенностью большинства современных СУБД является то , что они позволяют как определять базу данных с помощью **языка определения данных** (в дальнейшем DDL – data definition language), так и добавлять, изменять и извлекать информацию из базы данных посредством **языка управления данными** (в дальнейшем DML – data manipulation language). Наиболее распространенным и стандартизированным языком управления и определения является **язык структурированных запросов** (в дальнейшем SQL – structured query language). Язык SQL в настоящее время поддерживается большинством СУБД с различными и незначительными отклонениями от стандарта, регламентирующего общие правила задания запросов. В этом случае можно говорить о “диалектах” SQL поддерживаемых конкретными СУБД. Заметим, что существенных отличий данных диалектов от стандарта SQL не имеется, так же, как и отсутствуют существенные различия между этими диалектами.

Главным отличием языка SQL от других языков программирования является его «непроцедурность», т.е. посредством SQL просто указывают, какая информация из базы данных необходима. Поэтому здесь становится очевидным тот факт, что программисту прикладных программ, использующих базу данных, необходимо очень четко представлять её структуру, а также свойства и возможности СУБД, которая поддерживает работоспособность базы данных.

Исходный материал: Для выполнения лабораторной работы студентам предлагается учебная база данных малого предприятия по аренде недвижимости. Спроектирована она по образцу, приведенному в [1].

Такая база данных содержит пять таблиц: Branch, Staff, Property_for_rent, Renter, Owner, Viewing.

Информационная нагрузка таблиц данной базы данных такова:

– табл. Branch предназначена для сохранения информации об отделениях (офисах) предприятия и оснащена следующими атрибутами:

bno	street	city	tel_no

Здесь bno является первичным ключом и в соответствии с правилом целостности сущности не способен принимать неопределенных значений. Предназначение остальных атрибутов не вызывает трудностей.

– табл. Staff предназначена для сохранения информации о сотрудниках и оснащена следующими атрибутами:

sno	fname	lname	address	tel_no	position	sex	dob	salary	bno

В данной таблице sno – первичный ключ, предназначенный для уникальной идентификации записей о сотрудниках; position – строковый атрибут, содержание которого определяет занимаемую должность; dob – атрибут типа даты с данными о днях рождения сотрудников; salary – числовой атрибут с зарплатой сотрудников. Атрибут bno – внешний ключ для связи с табл. branch.

– табл. Property_for_rent с информацией об объектах недвижимости, предлагаемых в аренду, имеет следующие атрибуты:

pno	street	city	type	rooms	rent	ono	sno	bno

Здесь pno – первичный ключ, type – строковый атрибут с информацией о типе предлагаемого объекта недвижимости; в данном случае на значения атрибута наложено ограничение, т.е. данный атрибут может принимать либо значение ’h’, либо ‘f’. Rooms и rent – числовые атрибуты, причем rent имеет смысл рентной стоимости объекта. Ono, sno, bno – внешние ключи таблицы для связи с табл. Owner, Staff, Branch соответственно.

– табл. Renter содержит информацию о потенциальных арендаторах и содержит следующие атрибуты:

rno	fname	lname	address	tel_no	pref_type	max_rent	bno

Здесь Rno – первичный ключ, 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 табл. Branch.

– табл. Owner определяет владельцев объектов недвижимости, которые сдаются в аренду:

ono	fname	lname	address	tel_no

В данной таблице ono является уникальным идентификатором (первичным ключом) строк таблицы.

– табл. Viewing содержит результаты осмотра арендаторами предполагаемых объектов аренды.

rno	pno	date	comment

Особенность данной таблицы – наличие составного первичного ключа, состоящего из атрибутов rno и pno. Каждый из них в отдельности является внешним ключом для связи с табл. Renter (кто из потенциальных арендаторов производил осмотр) и Property_for_rent (какой из объектов осматривался). Помимо этого, как видно из вышеприведенной диаграммы, данная таблица содержит атрибуты date, определяющий дату осмотра (типа даты), и comment – самый «длинный» строковый атрибут базы данных, предназначенный для сохранения сделанных потенциальным арендатором комментариев.

Запросы на извлечение информации

Инструкция SQL *select* извлекает информацию из базы данных и возвращает её в виде таблицы результатов запроса. Данная инструкция состоит из шести синтаксических единиц, называемых зачастую предложениями. 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* указывается список таблиц, которые содержат элементы данных, извлекаемые запросом.

Например, следующий запрос извлекает из таблицы Staff три столбца:

Вывести для каждого сотрудника имя, фамилию и занимаемую должность

select fname, lname, position
from staff**;**

Помимо этого в предложении *select* может содержаться и так называемый «вычисляемый столбец», 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 определяет название столбца в результирующей таблице.

Предложение ***where*** показывает, что в результаты запроса следует включать только некоторые строки. В данном предложении вслед за ключевым словом ***where*** следует *условие отбора*, которое и определяет, какие строки должны включаться в результирующий набор.

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

- сравнение;
- проверка на принадлежность диапазону;
- проверка на членство в множестве;
- проверка на соответствие шаблону;
- проверка на равенство значению NULL.

Приведем примеры использования различных условий отбора в предложении ***where***.

Пример 1
Найти служащих, родившихся до 1988 года
select fname, lname
from staff
where DOB<’01-Jan-88’;

Здесь следует обратить внимание на формат записи данных типа «дата». В различных СУБД формат записи даты неодинаков. Используемый в примере формат поддерживается в ORACLE и специфичен для него. Даты в ORACLE, так же как и строковые константы, заключаются в парные одинарные кавычки.

Пример 2
Найти служащих, родившихся в интервале времени с 1 октября 1963 по 31 декабря 1971
select fname, lname
from staff
where DOB ***between*** ’01-Oct-63’ ***and*** ’31-Dec-71’;

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

Пример 3

Вывести информацию об офисах, расположенных в Минске, Витебске и Бресте

select address, tel_no
from branch
where city ***in*** (‘Минск’, ‘Витебск’, ‘Брест’);

Проверка ***in*** не добавляет новых возможностей, так же как и ***between, and***, так как условие X ***in*** (A, B, C) полностью эквивалентно условию (X=A) ***or*** (X=B) ***or*** (X=C).

Пример 4

Вывести информацию о всех сотрудниках, фамилии которых начинаются на букву К

select lname, address, tel_no
from staff
where lname ***like*** ‘K%’

Здесь также следует обратить внимание на запись шаблона строки сравнения в условии ***like***. Указанный способ задания строки шаблона характерен для диалекта ORACLE и отличен от регламентированного стандартом. Символ ‘%’ замещает произвольную последовательность символов, а ‘_’ — одиночный символ. Строки-шаблоны так же, как и обыкновенные строки-константы, заключаются в парные одинарные кавычки.

Запросы с 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*** указать столбцы соединения.

Составить список всех сотрудников, работающих в Минском отделении

select fname, lname, position, S.tel_no
from Branch B, Staff S
where B.bno=S.bno ***and*** city = ‘Минск’;

Тот же запрос можно выполнить с помощью подзапроса:

select fname, lname, position, tel_no
from staff ***where*** bno ***in*** (***select*** bno ***from*** branch ***where*** city= ‘Минск’);

В связи с подчиненными запросами можно выделить ряд особенностей:

— таблица результатов подчиненного запроса всегда состоит из одного столбца;

— в подчиненный запрос не может включаться предложение *order by*;

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

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. 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*() – для вычисления суммы всех значений столбца-аргумента;

— *avg*() – для вычисления среднего значения столбца;

— *min*() – определяет минимальное значение столбца;

— *max*() – определяет максимальное значение столбца;

— *count*() – подсчитывает число всех определенных значений столбца;

— *count*(*) – подсчитывает число строк таблицы.

Определить, сколько в среднем получают сотрудники в зависимости от занимаемой ими должности

```
select position, avg(salary)
from staff
group by position;
```

Подсчитать количество сотрудников, работающих в каждом из офисов, исключив офисы, в которых работает менее 2 человек

```
select bno, count(sno)
from staff
group by bno;
having count(sno)>2;
```

Представления

Представление – объект базы данных, представляющий собой именованный и сохраненный запрос. Часто представления также называют «виртуальными таблицами». В случае если определение представления простое, СУБД выполняет его «на лету», в противном случае СУБД приходится «материализовать» представление, т.е. сохранять его результаты

во временной таблице. Создаются представления посредством инструкции *create view* . Использование данной инструкции иллюстрируется примером.

Создать представление, включающее в себя список сотрудников, работающих в отделениях Минска

```
create view Minsk as select fname, lname, address, position, tel_no, sex, dob
from staff
where bno in (select bno
              from branch
              where city='Минск');
```

- ***Варианты заданий***

1. Получить список сотрудников с зарплатой от 200 до 300 тыс. рублей.

Получить список сотрудников, работающих в офисах Бреста и Гомеля.

Определить суммарную и среднюю зарплату сотрудников в зависимости от занимаемой ими должности.

Создать представление с информацией об офисах в Бресте.

2. Определить адреса и телефоны офисов, расположенных в Минске и Гродно.

Получить список сотрудников, предлагающих для аренды 3-комнатные квартиры.

Вывести итоговый отчет о средней и суммарной зарплатах в зависимости от половой принадлежности сотрудников.

Создать представление с информацией о директорах отделений.

3. Определить адреса всех 3-комнатных квартир, предлагаемых в аренду.

Получить список арендаторов, осматривавших объекты аренды 20 октября 1999 года.

Определить минимальную и максимальную зарплаты сотрудников различных отделений.

Создать представление с информацией о владельцах, чьи дома или квартиры осматривались потенциальными арендаторами.

4. Вывести номера домашних телефонов всех директоров.

Составить список владельцев всех 3-комнатных квартир.

Подсчитать количество сотрудников в каждом из отделений.

Создать представление сотрудниках и объектов, которые они предлагают в аренду.

5. Вывести список сотрудников, родившихся до 1980 года.

Подсчитать количество сотрудников, работающих в отделении в Бресте.

Вывести количество арендаторов, желающих арендовать 3- и 4-комнатные квартиры.

Создать представление об объектах с минимальной рентной стоимостью.

6. Определить адреса всех квартир с рентной стоимостью не более 300 тыс. рублей.

Подсчитать количество менеджеров, работающих в Минске.

Получить итоговый список с количеством домов и квартир, сдаваемых в аренду.

Создать представление о арендаторах, желающих арендовать 3-комнатные квартиры.

Создать представление об отделении с максимальным количеством работающих сотрудников.

7. Вывести домашние телефоны всех потенциальных арендаторов, желающих арендовать дома.

Вывести телефоны владельцев, дома или квартиры которых осматривались 12 сентября 2001 года.

Определить квартиры и дома минимальной рентной стоимости.

Создать представление о женщинах-директорах.

8. Вывести список всех женщин-менеджеров.

Определить максимальную зарплату сотрудников в отделении в Гродно.

Определить количество осмотров с группировкой по датам.

Создать представление о количестве сделанных осмотров с комментариями.

9. Определить количество объектов, осмотренных потенциальными арендаторами за октябрь 1996 года.

Создать список сотрудников, предлагающих объекты недвижимости в Минске.

Определить суммарную рентную стоимость объектов в Минске и Гродно.

Создать представление о сотрудниках, чьи фамилии начинаются с буквы ‘О’.

10. Создать список арендаторов, желающих снять 4-комнатные квартиры.

Определить количество потенциальных арендаторов, осмотревших предлагаемые им квартиры или дома.

Определить, какие из офисов имеют более 3-х сотрудников.

Создать представление, содержащее информацию об отделении, где предлагаются в аренду самые недорогие 2-комнатные квартиры в смысле их средней стоимости.

- ***Контрольные вопросы***

1. Как вы понимаете значение NULL?
2. Какова общая структура запроса на извлечение информации?
3. Перечислите особенности итоговых запросов.
4. Что такое представление и для чего создаются такие объекты базы данных.

Лабораторная работа № 2

Создание баз данных посредством SQL

Цель работы – изучить возможности создания баз данных посредством инструкций SQL, правила задания ограничений различных типов, а также запросов на добавление, обновление, удаление информации из базы данных.

- ***Теоретические сведения***

Условия целостности данных

Для сохранения непротиворечивости и правильности хранимой информации СУБД поддерживает так называемые условия целостности данных.

Как правило, в реляционных БД используются следующие условия целостности:

– *обязательное наличие данных*. Некоторые столбцы базы данных должны обязательно содержать некоторые определенные значения, т.е. для них не допускаются значения NULL. В качестве примера из рассмотренной выше учебной базы данных такими столбцами являются, очевидно, столбцы fname, lname и некоторые другие. Стандарт ANSI/ISO и большинство коммерческих СУБД (ORACLE в частности) поддерживают выполнение подобного условия посредством ограничения *NOT NULL*;

– *условие на значение*. Во многих коммерческих СУБД при создании базы данных для некоторых столбцов таблиц можно назначить условия на принимаемые ими значения. Данные условия задаются в инструкциях по созданию таблиц посредством так называемых *check conditions*. В качестве примера можно рассмотреть фрагмент инструкции *create table* для создания таблиц:

```
create table staff (sno integer not null,
                    age integer,
                    check (sno between 101 and 199),
                    check (age >=21));
```

– *целостность таблицы (сущности)*. Первичный ключ таблицы должен в каждой строке иметь уникальное значение и не допускать значений NULL. Создание первичных ключей таблиц в БД поддерживается СУБД посредством ограничения *primary key*;

– *деловые правила*. Обновление информации в базе данных может быть ограничено так называемыми деловыми правилами. К примеру, в системе можно запретить принимать заказы на определенный вид продукции, в случае если на складе не имеется её достаточного количества. Реализация данных правил в СУБД может реализовываться посредством автоматически запускаемых процедур, запуск которых осуществляется после выполнения

некоторых из операций по изменению содержимого записей базы. Такие процедуры сохраняются в откомпилированном виде в базе и являются наряду с таблицами также объектами базы. Эти процедуры называются *триггерами* и при создании «прикрепляются» к определенным таблицам, а также настраиваются на определенные операции изменения записей этих таблиц;

— *непротиворечивость*. Многие реальные операции вызывают в базе данных несколько изменений одновременно. Например, операция увольнения сотрудника должна реально сопровождаться переводом всех выполняемых им работ на какого-то другого сотрудника либо путем временного задания значений NULL или значений по умолчанию для некоторых предназначенных для этого столбцов. Зачастую последовательное выполнение инструкций, в совокупности решающих общую задачу по изменению информации в базе данных с соблюдением конечного непротиворечивого состояния базы данных, решается СУБД посредством поддержки механизма *транзакций*. *Транзакция* считается успешно завершенной, если каждая из операций, входящих в неё, выполнена успешно. Подтверждением успешного завершения является выполнение инструкции ***commit***. В случае неуспешного завершения хотя бы одной из операций транзакция должна отменить все произведённые ранее изменения в базе и перевести её в начальное непротиворечивое состояние посредством инструкции ***rollback***. Выполнение инструкции ***rollback*** вызывает «откат» текущей транзакции;

— *ссылочная целостность*. В рассматриваемом контексте нарушение целостности может произойти при выполнении операций удаления или обновления над связанными строками различных таблиц базы данных. Стандартом ANSI/ISO регламентированы *правила удаления и обновления* связанных строк таблиц. Таких правил четыре:

— ***restrict***. Запрещает удалять или обновлять строки-предки, в случае если на них ссылаются строки-потомки из других таблиц базы данных. В некоторых СУБД данное правило носит название ***no action***. Следует также иметь в виду, что данное правило зачастую воспринимается СУБД по умолчанию;

— ***cascade***. Согласно ему СУБД производит автоматическое (каскадное) удаление или обновление значений в некоторых столбцах строк-потомков при удалении или изменении соответствующих им столбцов в строках-предках. Данное правило опасно в употреблении, в случае если в базе существуют разнообразные множественные связи. Правило реализуется посредством ограничения ***foreign key*** посредством фраз ***on delete*** или ***on update cascade***;

— ***set null*** и ***set default***. Регламентируются стандартом ANSI/ISO, однако не поддерживаются СУБД ORACLE. В соответствии с этими правилами при удалении или изменении первичного ключа строки-предка во внешнем ключе строки-потомка устанавливаются значения ***null*** или значения по умолчанию.

Создание таблиц

Инструкция *create table* позволяет создавать таблицы БД и ограничения на значения столбцов, а также создавать связи типа первичный ключ - внешний ключ между таблицами. Синтаксическая диаграмма данной инструкции представлена на рис. 1.

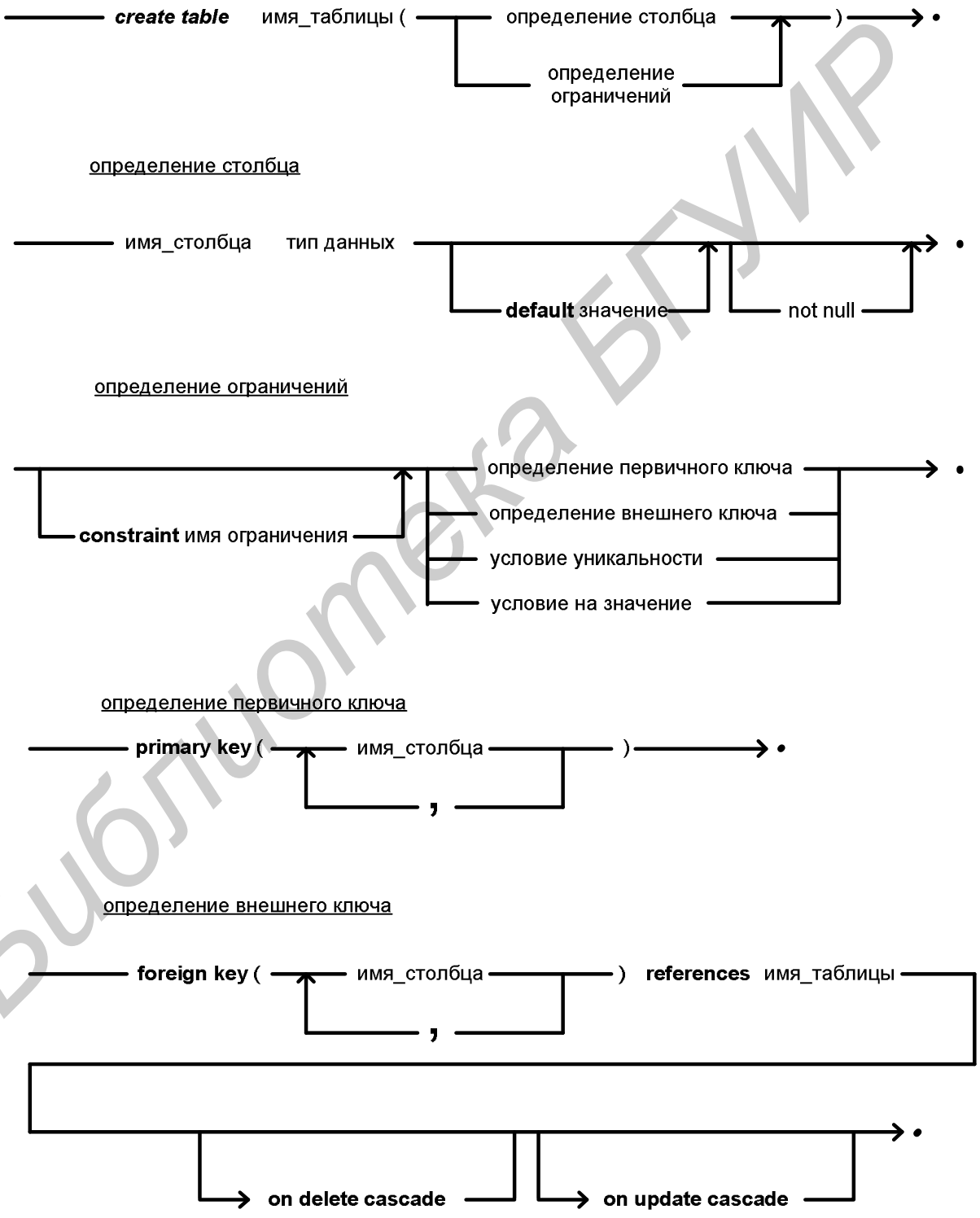

Рис. 1. Синтаксическая диаграмма *create table* (окончание см. на с.17)

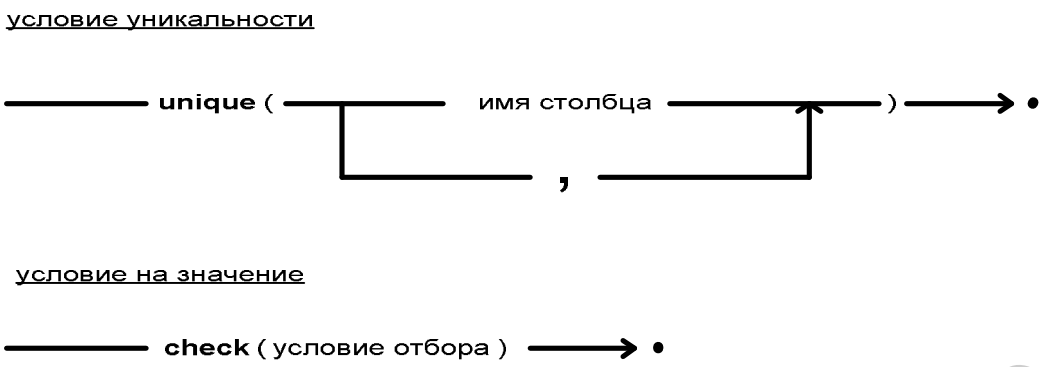

Рис. 1. Окончание (начало см. на с.16)

Создание таблицы ORDERS посредством *create table* можно продемонстрировать следующим примером. На рис. 2 приведены таблицы и предполагаемые связи, создаваемые между ними.

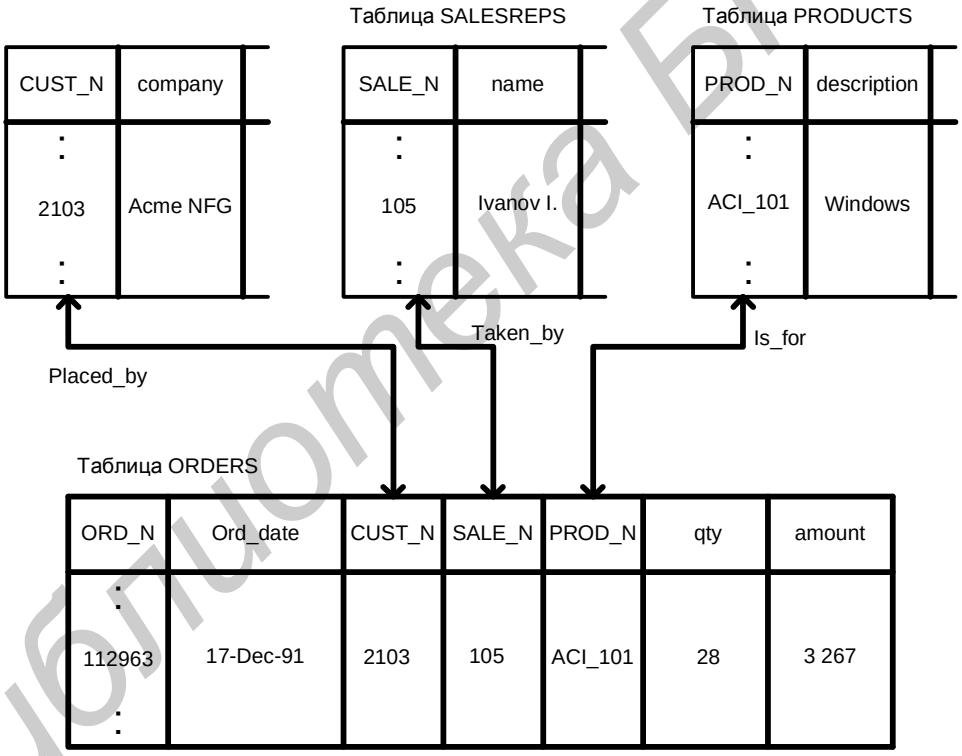

Рис. 2. Пример именованных таблиц и именованных связей между ними

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 *integer*,

primary key (ord_n),
unique (ord_n),
constraint placed_by *foreign key* (cust_n) *references* customers *on delete cascade*,
constraint taken_by *foreign key* (sale_n) *references* salesreps,
constraint is_for *foreign key* (prod_n) *references* products);

Варианты заданий

1. Создать базу данных малого коммерческого предприятия (таблицы сотрудников, заказчиков, продукции, заказов) со всеми необходимыми ограничениями и именованными связями между таблицами.

2. Создать базу данных произвольного сборочного производства. Предварительно зарисовать иерархическую структуру используемых деталей. Уровни иерархии выдерживать по этапам сборки. Перенести иерархию на таблицы реляционной базы данных с организацией связей между таблицами, отражающими реальные связи между этапами сборки.

3. Создать базу данных для системы визуального проектирования радиоэлектронных устройств. Предусмотреть таблицы различных радиоэлектронных модулей с соответствующими атрибутами типа диапазонов входных токов (напряжений, сопротивлений) и др. В результирующей таблице готовых устройств отмечать ссылки на используемые модули.

- ***Контрольные вопросы***

1. Какие типы ограничений используются при создании таблиц?
2. Какие правила можно использовать для обеспечения ссылочной целостности при создании связанных таблиц БД?
3. Объясните порядок задания определения столбцов и ограничений при создании таблиц в инструкции *create table.*

Лабораторная работа № 3

Процедуры и функции PL/SQL

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

- ***Теоретические сведения***

Структура программ PL/SQL

Собственно SQL – это лишь язык доступа к данным, дающий возможность приложениям помещать данные в базы данных и извлекать их оттуда. Другими словами, SQL не является полнофункциональным языком программирования, которым можно пользоваться для разработки эффективных приложений баз данных. Для создания приложений необходимо применять процедурные языки, которые будут охватывать SQL и таким образом взаимодействовать с базами данных. Таким языком является «собственный» процедурный язык ORACLE под названием PL/SQL. При его изучении необходимо вначале ознакомиться со структурной организацией программ PL/SQL.

Базовой единицей PL/SQL является блок. Блоки имеют следующую структуру:

```
declare
/*раздел объявлений (переменные, типы, курсоры и др.)*/

begin
/*Основной раздел блока, называемый выполняемым. Содержит процедурные и SQL - операторы*/

exception
/*Раздел обработки исключительных ситуаций. Содержит операторы обработки ошибок*/

end;
```

Обязателен только выполняемый раздел, кроме того, он должен содержать по крайней мере один выполняемый оператор.

Язык программирования PL/SQL разработан на базе языка третьего поколения Ada. Одним из общих свойств этих языков и является их блочная структура.

Объявления

В блоке программ PL/SQL можно объявлять конструкции различных типов. Рассмотрим, как объявляются переменные и константы; подтипы, определяемые пользователями; курсоры.

Переменные и именованные константы могут иметь любой тип данных ORACLE или ANSI/ISO. В следующем примере объявляются переменная и именованная константа при помощи ANSI – типа *integer*:

declare
id *integer*;
standard *constant integer* :=500;

Когда в программе объявляется переменная, ей может быть присвоено начальное значение либо значение по умолчанию:

declare
id *integer*:=0;
standard *integer default* 500;

Подтипы, определяемые пользователями. Пример объявления пользовательских подтипов и соответствующих переменных удобно пояснить на примере:

declare
varchar2_50 *varchar2*(50);
subtype description *is* varchar2_50;
current_description description *default* ‘unknown’

Тип *varchar2*() – тип данных ORACLE, предназначенный для задания строковых переменных переменной длины.

Атрибуты. В программах PL/SQL можно использовать атрибуты *%type* и *%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 баз данных.

К примеру, с помощью атрибута *%type* можно объявлять тип данных, принадлежащий другой программной конструкции или столбцу таблицы базы данных.

declare
id parts.id *%type*;
unit_price parts.price *%type*;

В приведенном выше фрагменте предполагается, что parts является таблицей базы данных, a id и price её атрибутами.

С помощью атрибута *%rowtype* можно объявлять переменные, имеющие тип записи, и другие конструкции:

```
declare
type parts_table is table of parts %rowtype;
current_part parts_table;
```

Здесь объявляется «агрегированный» тип parts_table и переменная current_part этого типа.

Курсоры. В литературе [2] курсоры иногда называют *рабочей областью SQL-оператора*. Однако точнее его можно определить как указатель на текущую строку результирующего множества оператора. Операторы *select*, возвращающие одну строку, обрабатываются СУБД автоматически, т. е. она сама создаёт курсор и считывает данные посредством него из результирующего вектора. Для обработки строк запроса, возвращающего несколько строк, приложение должно объявлять курсор явно, указав его имя, а затем ссылаться на него при обработке строк по очереди. При обработке строк следует иметь в виду, что курсор устанавливается перед первой строкой результирующего множества, сформированного СУБД по выполнении запроса и сохраненного на сервере БД. Следующий пример демонстрирует процедуру объявления курсоров в блоке объявлений программ PL/SQL:

```
declare
cursor part_cur is select * from parts;
cursor cust_cur (state_id char) is
select id, l_name, f_name, phone
from customers
where state=state_id;
```

Здесь part_cur – это простой курсор, соответствующий всем строкам и столбцам таблицы parts, cust_cur – пример параметризованного курсора с параметром state_id.

Функциональные возможности программ.

Как видно из предыдущих примеров, оператор присваивания в PL/SQL обозначается символом ‘:=’.

Управление выполнением программ. Как и во всех языках программирования, в PL/SQL существуют операторы управления

вычислительным процессом, такие, как операторы условного управления, итерационного и последовательного управления. Условное управление осуществляется посредством оператора *if-elsif-else*:

if условие *then*
оператор 1;
оператор 2;
…
elsif (не elseif)
оператор 3;
оператор 4;
…
else
оператор 5;
оператор 6;
…
end if;

Для итерационного управления ходом вычислений можно применять три вида циклов:
цикл с постусловием *exit when*

loop
оператор 1;
оператор 2;
exit when условие;
end loop;

цикл с предусловием *while*

while условие *loop*
оператор 1;
оператор 2;
end loop;

цикл с предусловием *for*

<<внешний цикл>> ─ метка или имя цикла
for x *in* y..z
loop
внешний оператор 1;
<<внутренний цикл>>
loop
внутренний оператор 1;
внутренний оператор 2;

exit внешний цикл *when* условие1;
exit внутренний цикл *when* условие2;
end loop внутренний цикл;
внешний оператор 2;
…
end loop;

Взаимодействие с базами данных. Основное назначение языка PL/SQL ─ создание программ для работы с базами данных. Программа может взаимодействовать с базами данных только посредством SQL. Программы PL/SQL могут управлять информацией, содержащейся в базе данных, используя SQL-операторы DML, курсоры и динамический SQL. О динамическом SQL речь будет вестись в теоретическом описании к следующей лабораторной. Здесь же остановимся на стандартном DML и работе с курсорами.

Стандартный DML. Для внесения изменений в строки таблицы базы данных программы PL/SQL могут включать любые корректные операторы *insert*, *update* или *delete*. Для задания переменной некоторого значения или набора значений можно использовать команду *select into*. Например:

```
declare
current_part parts %rowtype;
begin
select * into current_part
from parts
where id=6;
```

Работа с курсорами. Для работы со строками, извлекаемыми из результирующего множества «многострочного» запроса посредством курсора, необходимо выполнить три операции: открыть курсор посредством команды *open*, считать информацию в ранее объявленные переменные посредством *fetch* и закрыть курсор посредством *close*. Например:

```
declare
cursor parts_cur is
select * from parts;
current_part parts %rowtype;
begin
open parts_cur;
loop
fetch parts_cur into current_part;
…другие операторы…
end loop;
close part_cur;
…
```

Для упрощения необходимых при установке и обработке курсоров можно пользоваться *курсорными циклами for*:

declare
cursor parts_cur *is*
select * from parts;
begin
for current_part *in* parts_cur
loop
…другие операторы…
end loop;

В курсорном цикле *for* автоматически объявляется переменная или запись, с помощью которой можно считывать записи, открывать курсор, выбирать из него строки и закрывать курсор, когда из него выбирается последняя строка.

В программах PL/SQL можно использовать несколько уникальных курсорных атрибутов ― *%isopen*, *%found*, *%notfound*, *%rowcount* ― для принятия решения во время обработки курсоров.

%found– при употреблении с курсором в условии возвращает TRUE, если курсор был предварительно открыт;

%isopen – возвращает число строк в курсоре после его открытия;

%notfound – возвращает FALSE, после того как из курсора была вычитана последняя строка;

%rowcount– принимает значение TRUE, после того как из курсора вычитана последняя строка.

Пример использования можно продемонстрировать следующим фрагментом «псевдокода»:

…
while parts_cur *%found*
loop
fetch parts_cur *into* current_part;
…другие операторы…
end loop;

Обработка исключительных ситуаций

В программах PL/SQL подпрограммы обработки ошибок представляют собой обработчики исключительных ситуаций. Распознав ошибку, программа PL/SQL устанавливает (*raises*) исключительную ситуацию и передает управление соответствующей подпрограмме – обработчику исключительной ситуации, которая не является частью тела программы.

Исключительная ситуация представляет собой поименованное условие возникновения ошибки. В PL/SQL имеется множество предопределенных

исключительных ситуаций, соответствующих наиболее часто встречающимся в ORACLE ошибкам. Например:

– программа распознает исключительную ситуацию NO_DATA_FOUND в случае, если в результирующем множестве оператора *select into* нет строк, и исключительную ситуацию TOO_MANY_ROWS, когда в результирующем множестве этого оператора более одной строки;

– программа распознает исключительную ситуацию DUP_VAL_ON_INDEX (повторяющееся значение в индексе) в случае, если оператор *insert* или *update* дублирует ключевое значение, уже находящееся в таблице.

В PL/SQL включено около 20 исключительных ситуаций. Когда в программе встречается предопределенная исключительная ситуация, управление программой передаётся соответствующему обработчику исключительных ситуаций, если таковой имеется.

Исключительные ситуации, определяемые пользователями, можно объявлять в разделе объявлений программы. Однако для того, чтобы затем установить такую исключительную ситуацию, необходимо выполнить её явную проверку. При желании можно назначить индивидуальный номер ошибки ORACLE исключительной ситуации, определяемой пользователем. При вызове программой указанной ошибки ORACLE автоматически устанавливается исключительная ситуация, определяемая пользователем.

Нижеследующий пример демонстрирует использование заранее предопределенных и определяемых пользователем исключительных ситуаций, а также соответствующих обработчиков:

```
declare
invalid_part exception;
insufficient_privileges exception;
err_num integer;
err_msg varchar2(2000);
part_num integer;
begin
select … into … from…;
update parts
set unit_price=20.00
where id=6;
if SQL %NOTFOUND then
raise invalid_part;
end if;
exception
when no_data_found then
raise_application_error (-20001, ‘No rows found’);
when too_many_rows then
raise_application_error (-20002, ‘Too many rows found’);
when invalid_part then
```

raise_application_error (-20003, ’Invalid part ID’);
when insufficient_privileges *then*
raise_application_error(-20004,’Insufficient privileges to update table’);
when others then
err_num:=SQLCODE;
err_msg:=SUBSTR(SQLERRM, 1, 100);
raise_application_error(-20000, err_num || ‘ ‘||err_msg);
…

В данном примере:

– для возвращения в вызывающую среду номера и сообщения об ошибке, определяемой пользователем, применяется процедура *raise_application_error* (установить ошибку приложения). Номера сообщений об ошибках, определяемых пользователем, должны лежать в диапазоне от –20000 до –20999;

– для создания общего обработчика всех исключительных ситуаций, для которых не определены собственные обработчики, применяется синтаксис *when others*;

– для возвращение номера и сообщения о самой последней ошибке ORACLE применяются специальные функции SQLCODE и SQLERRM.

Типы программ PL/SQL. Хранимые процедуры и функции

В ORACLE различают три типа программ PL/SQL: *анонимные блоки*, *хранимые процедуры и функции*. Анонимные блоки – неименованные блоки PL/SQL, не хранящиеся в базе данных. В процессе работы приложение посылает такой блок серверу, и после его обработки блок прекращает свое существование.

Хранимые процедуры и функции в отличие от анонимных блоков сохраняются в базе данных и наряду с таблицами, представлениями и т.д. являются самостоятельными объектами баз данных ORACLE. Процедуры и функции сохраняются в базе данных 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, разумеется, удаляется раньше, и при очередном вызове данный код снова загружается в пул с диска. Такая организация способствует производительности выполнения вызываемых процедур и функций, поскольку исключает постоянную загрузку с диска исполняемого кода. Хранимая процедура отличается от функции тем, что функция в отличие от процедуры возвращает значения в вызывающую среду.

Создание процедур. Процедуры создаются посредством инструкции *create procedure*:

*c**reate*[*or replace*] *procedure* имя_процедуры [(аргумент1 [{*in* | *out* | *in out*}] тип, … аргумент2 [{*in* | *out* | *in out*}] тип)] {*is*|*as*} тело процедуры/

Чтобы изменить текст процедуры, её необходимо удалить и повторно создать её. Во время разработки процедур эта операция повторяется достаточно часто, поэтому ключевые слова *or replace* позволяют выполнить такую операцию за один раз. Если процедура существует, то она удаляется без всякого предупреждения (в данном случае вызов инструкции *drop procedure* не требуется), если же не существовала, то она просто создаётся.

Параметры *in*, *out* и *in out* используются как при создании процедур, так и функций. Смысл параметра *in* в следующем: значение физического параметра передаётся в функцию. Внутри процедуры формальный параметр рассматривается в качестве константы PL/SQL (параметр только для чтения) и не может быть изменен. Когда процедура завершается и управление программой передаётся в вызывающую среду, фактический параметр не изменяется. При использовании параметра *out* любое значение, которое имеет фактический параметр при вызове процедуры, игнорируется. Внутри процедуры формальный параметр рассматривается в качестве неинициализированной переменной, т.е содержит *null*-значение, и можно как записать в него значение, так и считать значение из него. Когда управление передаётся в вызывающую среду, содержание формальной переменной присваивается фактическому параметру. Параметр *in out* ─ это комбинация параметров *in* и *out.* В данном случае формальный параметр рассматривается в качестве инициализированной переменной.

Создание функций. Создание функций отличается от создания процедур названием инструкции и наличием оператора *return*.

*c**reate*[*or replace*] *function* имя_функции [(аргумент1 [{*in* | *out* | *in out*}] тип, … аргумент2 [{*in* | *out* | *in out*}] тип)] *return* возвращаемый тип{*is*|*as*} тело функции оператор *return*/

Оператор *return* имеет общий синтаксис: *return* выражение. Выражение ─ это возвращаемое значение. Значение выражения преобразуется в тип, указанный в команде *return* при описании функции, если, конечно, это значение уже не имеет данный тип.

Ниже приведен пример создания простой хранимой функции:

create or replace function get_customer_address (last *in varchar2*, firsr *in varchar2*)
return varchar2 is
addr *varchar2*(20);

```
begin
select address into addr
from customers
where lname=last and fname=first;
return addr;
exception
when others then
return NULL
end get_customer_address;
```

Вызов процедур и функций. Приложение может вызывать процедуру в анонимном блоке PL/SQL. Вызов функций может осуществляться либо в анонимном блоке посредством оператора присваивания, либо в условии *when* SQL-оператора.

Приведём примеры:

```
declare
cur_cust_last varchar2(100);
cur_cust_first varchar2(100);
cur_cust_addr varchar2(100);
begin
…
cur_cust_addr:= get_customer_address(cur_cust_last, cur_cust_first);
…
end/
```

Вызов функции в условии SQL-оператора:

```
delete from orders
where address= get_customer_address (‘Иванов’,’Иван’);
```

- ***Варианты заданий***

1. Написать процедуру, повышающую заработную плату сотрудников, обслуживающих максимальное количество объектов аренды.

Создать функцию, подсчитывающую количество значений неопределенно (*null*) в столбце зарплат сотрудников, возвратить подсчитанное количество.

2. Написать процедуру изменения домашнего адреса клиента по указанному в качестве параметра имени.

Создать функцию, возвращающую адрес отделения с наименьшим количеством сотрудников.

3. Создать процедуру, переносящую информацию о сотрудниках, родившихся не позднее 1 января 1980 года, во вспомогательную таблицу.

Создать функцию, определяющую, существуют ли отделения, где количество сотрудников превышает определённое количество процентов от общего количества сотрудников предприятия. Возвратить *TRUE* или *FALSE* , а также осуществить корректный её вызов из анонимного блока PL/SQL.

4. Создать процедуру, выполняющую подсчет количества клиентов, обратившихся в каждый из офисов. Для вывода информации воспользоваться модулем DBMS_OUTPUT для отладки программ. (Описание модулей приводится в теоретических сведениях к лабораторной работе 5.)

Создать функцию, которая возвращала бы среднее количество сотрудников, работающих в каждом из отделений фирмы.

5. Написать функцию, возвращающую процент обслуживаемых объектов по каждому заданному офису. Вызвать функцию из цикла анонимного блока, результаты вывести на экран, используя модуль DBMS_OUTPUT.

Создать процедуру, обеспечивающую удаление арендатора из таблицы по указанным имени и фамилии.

6. Создать процедуру, обеспечивающую вывод на экран информации обо всех сотрудниках, работающих в городе, заданном в качестве аргумента.

Написать функцию, которая определяет, работает ли тот или иной сотрудник на предприятии (путем возвращения TRUE или FALSE), а в параметре *out* по выходе из функции записывает его телефонный номер.

7. Создать процедуру, копирующую строки с результатами осмотров за текущую неделю, во вспомогательную таблицу.

Создать функцию, возвращающую идентификатор отделения, в котором работает максимальное количество сотрудников.

8. Создать процедуру, которая в параметре *in out* возвращала адрес объекта недвижимости. Фамилия владельца указывается в этом же параметре. Осуществить вызов с целью вывода переменной на экран посредством использования процедур модуля DBMS_OUTPUT.

Создать функцию, подсчитывающую количество осмотров, сделанных потенциальными арендаторами объектов аренды за текущий день. В вызывающую среду возвращать количество осмотров, в параметре *out* — сделанные комментарии.

9. Создать функцию, определяющую, действительно ли владелец (параметр 1) является владельцем объекта (параметр 2). Возвращать TRUE или FALSE. В случае FALSE в параметр 1 записывать фамилию истинного владельца данного объекта.

Создать процедуру, увеличивающую зарплату сотрудников, которые обслуживают наибольшее количество объектов, на процент, указанный в её параметре.

10. Создать процедуру, «переводящую» сотрудников заданного отделения в другие отделения этого же города. В каждый из офисов переводить по приблизительно одинаковому, в смысле среднего значения, количеству сотрудников.

Создать функцию, подсчитывающую количество сотрудников, работающих в заданном городе.

- ***Контрольные вопросы***

1. Назовите, какие типы программ используются в PL/SQL. Чем они отличаются друг от друга?
2. Как обрабатываются многострочные запросы в программах PL/SQL?
3. Расскажите о структурных особенностях программ PL/SQL.
4. Как производится обработка ошибочных ситуаций в программах PL/SQL?

Лабораторная работа № 4

Триггеры баз данных

Цель работы – изучить предназначение и особенности создания триггеров баз данных по сравнению с хранимыми процедурами и функциями.

Теоретические сведения

Триггер базы данных ─ это хранимая процедура, которую можно связать с некоторой таблицей. Когда приложение выполняет SQL-оператор DML над таблицей, удовлетворяющей условиям, указанным в триггере, ORACLE автоматически активизирует или выполняет триггер. Таким образом, триггеры можно использовать для настройки реакции ORACLE на различные события приложения. Вне зависимости от типа триггера все триггеры создаются одинаково. Общий синтаксис создания таков:

create [*or replace*] *trigger* имя_триггера
{*before* | *after* | *instead of*} активизирующее_событие
on имя_таблицы
[*for each row*]
тело_триггера

Здесь активизирующее_событие указывает событие активации триггера (далее указывается конкретная таблица или представление).

Например, следующий триггер автоматически регистрирует изменения, вносимые в таблицу parts:

```
create or replace trigger parts_log
after insert or update or delete on parts
declare
    stmt_type char(1);
begin
    if inserting then
        stmt_type:='I';
    elsif updating then
        stmt_type:='U';
    else
        stmt_type:='D';
    end if;
insert into part_change_log
values(stmt_type, USER);
end parts_log;
```

Как видно из вышеприведенного примера, описание триггера содержит однозначно определяемые части:

– список операторов, активизирующих триггер, который включает insert, update и/или delete;

– способ активизации триггера до (*before*) или после (*after*) выполнения оператора триггера в зависимости от логики конкретного приложения.

Помимо этого в описании триггера указывается, должен ли триггер активизироваться один раз (независимо от того, на какое количество строк оказывает воздействие оператор триггера – *операторный триггер*)), или для каждой изменяемой строки (такие триггеры называются *строчными*). Следует также отметить, что триггер может быть связан с одной и только с одной таблицей.

Приведенный выше триггер – это операторный триггер. В следующем примере триггер parts_log является строчным:

```
create or replace trigger parts_log
before insert or update or delete on parts
for each row
declare
    stmt_type char(1);
begin
    if inserting then
        stmt_type:='I';
    elsif updating then
        stmt_type:='U';
    else
        stmt_type:='D';
    end if;
insert into part_change_log
values (:new.id, :old.id, :new.unit_price, :old. unit_price, :new.description, :old. description, stmt_type, USER, TO_CHAR (SYSDATE, 'DD-MON-YYYY'));
end parts_log;
```

Из рассмотренных выше примеров видно, что в PL/SQL имеются следующие уникальные языковые конструкции для триггеров баз данных:

– предикаты *inserting*, *updating*, *deleting* могут использоваться в операторах условного перехода в теле триггера;

– значения корреляции позволяют с помощью строковых триггеров обращаться к новым и старым значениям полей текущей строки. Когда оператором триггера является *insert*, все старые значения полей являются *null*-значениями. Аналогично, когда оператором триггера является *delete*, все новые значения являются соответственно *null*-значениями.

В предыдущих версиях ORACLE обновляемыми были лишь те представления, которые соответствовали основному правилу обновления

представлений. В 8-й версии ORACLE любое представление, для которого создан триггер типа *instead of,* может считаться обновленным. Детальное описание триггеров *instead of* приведено в [3].

- ***Варианты заданий***

1. Создать триггер, настроенный на операцию добавления записей в таблицу сотрудников, с фиксацией идентификатора строки и даты выполненного обновления во вспомогательной таблице базы данных.

2. Создать строчный триггер, активизирующийся каждый раз для удаляемых строк таблицы с фиксацией пользователя, выполнившего данные удаления, а также старых данных об имени, фамилии, дне рождения, адресах и телефонах удаленных строк с информацией о сотрудниках предприятия.

3. Создать триггер, который бы фиксировал во вспомогательной таблице изменения, произведенные над столбцом рентной стоимости для таблицы с описанием объектов недвижимости с сохранением значений до и после изменения, а также имени пользователя, от лица которого данные изменения производились.

4. Создать триггер, настроенный на ввод новых строк в таблицу с описанием сотрудников. Реакцией на добавление должна быть проверка количества работающих сотрудников в заданном отделении и занесение (путем разового обновления данных соответствующего столбца) этого количества с учетом нового сотрудника во вспомогательную таблицу. В случае превышения вычисленного значения 5-ти в строку с данным отделением во вспомогательной таблице следует добавить некоторое информирующее сообщение.

5. Создать триггер, фиксирующий ежеквартальное увеличение заработной платы для сотрудников предприятия. Данные о предварительных изменениях фиксировать во вспомогательной таблице с указанием даты сделанных изменений. Перед обновлением триггер считывает информацию из вспомогательной таблицы, сравнивая текущую дату с датой последней прибавки; в случае если со времени последнего пересчета прошло более 3-х месяцев, во вспомогательной таблице фиксируются дата текущего пересчета и значения средних зарплат сотрудников до и после изменения в зависимости от занимаемых должностей.

- ***Контрольные вопросы***

1. Что такое триггер?
2. Чем отличаются триггеры от хранимых процедур и функций?
3. Какие типы триггеров вы знаете?

Лабораторная работа № 5

Динамический SQL

Цель работы – ознакомиться с таким объектом баз данных, как модули, а также с системными модулями DBMS_OUTPUT и DBMS_SQL, предназначенными для отладки приложений PL/SQL и написания процедур динамической обработки SQL-операторов.

Теоретические сведения

Перед тем как рассмотреть использование модуля DBMS_SQL для организации «динамических» процедур, следует рассмотреть такие объекты баз данных, как *модули*.

Модули

Модуль (*package*) – это группа процедур, функций и других конструкций, хранимых вместе в базе данных как одна единица. Модули особенно полезны для компоновки нескольких процедур и функций, имеющих отношение к конкретному приложению баз данных.

Модуль состоит из двух частей: описания и тела.

– Описание модуля определяет интерфейс связи с этим модулем. В описании модуля объявляются все переменные и именованные константы, курсоры, процедуры, функции и другие конструкции модуля, которые необходимо сделать доступными для программ, внешних по отношению к этому модулю. Другими словами, всё объявленное в описании модуля является общим.

– В теле модуля (*package body*) определяются все общие процедуры и функции, объявленные в описании модуля. Кроме того, в тело модуля могут включаться определения других конструкций, не указанных в его описании. Такие конструкции модуля являются частными, т.е. доступными только для программ внутри модуля.

Ниже приведен пример определения описания модуля и его тела:

create or replace package part_mgmt *is*
- глобальная переменная
- current_part parts *%rowtype*;
- процедуры и функции
- insert_part вводит новый элемент ассортимента в таблицу parts
- update_part_unitprice обновляет цену элемента ассортимента
- delete_part удаляет элемент ассортимента

---

procedure insert_part (part_record parts *%rowtype*);

procedure update_part_unitprice (part_id *in integer*, new_price *in number*);
procedure delete_part (part_id *in integer*);
end part_mgmt;

create or replace package body part_mgmt *is*
procedure insert_part (part_record parts *%rowtype*) *is*
dup_primary_key *exception*;
begin
insert into parts
values (part_record.id, part_record.unitprice, part_record.description);
exception
when dup_ primary_key *then*
raise_application_error (-20001,'Дубликат ID');
when others then
raise_application_error (-20002,'Неопределенная ошибка');
end insert_part;

_____

…определение других процедур и функций модуля…
end part_mgmt;

Обращение к процедурам и функциям модуля происходит посредством уточняющей записи через точку.

Модуль DBMS_OUTPUT

С помощью процедур модуля DBMS_OUTPUT реализованы две базовые операции: GET и PUT. Операция PUT берет свои аргументы и помещает во внутренний буфер для хранения. Операция GET считывает этот буфер и возвращает его содержимое процедуре в качестве аргумента. Размер буфера устанавливается с помощью процедуры ENABLE.

Выполнение операции PUT обеспечивается процедурами PUT, PUT_LINE и NEW_LINE, а выполнение операции GET – процедурами GET_LINE и GET_LINES. Управляют буфером процедуры ENABLE и DISABLE.

Процедуры PUT и PUT_LINE вызываются следующим образом:

procedure PUT (a *varchar2*);
procedure PUT (a *number*);
procedure PUT (a *date*);

procedure PUT_LINE (a *varchar2*);
procedure PUT_LINE (a *number*);
procedure PUT_LINE (a *date*);

Данные процедуры переопределяются типом параметра.

Буфер организован в виде строк, каждая из которых может состоять не более чем из 255 байт. PUT_LINE добавляет к аргументу символ новой строки, сообщая о конце строки; PUT же этого не делает. Вызов PUT_LINE аналогичен вызову PUT с последующим вызовом NEW_LINE.

Процедура GET_LINE вызывается следующим образом:

procedure GET_LINE (line *out varchar2*, status *out integer*);

Здесь line представляет собой последовательность символов, из которых состоит одна строка буфера, а status указывает на то, успешно или нет была считана эта строка. Максимальная длина строки ─ 255 байт. Если строка считана, то в переменной status находится 0, если в буфере больше нет строк для считывания , то в status ─ 1.

Аргументом процедуры GET_LINES является индексная таблица. Тип таблицы и вызов данной процедуры выглядят следующим образом:

type CHARARR *is table of varchar2*(255) *index by binary integer*;
procedure GET_LINES (lines *out* chararr, numlines *in out* integer);

Здесь numlines ─ число запрошенных строк, на входе в GET_LINES указывается их число, на выходе ─ число фактически возвращаемых строк.

Тип CHARARR определен в модуле DBMS_OUTPUT, поэтому если GET_LINES вызывается явным образом, нужно объявлять переменную с типом DBMS_OUTPUT.CHARARR. Например:

```
declare
        v_Data DBMS_OUTPUT.CHARARR;
        v_NumLines number;
begin
        DBMS_OUTPUT.ENABLE (1000000);
        DBMS_OUTPUT.PUT_LINE (‘Line one’);
        DBMS_OUTPUT.PUT_LINE (‘Line two’);
        DBMS_OUTPUT.PUT_LINE (‘Line three’);
        v_NumLines:=3;
        DBMS_OUTPUT.GET_LINES( v_Data, v_NumLines);
        for v_Counter in 1..3 v_NumLines loop
                insert into temp_table (char_col)
                values (v_Data(v_Counter));
        end loop;
end;
```

Процедура ENABLE задает размер буфера в байтах, по умолчанию задается размер 20 000 байт, а максимальный размер – 1 000 000 байт. Если объявлена процедура DISABLE, то содержимое буфера уничтожается и последующие вызовы PUT и PUT_LINE бесполезны.

По существу модуль DBMS_OUTPUT реализует алгоритм «первым пришел — первым обслужен». В утилите SQL*Plus имеется средство, называемое SERVEROUTPUT (серверный вывод), команда SQL*Plus, называемая SET SERVEROUTPUT ON, неявно вызывает процедуру DBMS_OUTPUT.ENABLE, которая устанавливает внутренний буфер серверного вывода. Если нужно, можно указать размер буфера с помощью команды

SET SERVEROUTPUT ON SIZE *размер_буфера*

Здесь *размер_буфера* — первоначальный размер буфера (аргумент процедуры DBMS_OUTPUT.ENABLE, вызываемой по умолчанию). Процедура DBMS_OUTPUT.GET_LINES вызывается после окончания блока PL/SQL. Это означает, что результаты выводятся на экран после завершения блока, а не во время его выполнения. На рис. 3 продемонстрированы результаты использования средств серверного вывода в утилите SQL*Plus.

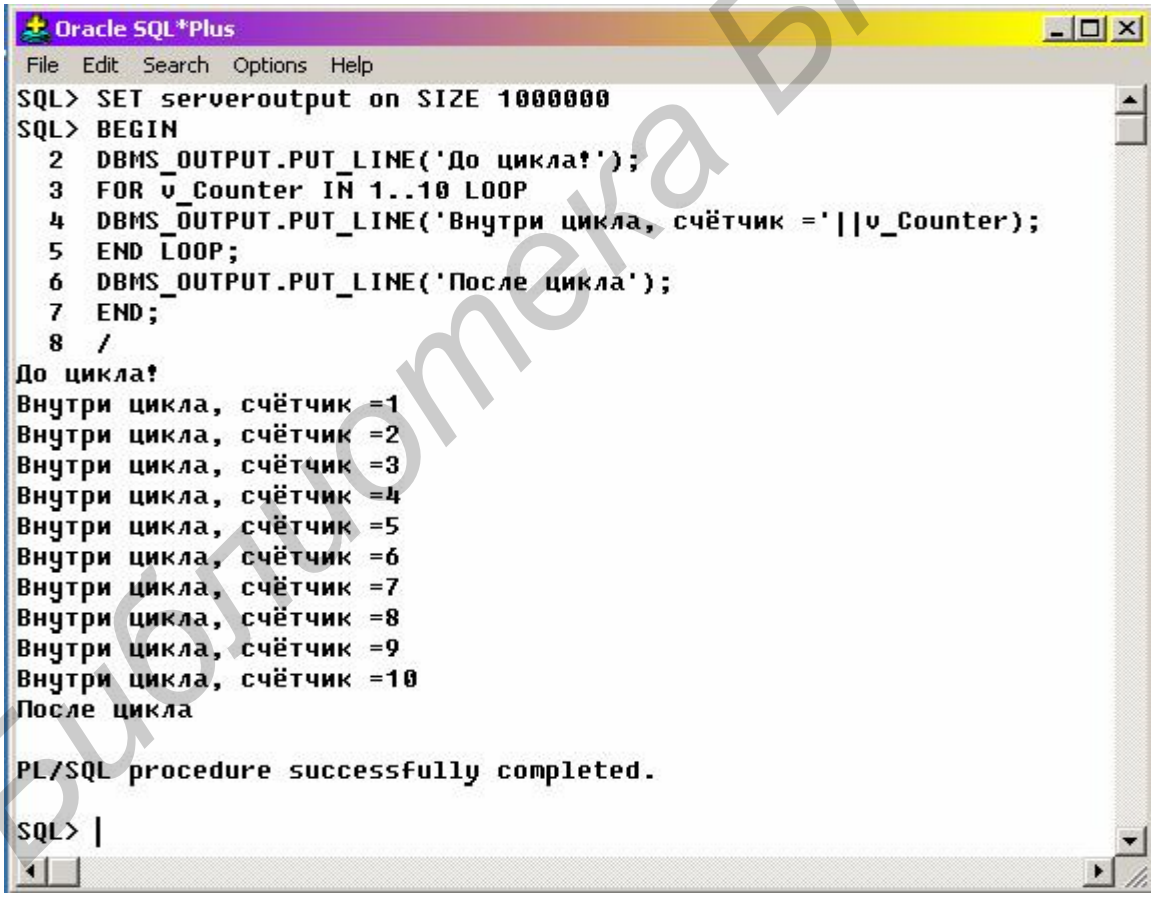

Рис. 3. Использование SERVEROUTPUT и PUT_LINE

Обзор динамического SQL

Все программы, приведенные выше, 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.

Выполнять SQL-конструкции можно двумя способами:

– с помощью модуля DBMS_SQL;

– посредством внутреннего динамического SQL (для ORACLE 8*i*).

В данном пособии рассматриваются вопросы написания приложений, использующих только модуль DBMS_SQL.

Модуль 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 извлечения (*select*) – описание выходных переменных с помощью DBMS_SQL.DEFINE_COLUMN.
6. Выполнения 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*);

Здесь c ─ идентификационный номер курсора, который предварительно должен быть открыт посредством OPEN_CURSOR;

─ statement ─ оператор, грамматический разбор которого выполняется;

─ language flag ─ указывает, как трактовать оператор; значение NATIVE;

─ 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*,
name *in varchar2*,
value *in varchar2*,
out_value_size *in integer*);

Здесь параметр 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

Осуществляется посредством функции EXECUTE. Описание её в модуле выглядит следующим образом:

function EXECUTE (c *in integer*) *return integer*;

Здесь 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 курсора

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.

Рассмотрим пример:

```
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 =:na
                    where fname=:fname and
                    lname=:lname';
    DBMS_SQL.PARSE (v_cursor_id, v_updatestmt, DBMS_SQL.NATIVE);
    DBMS_SQL.BIND_VARIABLE (v_cursor_id, :na, 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 нижеперечисленных 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 SQL-операторов *insert*, *update*, *delete*, обработка инструкций *select* включает дополнительно описание элементов списка выбора, считывание строк и запись результатов в переменные PL/SQL.

Процесс определение элементов списка выбора напоминает привязку входных переменных, за исключением того, что элементы списка выбора должны быть не привязаны, а только определены. В процедуре DEFINE_CLUMN 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 *number* , *date* данные этих типов не имеют фиксированной длины, заранее известной компилятору.

Таблица 1

Параметры DEFINE_COLUMN

Параметр	**Тип**	**Предназначение**
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. Как в DEFINE_COLUMN, так и в COLUMN_VALUE обычно используются одни и те же переменные
column_size	*integer*	Максимальный ожидаемый размер данных. Обязателен для тех типов, длина которых не известна 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 (c *in integer*)
return integer;

FETCH_ROWS возвращает число считываемых строк. FETCH_ROWS и COLUMN_VALUE вызывают в цикле несколько раз до тех пор, пока FETCH_ROWS не возвратит нуль.

После успешно выполненного считывания строк производят 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 её параметров – в табл. 2:

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*,
position *in integer*,
value *out varchar2*,
column_error *out number*,
actual_length *out number*);

Таблица 2

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
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)

В нижеследующем примере создается процедура, определяющая имена и фамилии сотрудников по заданной для данной процедуры должности сотрудника:

```
create or replace procedure DynamicQuery (p_position in staff.position %type) is
v_cursor_id integer;
v_select_stmt varchar2(500);
v_first_name staff.fname %type;
v_last_name staff.lname %type;
v_dummy integer;
begin
    v_cursor_id := DBMS_SQL.OPEN_CURSOR;
    v_select_stmt := 'select lname, fname
                      from staff
```

```
                    where position = :pos
                    order by lname';
    DBMS_SQL.PARSE (v_cursor_id, v_select_stmt, DBMS_SQL.NATIVE);
    DBMS_SQL.BIND_VARIABLE (v_cursor_id, ':pos', p_position);
    DBMS_SQL.DEFINE_COLUMN (v_cursor_id, 1, v_last_name, 25);
    DBMS_SQL.DEFINE_COLUMN (v_cursor_id, 2, v_first_name, 25);
    v_dummy := DBMS_SQL.EXECUTE (v_cursor_id);
    loop
        if DBMS_SQL.FETCH_ROWS (v_cursor_id)=0 then
            exit;
        end if;
        DBMS_SQL.COLUMN_VALUE (v_cursor_id, 1, v_last_name);
        DBMS_SQL.COLUMN_VALUE (v_cursor_id, 2, v_first_name);
        insert into temp_table (name_col)
        values (v_last_name || ' ' || v_first_name);
    end loop;
    DBMS_SQL.CLOSE_CURSOR (v_cursor_id);
    commit;
exception
    when others then
        --Закроем курсор, а затем повторно установим ошибку
        DBMS_SQL.CLOSE_CURSOR (v_cursor_id);
        raise;
end DynamicQuery;
```

Варианты заданий

1. Создать процедуру посредством динамического SQL, которая бы средствами серверного вывода представляла информацию о именах, номерах телефонов и адресах всех сотрудников, занимающих определенную должность; в выходном параметре процедуры определить их количество.

2. Создать функцию, подсчитывающую среднюю зарплату сотрудников в зависимости от места работы, информацию вынести во временную таблицу, возвращать среднее количество сотрудников на отделение. Функцию создавать средствами модуля DBMS_SQL.

3. Создать динамическую процедуру, осуществляющую поиск сотрудника по заданному шаблону и выводящую во временную таблицу всю не *nul*-информацию о нем, в выходном параметре процедуры передавать значение «*неопределенно*», в случае если поиск не дал результатов.

4. Создать динамическую функцию, удаляющую отделение по указанному в качестве аргумента адресу. Всех работающих в нем сотрудников перевести в отделение, так чтобы в среднем на отделение приходилось одинаковое количество работающих.

5. Создать динамическую процедуру, изменяющую значения заработной платы сотрудников на указанный в аргументе процент, адрес офиса с сотрудниками, получающими надбавку, указывать в качестве второго входного аргумента.

6. Создать динамическую функцию, определяющую, были ли осмотры на число, указанное в качестве её аргумента; возвращать количество найденных осмотров или *null*, если таковых не было.

7. Создать динамическую процедуру, осуществляющую добавление информации о новых объектах с автоматическим закреплением данного объекта за сотрудниками, у которых меньше объектов, чем в среднем по отделению; если таковых несколько, объект закрепить за первым в упорядоченном по алфавиту списке сотрудником.

8. Создать динамическую функцию, осуществляющую добавление информации о новых объектах с закреплением их за определенным сотрудником, имя и фамилия которого указываются во входных аргументах функции. Если у данного сотрудника количество закрепленных за ним объектов и так превышает среднее количество на человека по отделению, то отменить ввод. Возвращать TRUE или FALSE в зависимости от того, успешно или нет было выполнено добавление.

9. Создать динамическую процедуру, определяющую средний возраст сотрудников. Значение передавать в вызывающую среду посредством выходного параметра.

Контрольные вопросы

1. Поясните принципы серверного вывода в ORACLE 8*i*.
2. В чем преимущества процедур с динамическим SQL?
3. Расскажите о порядке выполнения DML-операторов средствами модуля DBMS_SQL.
4. Какие особенности в обработке динамических запросов на извлечение информации?

Литература

1. Коннолли, Т., Бегг, К. Базы данных: Проектирование, реализация и сопровождение. Теория и практика. – М., – Киев : Вильямс, 2000.

2. Бобровски, С. ORACLE 8: Архитектура : Основные принципы построения и структура баз данных ORACLE. – М. : Лори, 1998.

3. Урман, С. ORACLE 8*i* : Новые возможности программирования на языке PL/SQL. – М. : Лори, 2001.

Св. план 2006, поз. 36

Учебное издание

Капанов Николай Анатольевич

БАЗЫ ДАННЫХ САПР

Лабораторный практикум
для студентов специальности
I-53 01 07 «Информационные технологии и управление
в технических системах»

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

Подписано в печать 27.11.2006. Формат 60x84 1/16. Бумага офсетная.
Гарнитура «Таймс». Печать ризографическая. Усл. печ. л. 2,91.
Уч.-изд. л. 2,0. Тираж 150 экз. Заказ 168.

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