Федеральное государственное автономное образовательное учреждение
высшего образования
"Национальный исследовательский университет
"Высшая школа экономики"

Московский институт электроники и математики им. А.Н Тихонова

Департамент компьютерной инженерии

СОЗДАНИЕ ПРОЦЕДУРНЫХ ОБЪЕКТОВ В РЕЛЯЦИОННЫХ БАЗАХ ДАННЫХ

Методические указания к лабораторным работам № 6-7 по курсу "Базы данных"

Москва

2015

Составитель: доцент, канд. техн. наук И.П. Карпова

УДК 681.3

Создание процедурных объектов в реляционных базах данных: Методические указания к лабораторным работам № 6-7 по курсу "Базы данных" / Московский институт электроники и математики НИУ ВШЭ; Сост.: И.П. Карпова. – М., 2015. – 38 с.

Лабораторные работы посвящены изучению процедурных объектов языка SQL – триггеров, процедур и функций. Работы включают изучение правил создания и применения триггеров, процедур и функций на примере СУБД Oracle.

Для студентов II-IV курсов технических факультетов, изучающих автоматизированные информационные системы и системы баз данных.

Таблиц: 12. Ил.: 4. Библиогр.: 3 назв.

Содержание

ЦЕЛЬ ВЫПОЛНЕНИЯ РАБОТ

Цель выполнения лабораторных работ – изучение основ создания процедурных объектов для работы с базами данных и получение практических навыков работы с процедурными объектами в реляционных базах данных.

Выполнение работ включает создание двух триггеров (работа №6), процедуры и функции (работа №7) для базы данных, работающей под управлением СУБД Oracle.

1. ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

1.1. Общие положения

В настоящее время практически во всех ведущих СУБД в том или ином виде реализованы процедурные объекты, такие как процедуры, функции и триггеры. Это соответствует концепции *активной базы данных,* которая основана на идее включения процессов обработки данных в базе данных.

Процедурные объекты определены в стандартах языка SQL. Но, несмотря на наличие стандарта, разные СУБД имеют отличия в их реализации, поэтому мы будем изучать эти объекты на примере конкретной СУБД – Oracle. В этой СУБД для создания процедурных объектов используется специальный язык PL/SQL. PL/SQL – это принадлежащее Oracle процедурное расширение SQL, языка реляционных баз данных. PL/SQL полностью интегрирует современные средства разработки программного обеспечения, такие как инкапсуляция данных, скрытие информации, переопределение имен и обработка исключений.

В Oracle все объекты принадлежат тем пользователям, от имени которых они создаются. Таким образом, схема базы данных разбивается на *подсхемы*, каждая из которых называется по имени пользователя – владельца. Например, если таблицу EMP (Сотрудники) создал пользователь с именем SCOTT, то таблица располагается в подсхеме SCOTT и полное имя этой таблицы будет SCOTT.EMP – имя объекта указывается через точку после имени владельца.

! Обратите внимание: в рамках каждой подсхемы все имена объектов (вне зависимости от их типа) должны быть **уникальными.**

1.2. Основные сведения о языке PL/SQL

1.2.1. Алфавит

Алфавит языка включает следующие символы:

- Латинские буквы: a ... z A ... Z
- Арабские цифры: 0 ... 9
- Символы табуляция, пробел и возврат каретки ("пропуски")
- Символы ()+-*/<>=!~;:.'@%,"#$^&_|{ }?[ ]
- # $ _ – дополнительные символы, которые можно использовать как буквы

PL/SQL (как и SQL) не различает прописных и строчных букв, и рассматривает строчные буквы как эквиваленты соответствующих прописных букв, за исключением строковых и символьных литералов.

1.2.2. Лексические единицы

Строка текста программы PL/SQL распадается на группы символов, называемые *лексическими единицами*, которые можно классифицировать следующим образом:

- разделители (простые и составные символы),
- идентификаторы, в том числе зарезервированные слова,
- литералы,
- комментарии.

Простые символы кодируются как одиночные символы:

+	оператор сложения
-	оператор вычитания/отрицания
*	оператор умножения
/	оператор деления
=	оператор сравнения
<	оператор сравнения (меньше)
>	оператор сравнения (больше)
(	левый ограничитель выражения или списка
)	правый ограничитель выражения или списка
;	терминатор предложения
%	индикатор атрибута
,	разделитель элементов списка
.	селектор компоненты
@	индикатор удаленного доступа
'	ограничитель символьной строки
"	ограничитель идентификатора
:	индикатор хост-переменной

Составные символы кодируются как пары символов:

**	оператор возведения в степень
<>	оператор сравнения (не равно)
!=	оператор сравнения (не равно)
~=	оператор сравнения (не равно)
^=	оператор сравнения (не равно)
<=	оператор сравнения (меньше или равно)
>=	оператор сравнения (больше или равно)
:=	оператор присваивания
=>	оператор ассоциации
..	оператор интервала
||	оператор конкатенации символьных строк
<<	левый ограничитель метки
>>	правый ограничитель метки
--	индикатор однострочного комментария
/*	(начальный) ограничитель многострочного комментария
*/	(конечный) ограничитель многострочного комментария

Идентификатор – последовательность символов, которая начинается с латинской буквы (или знаков #, $, _) и содержит буквы, цифры или заменяющие буквы символы. Длина идентификатора не может превышать 30 символов.

Литерал – это явное число, символ, строка или булевское значение, не представленное идентификатором. Примерами могут служить числовой литерал 147 и булевский литерал FALSE.

Числовые литералы

Целочисленный литерал – это целое число с необязательным знаком и без десятичной точки. Примеры:

0 30 6 -14 0 +32767

Вещественный литерал – это целое или дробное число с необязательным знаком и с десятичной точкой. Примеры:

6.6667 0.0 -12.0 +8300.00 .5 25. 1.0E-7 3.14159e0

Булевские литералы

Булевские литералы – это предопределенные значения TRUE и FALSE, а также "не-значение" NULL, которое обозначает отсутствие, неизвестность или неприменимость значения.

Булевские литералы НЕ являются строками.

Символьные литералы

Символьный литерал – это одиночный символ, окруженный одиночными апострофами. Примеры:

'Z' '%' '7' ' ' 'z' '('

Строковые литералы

Строковый литерал – это последовательностью из нуля или более символов, заключенной в апострофы. Примеры:

'Hello, world!' 'XYZ Corporation' '10-NOV-91' '$1,000,000'

Все строковые литералы, за исключением пустой строки (''), имеют тип CHAR. Если необходимо включить апостроф в литерал, его необходимо изображать в виде двойного апострофа (''), что не то же самое, что двойная кавычка ("):

'Don''t leave without saving your work.'

Комментарии

-- однострочный комментарий

/* многострочный комментарий */

Комментарии нельзя вкладывать друг в друга.

1.2.3. Типы данных

Каждая константа и переменная имеет тип данных, который специфицирует ее формат хранения, ограничения и допустимый интервал значений. PL/SQL предусматривает разнообразие предопределенных скалярных и составных типов данных. *Скалярный* тип не имеет внутренних компонент. *Составной* тип имеет внутренние компоненты, которыми можно манипулировать индивидуально. Рис.1 показывает предопределенные типы данных PL/SQL. Скалярные типы распадаются на семейства числовых, символьных, календарных и булевских данных.

Скалярные типы		Составные типы		
BINARY_INTEGER DEC DECIMAL DOUBLE PRECISION FLOAT INT INTEGER NATURAL NUMBER NUMERIC POSITIVE REAL SMALLINT	CHAR CHARACTER LONG LONG RAW RAW ROWID STRING VARCHAR VARCHAR2	RECORD	TABLE	VARRAY
	DATE			
	BOOLEAN			

Рис.1. Предопределенные типы данных PL/SQL

Числовые типы данных приведены в Табл. 1, нечисловые типы – в Табл.2, в Табл. 3-4 – диапазоны значений и значения по умолчанию для разных типов данных.

Таблица 1. Числовые типы данных

Тип данных	Подтип	Описание
BINARY_INTEGER	NATURAL, NATURALN, POSITIVE, POSITIVEN, SIGTYPE	Целые числа со знаком. Использует библиотечную арифметику. NATURAL, NATURALN – только неотрицательные целые числа; последний запрещает null-значения. POSITIVE, POSITIVEN – только положительные целые числа; последний запрещает null-значения. SIGTYPE – знаковый тип: -1, 0 и 1.
NUMBER (точность, масштаб)	DEC, DECIMAL, DOUBLE PRECISION, FLOAT(точность), INTEGER, INT, NUMERIC, REAL, SMALLINT	Числа с фиксированной или плавающей точкой. Использует библиотечную арифметику.
PLS_INTEGER		Целые числа со знаком. Для ускорения вычислений использует машинную арифметику.

Таблица 2. Нечисловые типы данных

Тип данных	Подтип	Описание
CHAR(размер)	CHARACTER(размер)	Строки символов фиксированной длины. Максимальный размер 32767 байтов (для БД – 2000 байтов).
VARCHAR2(размер)	VARCHAR(размер), STRING	Строки символов переменной длины. Максимальный размер 32767 байтов (для БД – 4000 байтов).
DATE		Даты, часы, минуты, секунды.
BOOLEAN		Логические значения: TRUE – истина, FALSE – ложь, NULL – null-значения.
CLOB		Большие однобайтовые символьные объекты.
BLOB		Большие двоичные объекты.
BFILE		Указатели на объекты LOB, управляемые файловыми системами, внешними по отношению к БД.

Таблица 3. Диапазоны значений и значения по умолчанию для разных типов данных

Тип и параметры типа	Минимальное значение (размер)	Максимальное значение (размер)	Примечание	Значение параметров по умолчанию
Числовые типы				
BYNARY_ INTEGER	$-2^{31}-1$	$2^{31}-1$		–
NATURAL	0	2147483647		–
POSITIVE	1	2147483647		–
NUBMER [(точность, масштаб)]	1.0E-129	9.99E125	точность: 1÷38 масштаб: -84÷127	точность – 38 масштаб = 0
подтипы NUBMER	DEC, DECIMAL, PRECISION, DOUBLE FLOAT, SMALLIN,T INTEGER,NUMERIC,REAL,INT		аналогично базовому типу	
Календарный тип				
DATE	1.01.14712 г. до н.э.	31.12.314712г. н.э.	При отсутствии даты – первый день текущего месяца; при отсутствии времени – полночь.	
Тип "Идентификатор строки"				
ROWID			6-байтовые двоичные значения. Подтип типа CHAR.	

Таблица 4. Диапазоны значений и значения по умолчанию для разных типов данных

Тип и параметры типа	Минимальное значение (размер)	Максимальное значение (размер)	Примечание	Значение параметров по умолчанию
Символьные типы				
CHAR [(длина)]	1	32767		длина = 1
подтипы CHAR	**STRING, CHARACTER**		аналогично базовому типу	
VARCHAR 2 (длина)	1	32767		–
подтип VARCHAR2	VARCHAR		должен измениться	
LONG	1	32760		–
RAW (длина)	1	32767		–
LONG RAW	1	32760		–
Логический тип				
BOOLEAN	–	–	TRUE, FALSE, NULL	–

Перед присваиванием выбранного значения столбца переменной PL/SQL, если необходимо, преобразует это значение из типа данных исходного столбца в тип данных целевой переменной. Это происходит, например, когда выбирается значение столбца DATE в переменную VARCHAR2. Аналогично, перед присваиванием или сравнением значения переменной со значением столбца базы данных PL/SQL, если необходимо, преобразует значение из типа данных переменной в тип данных целевого столбца. Если PL/SQL не может определить, какое неявное преобразование необходимо, он выдает ошибку компиляции. В таких случаях необходимо использовать явные функции преобразования типов данных. В Табл.5 показано, какие неявные преобразования типов данных может выполнять PL/SQL, а в Табл.6 – функции для явных преобразований типов.

Таблица 5. Неявные преобразования типов данных

Откуда	Куда							
	BINARY_ INTEGER	CHAR	DATE	LONG	NUMBER	RAW	ROWID	VARCHAR2
BINARY_ INTEGER		да		да	да			да
CHAR	да		да	да	да	да	да	да
DATE		да		да				да
LONG		да				да		да
NUMBER	да	да		да				да
RAW		да		да				да
ROWID		да						да
VARCHAR2	да	да	да	да	да	да	да	

Таблица 6. Явные преобразования типов данных

Откуда	Куда				
	CHAR	DATE	NUMBER	RAW	ROWID
CHAR		TO_DATE	TO_NUMBER	HEXTORAW	CHARTORAWID
DATE	TO_CHAR				
NUMBER	TO_CHAR	TO_DATE			
RAW	RAWTOHEX				
ROWID	ROWIDTOCHAR				

1.2.4. Структура блоков

PL/SQL – это язык, структурированный блоками. Это значит, что основные единицы (процедуры, функции и анонимные блоки), составляющие программу PL/SQL, являются логическими блоками, которые могут содержать любое число вложенных в них подблоков. Обычно каждый логический блок соответствует некоторой проблеме или задаче, которую он решает. Таким образом, PL/SQL поддерживает подход к решению задач по принципу "разделяй и властвуй", известный как пошаговое уточнение.

Блок (или подблок) позволяет вам группировать логически связанные объявления и предложения. Благодаря этому можно размещать объявления близко к тем местам, где они используются. Объявления локальны в блоке, и перестают существовать, когда блок завершается. Как показывает Рис.2, блок PL/SQL имеет три части: декларативную часть, исполняемую часть и часть обработки исключений. (*Исключением* в PL/SQL называется условие, вызывающее предупреждение или ошибку.) Исполняемая часть обязательна; две остальные части блока могут отсутствовать.

```
[ DECLARE
      Объявления ]
BEGIN
      Выполняемые предложения
[ EXCEPTION
      Обработчики исключений ]
END;
```

Рис.2. Структура блока PL/SQL

Части программы, которые заключены в квадратные скобки [], являются необязательными.

1.2.5. Переменные и константы

PL/SQL позволяет объявить переменные и константы, а затем использовать их в SQL и процедурных предложениях в любом месте, где допускается использование выражения. Однако ссылки вперед не допускаются. Таким образом, необходимо объявить переменную или константу прежде, чем ссылаться на нее в других предложениях, в том числе в других объявлениях.

Объявления переменных. Объявляемая переменная может иметь любой тип данных, присущий SQL, такой как NUMBER, CHAR и DATE, или присущий PL/SQL, такой как BOOLEAN или BINARY_INTEGER. Например:

```
part_no NUMBER(4);      -- целая числовая переменная с 4-мя знаками
in_stock BOOLEAN;       -- логическая переменная
```

По умолчанию все переменные инициализируются значением NULL.

Присваивания переменным. Присваивать переменным значения можно при объявлении, с помощью оператора присваивания или указывая эту переменную в качестве приемника результата операции SELECT или FETCH. Примеры правильных присваиваний:

```
in_stock BOOLEAN := false;
tax := price * tax_rate;
raise := TO_NUMBER(SUBSTR('750 raise', 1, 3));
SELECT sal*0.1 INTO bonus FROM emp WHERE empno = emp_id;
```

После этого значение переменной bonus можно использовать в других вычислениях. **to_number** и **substr** – функции, описание которых можно посмотреть в Приложении 1.

Объявления констант. Объявление константы аналогично объявлению переменной, с той разницей, что необходимо добавить ключевое слово CONSTANT и немедленно присвоить константе значение. Впоследствии никакие присваивания константе не допускаются. Пример:

```
minimum_balance CONSTANT REAL := 10.00;
```

1.2.6. Атрибуты

Переменные и константы PL/SQL имеют АТРИБУТЫ, т.е. свойства, позволяющие ссылаться на тип данных и структуру объекта, не повторяя его объявление. Аналогичные атрибуты имеются у таблиц и столбцов базы данных, что позволяет упростить объявления переменных и констант.

Атрибут %TYPE представляет тип данных переменной, константы или столбца. Он особенно полезен при объявлении переменной, которая ссылается на столбец из таблицы базы данных. Например, предположим, что таблица books содержит столбец с именем title. Чтобы дать переменной my_title тот же тип данных, что у столбца title, не зная точного определения этого столбца в базе данных, надо объявить my_title с использованием атрибута %TYPE:

```
my_title books.title%TYPE;
```

Такое объявление переменной имеет еще одно преимущество: если определение столбца title в базе данных изменится (например, увеличится его длина), тип данных переменной my_title изменится соответственно во время выполнения.

В PL/SQL для группирования данных используются записи. Запись состоит из нескольких полей, в которых могут храниться значения данных. **Атрибут %ROWTYPE** обозначает тип записи, представляющей строку в таблице. Такая запись (т.е. переменная, объявленная с атрибутом %ROWTYPE) может хранить целую строку данных, выбранную из таблицы или извлеченную из курсора (что обсуждается позже). Столбцы в строке таблицы и соответствующие поля в записи имеют одинаковые имена и типы данных. В следующем примере объявляется запись с именем dept_rec. Ее поля имеют те же имена и типы данных, что и соответствующие столбцы в таблице dept.

```
DECLARE dept_rec dept%ROWTYPE;
```

Для обращения к значениям полей записи используются квалифицированные ссылки, как показывает следующий пример:

my_deptno := dept_rec.deptno;

Старшинство операций описано в Табл. 7.

Таблица 7. Старшинство операций

Оператор	Операция
**, NOT	возведение в степень, логическое отрицание
+, –	тождественность, отрицание
*, /	умножение, деление
+, –, ||	сложение, вычитание, конкатенация
=, !=, <, >, <=, >=, LIKE, IS NULL, BETWEEN, IN	сравнение
AND	конъюнкция
OR	дизъюнкция

1.2.7. Обработка неопределенных значений

Сравнения, в которых участвует NULL, **всегда дают NULL**; применение NOT к значению NULL дает NULL; в предложениях условного управления, если условие дает NULL, соответствующая группа предложений не выполняется.

PL/SQL трактует любую строку нулевой длины как NULL. Примеры:

```
x := 5;
y := NULL;
IF x != y THEN -- это условие даст NULL, а не TRUE
        -- ряд предложений; -- не выполняется
END IF;
```

Теперь рассмотрим пример ниже (Рис. 3). С точки зрения двухзначной логики (true / false) текст слева и справа идентичен: эти фрагменты программы выбирают максимальное из двух значений (***x*** и ***y***). Но с точки зрения трехзначной логики в первом случае переменной ***high*** будет присвоено значение ***y*** (null), а во втором ***high*** будет присвоено значение ***x*** (10).

```
x := 10;
y := NULL;
IF x > y
        THEN high := x;
        ELSE high := y;
END IF;
```

```
x := 10;
y := NULL;
IF NOT x > y
        THEN high := y;
        ELSE high := x;
END IF;
```

Рис.3. Пример влияния неопределенных значений

Если функции передается пустой аргумент, она возвращает NULL, за исключением следующих трех случаев:

DECODE

Функция DECODE сравнивает свой первый аргумент с одним или несколькими поисковыми выражениями, которые спарены с результирующими выражениями.

```
credit_limit := DECODE(rating, NULL, 1000, 'B', 2000, 'A',4000);
```

NVL

Если ее первый аргумент есть NULL, функция NVL возвращает значение своего второго аргумента:

```
start_date := NVL(hire_date, SYSDATE);
```

REPLACE

Если ее второй аргумент NULL, функция REPLACE возвращает значение своего первого аргумента, независимо от того, присутствует ли необязательный третий аргумент

```
new_string := REPLACE(old_string, NULL, my_string);
```

Если ее третий аргумент NULL, функция REPLACE возвращает значение своего первого аргумента, из которого удалены все вхождения второго аргумента.

1.2.8. Команды условного управления

Оператор **условного перехода IF** имеет две формы.

Форма 1:

```
IF  <условие>  THEN <ряд_предложений1>;
    [ ELSE <ряд_предложений2>; ]
END IF;
```

Форма 2:

```
IF  <условие1>  THEN  <ряд_предложений1>;
    ELSIF  <условие2>  THEN  <ряд_предложений2>;
    ...
    [ ELSE  <ряд_предложений3>; ]
END IF;
```

Пример:

```
IF sales > 50000  THEN  bonus := 1500;
    ELSIF sales > 35000  THEN  bonus := 500;
    ELSE bonus := 100;
END IF;
```

Команды циклов:

1) Цикл **LOOP – EXIT – END LOOP** (цикл с постусловием) :

```
LOOP
    <ряд_предложений>
    IF  <условие>  THEN  EXIT; -- немедленно выходит из цикла
    END IF;
...
END LOOP;
```

2) Цикл **LOOP – EXIT – END LOOP** (цикл с предусловием):

```
LOOP
        <ряд_предложений>
        EXIT WHEN <условие>; -- выйти из цикла при условии
        ...
END LOOP;
```

Метки цикла:

```
<<my_loop>> LOOP  ...  END LOOP my_loop;
```

Цикл **WHILE – LOOP – END LOOP** (цикл с предусловием):

```
WHILE  <условие>  LOOP
        <ряд_предложений>;
END LOOP;
```

3) Цикл **FOR – LOOP – END LOOP** (цикл со счетчиком):

```
FOR  <переменная цикла>  IN  [REVERSE]  нижняя_граница..верхняя_граница
LOOP
        <ряд_предложений>
END LOOP;
```

<переменная цикла> – целочисленная переменная, может не объявляться в части DECARE.

<нижняя_граница>, <верхняя_граница> – переменные или константы.

Пример:

```
FOR  i  IN  REVERSE  1..3  LOOP -- присваивает  i  значения  3, 2, 1
        <ряд_предложений>; -- будет выполнен три раза
END LOOP;
```

Предложение EXIT позволяет завершить цикл FOR прежде времени.

```
FOR  j  IN 1..10 LOOP
        ...
        EXIT  WHEN  <условие>;
        ...
END  LOOP;
```

```
<<outer>>
FOR  i  IN 1..5 LOOP ...
        FOR  j  IN 1..10 LOOP
                ...
                EXIT  outer  WHEN  <условие>; -- выход из обоих циклов
                ...
        END  LOOP;
END  LOOP  outer; -- управление будет передано сюда
```

```
BEGIN
...
GOTO  insert_row;
...
<<insert_row>>  INSERT  INTO  emp  VALUES...
...
END;

DECLARE  done  BOOLEAN;
BEGIN
        ...
        FOR  i  IN  1..50  LOOP
                IF  done  THEN  GOTO  end_loop;
                END  IF;
                ...
        <<end_loop>>   null;
        END LOOP; -- это не выполняемое предложение
END;
```

Предложение GOTO не может передавать управление:

- в предложение IF,
- в предложение LOOP,
- в подблок,
- из подпрограммы в окружающий блок,
- из обработчика исключений в текущий блок.

1.3. Подпрограммы

Подпрограмма – это поименованный блок PL/SQL, который принимает параметры и может быть вызван.

PL/SQL имеет два типа подпрограмм:

- процедуры (procedure);
- функции (function).

Обычно процедуру вызывают для того, чтобы выполнить некоторое действие, а функцию – для того, чтобы вычислить некоторое значение.

Подпрограммы можно определять в любом инструменте ORACLE, который поддерживает PL/SQL. Их можно объявлять:

- в блоках PL/SQL,
- в процедурах,
- в функциях,
- в пакетах.

Подпрограммы должны объявляться в конце декларативной секции, после всех других программных объектов.

Синтаксис описания процедуры:

procedure <имя процедуры> [(<список формальных параметров>)] is
    [-- объявления]

```
begin
    -- выполняемые действия
[ exception
    -- обработчики исключительных ситуаций ]
end [<имя процедуры>];
```

Синтаксис описания функции:

```
function <имя функции> [(<список формальных параметров>)]
return  <тип возвращаемого значения>  is
   [ -- объявления ]
begin
    -- выполняемые действия
    return  <возвращаемое значение>;
    ...
[ exception
    -- обработчики исключительных ситуаций ]
end [<имя функции>];
```

В списке параметров каждый параметр описывается следующим образом:

```
<имя переменной>  [ IN | OUT | IN OUT ] <тип данных>
        [ { := | DEFAULT } <значение> ]
```

Тип данных не может иметь ограничений по размеру.
IN – мода входного параметра (по умолчанию);
OUT – мода выходного параметра;
IN OUT – мода входного и выходного параметра.

Значение по умолчанию (DEFAULT) присваивается в той ситуации, когда при вызове процедуры (функции) фактический параметр не передается.

Пример:

```
declare
vdate  date;
function dt (v char)  return  date  is
begin
   return  to_date(v, 'dd-mm-yyyy');
end;
procedure period (d1 date, d2 date default trunc(sysdate))   is
begin
       ...
end;
begin
   vdate := dt ('15-10-2012');
   period (vdate);
   ...
end;
```

Вызовы пользовательских функций могут появляться в процедурных предложениях, но НЕ в предложениях SQL.

Три возможные моды (режима передачи): IN (умалчиваемая), OUT и IN OUT. Они могут использоваться в любой процедуре.

Параметр **с модой IN** передает значение вызываемой подпрограмме. Внутри подпрограммы такой параметр выступает как константа, поэтому ему нельзя присвоить значение. Параметры IN могут инициализироваться умалчиваемыми значениями.

Параметр **с модой OUT** позволяет возвращать значение вызывающей программе. Внутри подпрограммы такой параметр выступает как неинициализированная переменная. Поэтому его значение нельзя присваивать другим переменным или переприсвоить самому себе.

Параметр **IN OUT** позволяет передавать в подпрограмму начальные значения и возвращать обновленные значения вызывающей программе. Внутри подпрограммы такой параметр выступает как инициализированная переменная. Поэтому ему можно присвоить значение, а его значение можно присваивать другим переменным.

В функциях следует избегать использования моды OUT или IN OUT.

В Табл.8 описаны основные характеристики мод IN, OUT, IN OUT.

Таблица 8. Основные характеристики мод IN, OUT, IN OUT

IN	OUT	IN OUT
по умолчанию	задается явно	задается явно
передает значение подпрограмме	возвращает значение вызывающей подпрограмме	передает начальное значение подпрограмме; возвращает обновленное значение вызывающей подпрограмме
формальный параметр выступает как константа	формальный параметр может использоваться в выражениях; ему должно быть присвоено значение	формальный параметр выступает как инициализированная переменная
фактический параметр может быть константой, инициализированной переменной, литералом или выражением	фактический параметр должен быть переменной	фактический параметр должен быть переменной

Предложение RETURN немедленно завершает выполнение подпрограммы и возвращает управление вызывающей программе. Выполнение продолжается с предложения, следующего за вызовом подпрограммы.

Подпрограмма может содержать несколько предложений RETURN, ни одно из которых не обязано быть последним лексическим предложением в подпрограмме. Выполнение любого из них немедленно завершает подпрограмму. Однако наличие в подпрограмме нескольких точек выхода не является хорошей практикой программирования.

В процедурах предложение RETURN не может содержать выражение. Это предложение просто возвращает управление вызывающей программе до достижения нормального конца процедуры. Однако в функциях предложение RETURN ДОЛЖНО содержать выражение, которое вычисляется при выполнении предложения RETURN.

Результирующее значение присваивается идентификатору функции. Поэтому функция должна содержать хотя бы одно предложение RETURN. В противном случае PL/SQL возбуждает предопределенное исключение PROGRAM_ERROR во время выполнения.

PL/SQL требует, чтобы идентификатор был объявлен ДО его использования. Поэтому необходимо объявить подпрограмму, прежде чем вызывать ее.

Для рекурсивных подпрограмм, для объявления подпрограмм в алфавитном порядке и проч. предусмотрена возможность упреждающих объявлений:

```
DECLARE
PROCEDURE calc_rating (...); -- упреждающее объявление
/* Определить подпрограммы в алфавитном порядке. */
PROCEDURE award_bonus (...) IS ...
BEGIN
calc_rating(...);
...
```

PL/SQL позволяет определять несколько процедур (функций), имеющих одинаковые имена, но отличающихся набором параметров. Одноименные подпрограммы должны иметь разное количество параметров или, если количество параметров одинаково, они должны принадлежать разным семействам типов данных. На рис.4 приведен алгоритм разрешения вызова процедуры (функции).

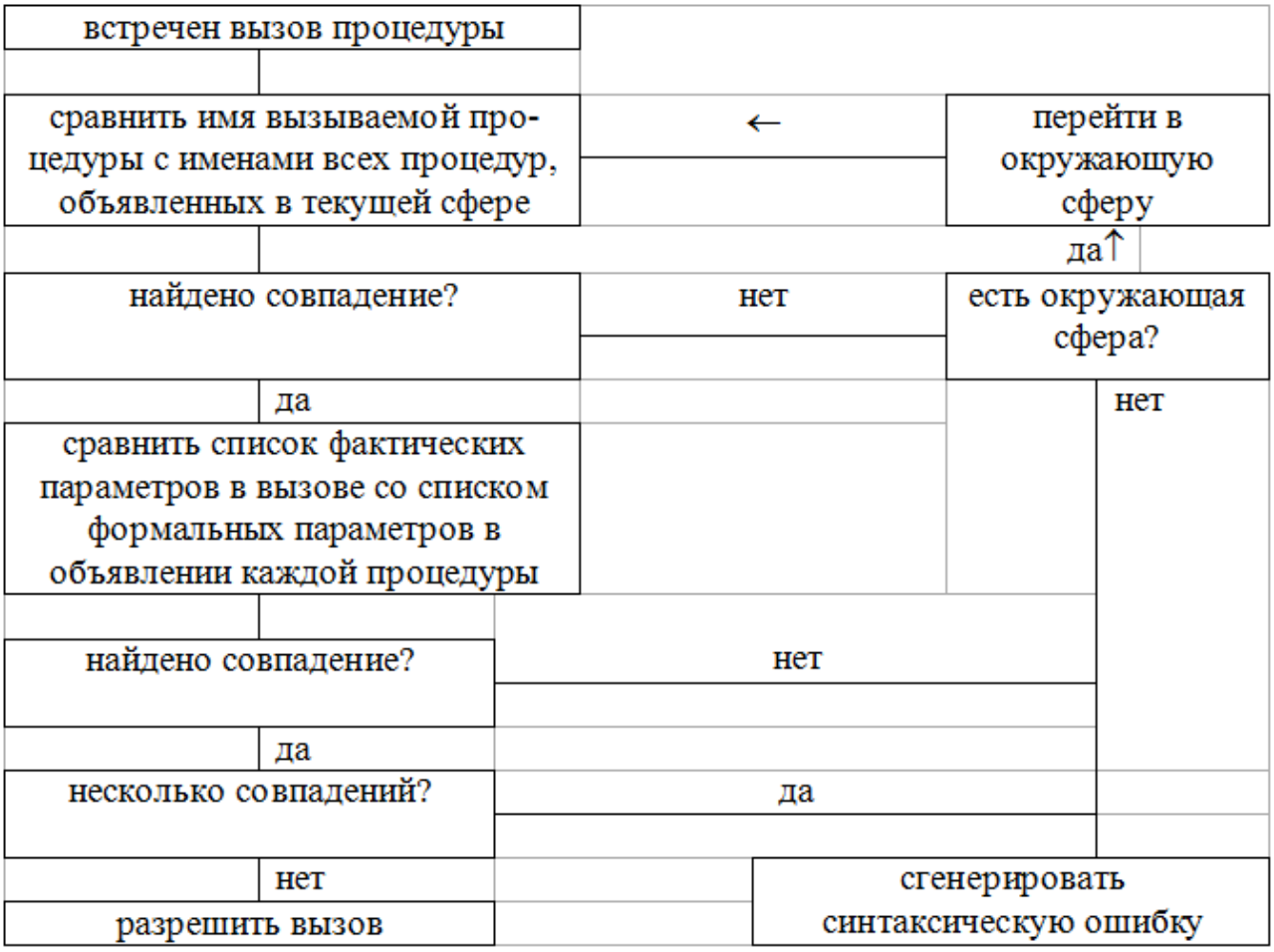

Рис.4. Алгоритм разрешения вызова процедуры (функции)

Рассмотрим пример функции, которая выполняет периодическую фиксацию транзакции. Параметры: m – текущее количество изменений, period – период фиксации. Функция увеличивает m на 1 и проверяет, достигнуто ли значение period. Если нет, возвращает увеличенное значение; если да, то commit и проверка таблицы break_table. Если break_table не пуста, то функция возвращает отрицательное число, иначе m обнуляется и возвращается 0.

```
function inc(m IN binary_integer, period binary_integer)
  return binary_integer is
v_break integer;
begin
  m := m+1;
  if m >= period then
    update prg set id = id+m where rownum < 2;
    commit;
    select count(*) into v_break from break_table;
    if v_break = 0 then m := 0; else return -1; end if;
  end if;
  return 0;
end;
```

Пример процедуры: процедура создания заказа на товары, которых осталось меньше допустимого минимума. Параметры: категория товара и коэффициент, определяющий количество заказываемого товара.

```
procedure  (cat  varchar, coeff  number)  order_goods
  IS
    num  number:=0;  -- количество заказываемых позиций
  BEGIN
    dbms_output.put_line('-=ORDER=-');  -- стандартная процедура вывода на экран
    for  smth  in
           (select  *  from  goods
                 where  gtype = cat and  -- выбираем товар нужной категории
                  ostatok < stock_min)   -- запасы которого подходят к концу
       loop
            num:=num+1;
            dbms_output.put_line('NAME: ' || smth.name || ' Weight: ' || smth.weight ||
                ' Unit: ' || smth.unit || ' AMOUNT NEEDED: ' || smth.stock_min*coeff);
    end loop;
    if  num = 0 then
       dbms_output.put_line(' order_goods: no goods found to order ');
    end  if;
  END;
/
```

1.4. Хранимые процедуры и функции

Операторы создания процедуры или функции Oracle использует следующий синтаксис:

```
create  [or replace]  procedure
   [имя_схемы.]имя_процедуры
   [(имя_параметра  [{ in | out | in out }]  тип_данных
   [, имя_параметра  [{ in | out | in out }]  тип_данных …])]
 { is | as }
программа_на_PL/SQL
```

```
create  [or replace]  function
   [имя_схемы.]имя_функции
   [(имя_параметра  [{ in | out | in out }]  тип_данных
   [, имя_параметра  [{ in | out | in out }]  тип_данных …])]
   return  тип_данных
  { is | as }
программа_на_PL/SQL
```

Для создания процедуры или функции необходимо иметь привилегию CREATE PROCEDURE, а для создания процедуры или функции в чужой схеме – привилегию CREATE ANY PROCEDURE.

Если операторы CREATE PROCEDURE / CREATE FUNCTION выполняются в приложениях Oracle SQL*Plus или SQL WorkSheet, то на экран выдается сообщение:

Procedure / Function created (Процедура /функция создана)

или

Procedure / Function created with errors (Процедура /функция создана с ошибками)

Во втором случае с помощью команды SHOW ERROR можно посмотреть список ошибок.

Если же вы работаете в Oracle Express Edition, то перечень ошибок выдается на экран автоматически.

Хранимая функция вызывается из другой функции (процедуры) или из команды SELECT, например:

```
select  to_char(sysdate, 'yyyy/mm/dd hh24:mi:ss')  from  dual;
```

Здесь to_char – функция преобразования значения первого параметра в строку, sysdate – функция, возвращая текущую дату и время, 'yyyy/mm/dd hh24:mi:ss' – формат выдачи даты-времени, dual – специальная таблица Oracle, которая имеет один столбец и одну строку.

Хранимую процедуру можно вызвать следующим образом.

1) Из другой подпрограммы:

```
create_dept(name, location);
```

2) Из прикладной программы: приложение прекомпилятора или приложение OCI может вызывать хранимые подпрограммы из анонимных блоков PL/SQL.

```
EXEC SQL EXECUTE
        BEGIN create_dept(:name, :location); END;
END-EXEC;
```

Фактические параметры :name и :location – это хост-переменные.

```
EXEC SQL EXECUTE
        BEGIN emp_actions.create_dept(:name, :location); END;
END-EXEC;
```

3) Из инструмента ORACLE:

```
SQL> EXECUTE create_dept('MARKETING', 'NEW YORK');
SQL> BEGIN create_dept('MARKETING', 'NEW YORK'); END;
```

1.5. Обработка исключительных ситуаций

Исключительная ситуация – это ситуация, при которой СУБД не может дальше продолжать работу (ситуация возникновения ошибки). **Исключение** – это условие ошибки. Преимущество такого подхода: централизованная обработка ошибок.

Исключительные ситуации бывают:

- Предопределенные (внутренние) – определенные по умолчанию.
- Пользовательские – определенные пользователем.

Внутренние исключения возбуждаются неявно (автоматически) системой исполнения; пользовательские исключения возбуждаются явно, посредством

предложений RAISE, которые могут также возбуждать предопределенные исключения. При возникновении исключительной ситуации:

- возбуждается исключение;
- программа прекращает нормальную работу, переходит в блок обработки исключительных ситуаций (exception), а затем возвращает управление в окружающую среду (сферу).

Когда возбуждается исключение, нормальное исполнение блока PL/SQL или подпрограммы останавливается, и управление передается на обработчик исключений этого блока или подпрограммы, что оформляется следующим образом:

```
EXCEPTION
  WHEN  имя_исключения1  THEN – обработчик
      ряд_предложений1
WHEN  имя_исключения2  [ OR  имя_исключения2  … ]  THEN
      ряд_предложений2       -- другой обработчик
      ...
WHEN  OTHERS  THEN       -- необязательный обработчик
ряд_предложений3
END;
/
```

Обращения к исключениям происходит по имени. Каждое исключение может упоминаться в части EXCEPTION **один раз**! Если один обработчик предназначен для обработки нескольких исключений, то их имена могут указываться в одном операторе WHEN с помощью логического оператора OR.

Необязательный обработчик исключений OTHERS – всегда **последний** обработчик исключений в блоке. Он всегда употребляется **отдельно.**

Пример обработки исключительной ситуации:

```
DECLARE  pe_ratio  NUMBER(3,1);
BEGIN
...
  SELECT  price / earnings  INTO  pe_ratio  FROM  stocks
      WHERE  symbol = 'XYZ';
-- может вызвать ошибку "деление на 0"
  INSERT  INTO  stats (symbol, ratio)  VALUES ('XYZ', pe_ratio);
  COMMIT;
EXCEPTION -- здесь начинаются обработчики исключений
  WHEN  ZERO_DIVIDE  THEN -- обрабатывает "деление на 0"
    INSERT  INTO  stats (symbol, ratio)  VALUES ('XYZ', NULL);
    COMMIT;
...
WHEN  OTHERS  THEN -- обрабатывает все прочие ошибки
   ROLLBACK;
END; -- здесь заканчиваются обработчики исключений и весь блок
/
```

Предопределенные поименованные исключительные ситуации приведены в Табл. 9.

Таблица 9. Предопределенные исключительные ситуации

Имя исключения	Ошибка ORACLE	Код SQLCODE
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
STORAGE_ERROR	ORA-06500	-6500
TIMEOUT_ON_RESOURCE	ORA-00051	-51
TOO_MANY_ROWS	ORA-01422	-1422
TRANSACTION_BACKED_OUT	ORA-00061	-61
VALUE_ERROR	ORA-06502	-6502
ZERO_DIVIDE	ORA-01476	-1476

Описание преопределенных исключительных ситуаций:

CURSOR_ALREADY_OPEN – возбуждается при попытке открыть уже открытый курсор. Курсорный цикл FOR автоматически открывает свой курсор. Поэтому нельзя войти в такой цикл, если данный курсор уже открыт. Нельзя явно открывать курсор внутри цикла FOR.

DUP_VAL_ON_INDEX – возбуждается, когда операция INSERT или UPDATE пытается создать повторяющееся значение в столбце, ограниченном опцией UNIQUE.

INVALID_CURSOR – возбуждается, когда вызов PL/SQL специфицирует некорректный курсор (например, при попытке закрыть неоткрытый курсор).

INVALID_NUMBER – возбуждается в предложении SQL, когда преобразование символьной строки в число сбивается из-за того, что строка не содержит правильного представления числа. В процедурных предложениях вместо этого исключения возбуждается VALUE_ERROR.

LOGIN_DENIED – возбуждается при некорректном имени пользователя или пароле при попытке подключения к ORACLE.

NO_DATA_FOUND – возбуждается, когда предложение SELECT INTO не возвращает ни одной строки, или при обращении к неинициализированной строке таблицы PL/SQL. Групповые функции SQL, такие как AVG или SUM, ВСЕГДА возвращают значение, даже если это значение есть NULL. NO_DATA_FOUND возбуждается, когда предложение SELECT INTO не возвращает строк, поэтому проверять значение SQL%NOTFOUND можно только в обработчике исключений.

NOT_LOGGED_ON – возбуждается, когда ваша программа PL/SQL выдает вызов ORACLE, не будучи подключена к ORACLE.

PROGRAM_ERROR – возбуждается, когда PL/SQL встретился с внутренней проблемой.

STORAGE_ERROR – возбуждается, когда PL/SQL исчерпал доступную память, или когда память запорчена.

TIMEOUT_ON_RESOURCE – возбуждается при возникновении таймаута, когда ORACLE ожидает ресурса.

TOO_MANY_ROWS – возбуждается, когда предложение SELECT INTO возвращает больше одной строки.

TRANSACTION_BACKED_OUT – обычно возбуждается, если удаленная часть транзакции была подвергнута неявному или явному откату. В таких случаях нужно выдать ROLLBACK, а затем повторить транзакцию.

VALUE_ERROR – возбуждается при возникновении арифметической ошибки, ошибки преобразования, ошибки усечения или ошибки ограничения. Например, VALUE_ERROR возбуждается при усечении строкового значения, присваиваемого переменной PL/SQL. (Однако при усечении строкового значения, присваиваемого хост-переменной, никакого исключения не возбуждается.) В процедурных предложениях VALUE_ERROR возбуждается при ошибке преобразования символьной строки в число. В предложениях SQL в таких случаях возбуждается INVALID_NUMBER.

ZERO_DIVIDE –возбуждается при попытке деления числа на 0.

Пользовательские исключения

Объявление пользовательских исключений:

<имя исключения> EXCEPTION;

Возбуждение пользовательских исключений происходит с помощью команды RAISE или вызова процедуры RAISE_APPLICATION_ERROR().

Пример:

```
DECLARE
   salary_error  EXCEPTION;
…
BEGIN
  FOR r IN (SELECT * FROM EMP) LOOP
    IF NOT (r.salary BETWEEN 5600 AND 100000)
      THEN RAISE salary_error;
      ELSE …
    END IF;
  END LOOP;
  COMMIT;
EXCEPTION WHEN salary_error THEN
  RAISE_APPLICATION_ERROR(-20100,
    'Недопустимая зарплата: меньше 5600 или больше 100000');
END;
/
```

Любое исключение можно перехватить и обработать только по имени. Если у предопределенного исключения нет имени, то для обработки непоименованных внутренних исключений можно использовать обработчик OTHERS или прагму EXCEPTION_INIT.

Прагма – это директива (указание) компилятору. Прагмы обрабатываются во время компиляции, а не во время выполнения.

Прагма EXCEPTION_INIT сообщает компилятору имя исключения, которое будет ассоциироваться с конкретным кодом ошибки ORACLE. Это позволяет обращаться к любому внутреннему исключению по имени, написав для него специальный обработчик. Синтаксис:

PRAGMA EXCEPTION_INIT(имя_исключения, код_ошибки_ORACLE);

Здесь *имя_исключения* – это имя исключения, ранее уже объявленного в этом блоке. Прагма должна появиться в той же декларативной части, что и соответствующее исключение.

Пример:

```
DECLARE
   insufficient_privileges  EXCEPTION;
PRAGMA  EXCEPTION_INIT(insufficient_privileges, -1031);
-- ORACLE возвращает код ошибки -1031, если, например,
-- пользователь пытается обновить таблицу, для которой
-- имеет лишь полномочия SELECT.
BEGIN
...
EXCEPTION
   WHEN  insufficient_privileges  THEN -- обработать ошибку ...
END;
/
```

При возникновении исключительной ситуации:

1. Возбуждается исключение:
   - Устанавливаются значения SQLCODE, SQLERRM.
   - Происходит выход из текущего курсорного цикла, курсор при этом закрывается.
2. Управление передается в блок обработки исключительных ситуаций (exception).
3. Выполняется обработка исключительной ситуации.
4. Управление передается в окружающий блок.

Если в текущем блоке нет обработчика исключений для возбужденного исключения, то в этом случае блок завершается, а исключение *продвигается* в окружающий блок. Теперь окружающий блок становится текущим, а исключение воспроизводит себя в нем. Если обработчик исключения не находится и в этом блоке, процесс поиска повторяется. Если текущий блок не имеет окружающего блока, PL/SQL возвращает ошибку "необрабатываемое исключение" в хост-окружение.

Перед выполнением блока PL/SQL или хранимой подпрограммы ORACLE устанавливает неявную точку сохранения. Если блок или подпрограмма сбивается в результате необработанного исключения, ORACLE осуществляет откат к этой точке сохранения.

Исключение можно обработать несколько раз. Пример повторной обработки исключения:

```
DECLARE
      out_of_balance  EXCEPTION;
BEGIN ...
      ---------------- начало подблока ------------------------
      BEGIN ...
         IF ... THEN  RAISE  out_of_balance; -- возбудить исключение
         END IF;
         ...
      EXCEPTION  WHEN  out_of_balance  THEN -- обработать ошибку
         RAISE;              -- повторно возбудить текущее исключение
      END;
      ---------------- конец подблока -------------------------
EXCEPTION
       WHEN  out_of_balance  THEN -- обработать ошибку иным способом
      ...
END;
/
```

Исключения могут также возбуждаться:

- в части DECLARE (некорректными выражениями инициализации в объявлениях);
- в части EXCEPTION.

Пример:

```
CREATE  TABLE  err_tabs (num  number(5),  msg  varchar(30),  edate  date);
DECLARE
      limit  NUMBER(3) := 5000; -- возбуждает исключение VALUE_ERROR
BEGIN
      ...
EXCEPTION
      WHEN  TOO_MANY_ROWS  THEN  INSERT  INTO  err_tabs
      VALUES(-1422, 'Команда SELECT вернула более одной строки‘, sysdate);
      -- возбудит исключение VALUE_ERROR, т.к. строка больше 30 символов,
      -- и продвинет его в окружающий блок
      WHEN VALUE_ERROR THEN … -- не перехватит исключение из DECLARE
      WHEN  OTHERS  THEN …     -- не перехватит исключение из EXCEPTION
END;
/
```

Встроенные функции обработки ошибок

SQLCODE

Функция SQLCODE возвращает код ошибки для последнего (текущего) исключения в блоке. При отсутствии ошибок SQLCODE возвращает 0. Функция также возвращает 0, если она вызывается извне обработчика исключений.

SQLERRM

Функция SQLERRM возвращает сообщение об ошибке по её коду. Если не передать SQLERRM код ошибки, то будет выдано сообщение об ошибке с кодом, возвращённым функцией SQLCODE. Максимальная длина строки, возвращаемой SQLERRM, составляет 512 байт (в некоторых более ранних версиях Oracle – всего 255 байт).

SQLERRM возвращает сообщение, ассоциированное с возникшей ошибкой ORACLE. Это сообщение начинается с кода ошибки ORACLE.

Для пользовательских исключений, SQLCODE возвращает +1, а SQLERRM возвращает сообщение User-Defined Exception если не была использована прагма EXCEPTION_INIT, чтобы ассоциировать исключение с номером ошибки ORACLE; в этом случае SQLCODE возвращает этот номер ошибки, а SQLERRM возвращает соответствующее сообщение об ошибке.

Если не возбуждено никакое исключение, то SQLCODE возвращает 0, а SQLERRM возвращает сообщение ORA-0000: normal, successful completion

Если передать функции SQLERRM номер ошибки, то она возвратит сообщение, ассоциированное с этим номером ошибки. Номер ошибки, передаваемый SQLERRM, должен быть отрицателен (за исключением +100 – no data found).

Нельзя использовать функции SQLCODE и SQLERRM непосредственно в предложениях SQL, можно только в процедурных предложениях.

Пример использования встроенных функций обработки ошибок:

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

Полезные приемы: продолжение работы после возбуждения исключения. Пример:

```
DECLARE
        pe_ratio  NUMBER(3,1);
BEGIN
        DELETE  FROM  stats  WHERE  symbol = 'XYZ';
        ---------------- начало подблока ------------------
        BEGIN
                SELECT price / NVL(earnings,0)  INTO  pe_ratio
                    FROM  stocks  WHERE  symbol = 'XYZ';
           EXCEPTION  WHEN  ZERO_DIVIDE  THEN  pe_ratio := 0;
        END;
        ---------------- конец подблока -------------------
        INSERT  INTO  stats (symbol, ratio)  VALUES ('XYZ', pe_ratio);
   EXCEPTION
        ...
END;
/
```

Полезные приемы: продолжение работы после возбуждения исключения. Пример:

```
DECLARE
        vdate  date;
        vemp  emp%ROWTYPE;
        vid      number(5);
BEGIN
        SELECT  *  INTO  vemp  FROM  emp  WHERE  tabno = &vid;
        ---------------- начало подблока ------------------
        BEGIN
            vdate := to_date(vemp.born, 'yyyy/mm/dd');
            EXCEPTION
               WHEN  VALUE_ERROR  THEN
                  INSERT  INTO  errors  VALUES(…);
        END;
        ---------------- конец подблока -------------------
        …
        EXCEPTION
        ...
END;
/
```

Полезные приемы: повтор транзакции. Пример:

```
DECLARE  name  CHAR(20);
                ans1  CHAR(3);    ans2  CHAR(3);
                ans3  CHAR(3);    suffix  NUMBER := 1;
        BEGIN
        ...
```

```
        FOR  I  IN  1..10  LOOP
        BEGIN ------------------- начало подблока -------------------
           SAVEPOINT  start_transaction; -- точка сохранения
           /* Удалить результаты опроса. */
           DELETE  FROM  results  WHERE  answer1 = 'NO';
           /* Добавить имя и ответы респондента. */
           INSERT  INTO  results  VALUES (name, ans1, ans2, ans3);
          /* Это может дать исключение DUP_VAL_ON_INDEX, если два
             респондента имеют одинаковые имена, так как индекс по столбцу
             name уникальный. */
           COMMIT;
           EXIT;
        EXCEPTION   WHEN  DUP_VAL_ON_INDEX  THEN
           ROLLBACK  TO  start_transaction; -- откат
           suffix := suffix + 1; -- попробуем исправить имя
           name := name || TO_CHAR(suffix);
           ...
        END; ------------------- конец подблока --------------------
        END LOOP;
    END;
    /
```

1.6. Триггеры

Триггер – это процедура, которая автоматически запускается при возникновении определенного события.

Событие триггера – событие, управляющее запуском триггера; описывается в виде логических условий.

В Oracle различают следующие типы триггеров:

- **простые триггеры** – они привязаны к определённой таблице, срабатывают при поступлении команд DML (Data Manipulation Language) и выполняются в рамках этой команды;
- **триггеры INSTEAD OF** – они привязаны к определённой таблице, выполняются вместо события триггера, которое является командой DML;
- **триггеры для событий уровня схемы** – они срабатывают при выполнении команд DDL (Data Definition Language) и при наступлении таких событий, как подключение и отключение от базы данных, а также возникновение серверной ошибки.

В рамках работы №6 будут изучаться только простые триггеры. Эти триггеры обычно предназначены для:

- проверки сложных ограничений целостности;
- автоматизации обработки данных;
- аудита (отслеживания) действий пользователей;
- установки начальных значений при добавлении данных в таблицы;

– проверки дифференцированных прав доступа.

Триггеры входят в стандарт SQL/1999. Общий синтаксис создания триггера в соответствии с этим стандартом приведен ниже:

```
trigger_definition ::=
    CREATE TRIGGER trigger_name
        { BEFORE | AFTER }
        { INSERT | DELETE | UPDATE [ OF column_commalist ] }
        ON table_name
        [ REFERENCING old_or_new_values_alias_list ]
        triggered_action
    triggered_action ::= [ FOR EACH { ROW | STATEMENT } ]
        [ WHEN  conditional_expression  ]    triggered_SQL_statement
    triggered_SQL_statement ::= SQL_procedure_statement
        | BEGIN ATOMIC
              SQL_procedure_statement_semicolonlist
        END
    old_or_new_values_alias ::= OLD [ ROW ] [ AS ] correlation_name
        | NEW [ ROW ] [ AS ] correlation_name
        | OLD TABLE [ AS ] identifier
        | NEW TABLE [ AS ] identifier
```

В Oracle команда создания простого триггера имеет следующий синтаксис:

```
    CREATE  [OR REPLACE]  TRIGGER  [имя_пользователя.]имя_триггера
        { BEFORE | AFTER }
        { INSERT | DELETE | UPDATE  [ OF список_столбцов ] }
        ON  [имя_пользователя.]имя_таблицы
        [ REFERENCING old_or_new_values_alias_list ]
        [ FOR EACH  { ROW | STATEMENT } ]
        [ WHEN  условие]
    [ DECLARE
    -- описание переменных, констант и др. элементов программы
    ]
    BEGIN
        -- программа на процедурном языке (PL/SQL)
        [ EXCEPTION
        -- обработка исключительных ситуаций
        ]
    END;
    /
```

Рассмотрим отдельные элементы этой команды:

CREATE [OR REPLACE] TRIGGER – команда создания (или переопределения) триггера.

имя_триггера – обычный идентификатор, должен начинаться с латинской буквы или знаков #, $ или _, может включать латинские буквы, цифры и указанные знаки. Если не указано [*имя_пользователя*.], то триггер создается в подсхеме текущего пользователя.

INSERT | DELETE | UPDATE [of *список_столбцов*] – событие триггера. Событием триггера может быть одна команда или любая комбинация указанных команд, при выполнении которых активизируется триггер.

BEFORE | AFTER – время срабатывания триггера: перед выполнением события триггера или после него. Обратите внимание: ограничения целостности проверяются во время выполнения команды – события триггера.

ON *имя таблицы* – таблица, к которой привязан триггер. Если не указано [*имя_пользователя*.], таблица должна принадлежать тому пользователю, от имени которого создается триггер.

FOR EACH { ROW | STATEMENT } – область действия триггера (для каждой строки или для предложения). По умолчанию создается триггер уровня предложения, поэтому для него область действия можно не указывать.

REFERENCING *old_or_new_values_alias_list* – переопределение корреляционных имен :*old* и :*new*, с помощью которых внутри триггера уровня строки можно обращаться к старому и новому значениям данных в этой строке (записи). Например:

REFERENCING old AS old_emp new AS new_emp

WHEN *условие* – условие срабатывания триггера. Если оно не выполняется, триггер не будет запущен.

DECLARE – начало необязательной декларативной части, в которой описываются переменные, константы и другие элементов программы.

BEGIN … END – операторные скобки, ограничивающие исполняемый код (блок) программы. В Oracle программа пишется на внутреннем процедурном языке PL/SQL.

EXCEPTION – необязательная часть обработки исключительных ситуаций.

Обратите внимание: после завершения кода программы необходимо добавить строку, в которой будет символ '/' – он означает завершение ввода текста программы. Если триггер будет создан с ошибками, то система напишет соответствующее сообщение. Посмотреть перечень ошибок в приложениях Oracle SQL*Plus или SQL WorkSheet можно с помощью команды **SHOW ERROR**.

Если триггер создан без ошибок, это не означает, что он работает правильно. Для того чтобы убедиться в этом, необходимо запустить команду - событие триггера и проверить правильность работы триггера.

Триггер запускается **автоматически** при наступлении события триггера. В зависимости от типа триггера он может запускаться до (before) или после (after) выполнения самой команды DML. Триггер уровня строки (for each row) выполняется один раз для каждой строки таблицы, которая затрагивается событием триггера. Триггер уровня предложения (for each statement) всегда выполняется один раз для каждой команды – события триггера, независимо от того, сколько строк она затрагивает (в том числе, ни одной).

Обратите внимание: корреляционные имена *:new* и *:old* позволяют обращаться к полям соответственно новой или старой записи и могут употребляться только в триггерах уровня строки. Возможность использования корреляционных имен в триггерах уровня строки зависит от команды (см. Табл. 10).

Таблица 10. Возможность использования корреляционных имен в триггерах уровня строки

Корреляционное имя	INSERT	UPDATE	DELETE
:new	+	+	
:old		+	+

Триггер выполняется в рамках той транзакции, к которой относится событие триггера, поэтому процедура триггера **не может содержать команды управления транзакциями (commit, rollback, save point) и команды DDL**.

Если одна команда инициирует выполнение более чем одного триггера, то, в зависимости от типов, они выполняются в таком порядке:

1. "Перед началом выполнения команды" (Before-statement trigger)
2. "Перед обработкой записи" (Before-row trigger)
3. "После обработки записи" (After-row trigger)
4. "После окончания выполнения команды" (After-statement trigger)

Если триггер должен проверять или изменять данные, которые требуется записать в базу данных, то этот триггер должен иметь тип BEFORE, а если он занимается выполнением каких-либо действий по результатам выполнения события триггера, то тип AFTER. В противном случае, если при выполнении команды – события триггера возникнет нарушение ограничений целостности, то откатится не только сама команда, но и действия, выполненные триггером, а это лишняя работа.

Примеры создания триггеров:

1. Проверка сложных ограничений целостности:

```
-- Зарплата сотрудника должна попадать в один из интервалов
-- значений зарплаты, которые установлены
-- для разных категорий сотрудников в таблице sal_grade.
create or replace trigger check_salary
   before INSERT or UPDATE of salary ON emp
   for each row
   when :new.salary>=4500
declare
   cnt number(3);
begin
       select count(*) into cnt from sal_grade
          where :new.salary between low_value and high_value;
       -- :new. – обращение к полю новой записи (добавляемой или изменяемой).
       if cnt = 0 then
          raise_application_error(-20050, 'Зарплата не попадает в допустимый интервал');
       end if;
end;
/
```

2. Аудит действий пользователя:

```
-- Отслеживаем действия пользователей над таблицей TAB
-- и фиксируем данные о них в специальной таблице TAB_LOG
-- записывая туда имя пользователя, признак действия и дату.

create  or  replace trigger  audit_tab
  after  INSERT  or  UPDATE  or  DELETE  ON  tab
declare  ch  char:='U';
begin
  if  INSERTING  then  ch:= 'I';
  elsif  DELETING  then  ch:= 'D';
  end  if;
  insert  into  tab_log  values ( substr(user, 1, 30), ch, sysdate);
end;
/
```

Здесь:
INSERTING, DELETING и **UPDATING** – условные предикаты, позволяющие определить, какая операция явилась событием триггера.
substr, user, sysdate – функции, описание которых можно посмотреть в Приложении 1.

2. Автоматизация обработки данных (перенос данных в архив при удалении их из основной таблицы):

```
-- Структура архивной таблицы:
create  table  emp_post(
        id      number(6)  not null,
        post   varchar2(40) not null,
        sdate date  not null);
```

```
-- Автоматическое добавление данных о  предыдущей должности сотрудника
-- в архив  с указанием даты, когда произошел перевод на  другую должность.
create  or  replace  trigger  cross_emp
  before  UPDATE  of  post  ON  emp
  for  each  row
begin
  insert  into  emp_post
    values(:old.id, :old.post, trunc(sysdate));
end;
/
```

Здесь:
:old. – обращение к полю старой записи (удалённой или изменяемой);
trunc и **sysdate** – функции, описание которых есть в Приложении 1.

3. Автоматизация обработки данных (установка значений по умолчанию):

```
-- Если дата приема сотрудника на работу не указана,
-- то установить текущую дату.
-- Перевести ФИО сотрудника в верхний регистр.
create or replace trigger set_emp
  before INSERT or UPDATE ON emp
  for each row
declare ch char:='U';
begin
  if :new.date_get IS NULL then :new.date_get := trunc(sysdate);
  else :new.date_get := trunc(:new.date_get);
  end if;
  :new.name := upper(:new.name);
end;
/
```

Здесь:
:new. – обращение к полю новой записи (добавляемой или изменяемой);
trunc, upper и **sysdate** – функции, описание которых есть в Приложении 1.

2. ВЫПОЛНЕНИЕ ЛАБОРАТОРНЫХ РАБОТ

Выполнение лабораторных работ заключается в создании двух триггеров (работа №6), процедуры и функции (работа №7) для базы данных, работающей под управлением СУБД Oracle. В дисплейном классе установлена версия Oracle Express Edition, которую можно бесплатно скачать с официального сайта oracle.com. Эта версия имеет простой оконный интерфейс, аналогичный интерфейсу браузера.

Логин и пароль выдаются преподавателем. Перед началом работы вам необходимо с помощью SQL-скриптов, написанных для 1-й лабораторной работы, создать исходную базу данных в соответствии со своим вариантом. Можно воспользоваться закладкой "Object Browser", но скрипт проще запустить через окно "SQL". Текст программ вводится через окно "SQL" -> "SQL Commands".

Задания на лабораторные работы 6 и 7 опубликованы на сайте http://rema44.ru/resurs/students/karpova .

Библиографический список

1. Смирнов С. Н. Задворьев И.С. Работаем с Oracle: Учебное пособие, 2-е изд., испр. и доп. – М.: Гелиос АРВ, 2002. – 496 с.
2. Грабер М. Введение в SQL. – М.: Лори, 2008. – 378 с.
3. Введение в язык PL/SQL. – http://www.irgups.ru/web-edu/~eugine/old/old/files/oracle/asu/oradoc/plsql/toplsql.htm

Приложение 1.
Справочник по стандартным функциям СУБД Oracle

- Функция **USER** возвращает имя пользователя, запустившего текущую сессию.

Некоторые функции работы со строками:

- Функция **LENGTH**(*строка*) возвращает длину строки, заданной параметром *строка*.
- Функция **INSTR**(*строка_1*, *строка_поиска* [, *позиция_начала_поиска* [,*номер_вхождения*]]) возвращает позицию вхождения строки, задаваемой параметром *строка_поиска,* в строку, задаваемую параметром *строка_1*. Позиция начала поиска задается необязательным числовым параметром *позиция_начала_поиска,* а необязательный параметр *номер_вхождения* задает требуемый номер вхождения строки поиска в основную строку. Значения по умолчанию для необязательных параметров 1. При отсутствии требуемого параметра вхождения строки поиска в основную строку функция возвращает значение 0.
- Функция **SUBSTR**(*строка_1*, *позиция* [, *длина_подстроки*]) возвращает подстроку параметра *строка_1,* начиная с позиции, заданной параметром *позиция,* и длиной, заданной параметром *длина_подстроки.* Если параметр *длина_подстроки* не задан, то возвращается подстрока до конца строки, заданной параметром *строка_1*.
- Функция **INITCAP**(*строка*) преобразует каждую первую букву слов параметра *строка* в прописную, а все последующие — в строчные.
- Функция **LOWER**(*строка*) преобразует каждую букву параметра *строка* в строчную.
- Функция **UPPER**(*строка*) преобразует каждую букву параметра *строка* в прописную.
- Функция **LPAD**(*строка_1*, *число_символов* [, *символ_наполнитель*]) возвращает значение параметра *строка_1,* дополненное слева до числа символов, которое задано параметром *число_символов,* символом-наполнителем, заданным параметром *символ_наполнитель*. По умолчанию символом-наполнителем является пробел.
- Функция **RPAD**(*строка_1*, *число_символов* [, *символ_наполнитель*]) возвращает значение параметра *строки_1,* дополненное справа до числа символов, которое задано параметром *число_символов,* символом-наполнителем, заданным параметром *символ_наполнитель*. По умолчанию символом-наполнителем является пробел.
- Функция **LTRIM**(*строка_1* [, *строка_шаблон*]) возвращает усеченное слева значение параметра *строка_1*. Из строки параметра *строка_1* символы удаляются слева до тех пор, пока удаляемый символ входит в множество символов параметра *строка_шаблон*. По умолчанию *строка_шаблон* состоит из символа пробела.
- Функция **RTRIM**(*строка_1* [, *строка_шаблон*]) возвращает усеченное справа значение параметра *строка_1*. Из строки параметра *строка_1* символы удаляются справа до тех пор, пока удаляемый символ входит в множество символов параметра *строка_шаблон*. По умолчанию *строка_шаблон* состоит из символа пробела.

Некоторые функции работы с числами:

- Функция **TRUNC** (*числовой_аргумент* [, *позиция*]) усекает значение параметра *числовой _аргумент* с точностью, определяемой параметром *позиция*. Параметр *позиция* определяет число десятичных знаков после запятой. Если параметр *пози-*

ция отрицательный, то аргумент округляется до целых чисел соответствующего масштаба (для значения параметра -1 до десятков, от -2 до сотен и т.д.). Значение параметра *позиция* по умолчанию 0.

- Функция **ROUND**(*числовой_аргумент* [*, позиция*]) округляет значение параметра *числовой_аргумент* с точностью, определяемой параметром *позиция.* Параметр *позиция* определяет число десятичных знаков после запятой. Если параметр *позиция* отрицательный, то аргумент округляется до целых чисел соответствующего масштаба (для значения параметра -1 до десятков, -2 до сотен и т.д.). Значение параметра *позиция* по умолчанию 0.
- Функция **MOD**(*числовой_аргумент, основание*) возвращает остаток от деления параметра *числовой_аргумент* на значение, определяемое параметром *основание.* Использование отрицательных значений параметра *основание* не рекомендуется, поскольку результат не соответствует принятому определению модуля числа.

Некоторые функции замены аргументов:

- Функция **NVL**(*аргумент_1, аргумент_2*) возвращает *аргумент_2*, если *аргумент_1* имеет неопределенное значение (NULL), в противном случае возвращается *аргумент_1*. Тип данных возвращаемого значения определяется типом данных параметра *аргумент_1*.
- Функция **DECODE**(*выражение, аргумент_1, результат_1*[*, аргумент_2, результат_2, ...*[*значение_по_умолчанию*]]) возвращает значение параметра *результат_x*, если параметр *выражение* совпадает с параметром *аргумент_x*, где x принимает значение 1,2,… Если совпадения нет, то возвращается *значение_по_умолчанию*. Если этот параметр не задан, то возвращается неопределенное значение (NULL).

Некоторые функции работы с датами:

- Функция **SYSDATE** возвращает дату и время, определяемые средствами операционной системы локального приложения.
- Функция **ROUND**(*дата* [*,формат*]) округляет значение параметра *дата* по шаблону, определяемому параметром *формат.* Если параметр *формат* опущен, то аргумент *дата* округляется до дней (время в начале дня устанавливается в полночь).
- Функция **TRUNC**(*дата* [*,формат*]) усекает значение параметра *дата* по шаблону, определяемому параметром *формат.* Если параметр *формат* опущен, то аргумент *дата* усекается до ближайшего дня (время в начале дня устанавливается в полночь).
- Функция **MONTHS_BETWEEN**(*дата1, дата2*) возвращает количество месяцев между двумя датами дата1 и дата2 с учетом знака как (дата1–дата2), возвращаемое число является дробным. Если числа (дни) совпадают (например, обе даты относятся к 10-му числу разных месяцев), то полученный результат является целым числом.

Некоторые функции преобразования типов данных:

- Функция **TO_CHAR**(*числовой_аргумент* [*,формат*]) возвращает результат преобразования значения параметра *числовой _аргумент* типа NUMBER в строку типа VARCHAR2. Если параметр *формат* опущен, *числовой_аргумент* преобразовывается в строку с длиной, достаточной для хранения всех значащих цифр. Некоторые значения параметра *формат* для преобразования числовых значений представлены в табл. 11.

Таблица 11. Некоторые значения параметра числовой ***формат***

Параметр формат	Тип выводимого результата
9	Выводится цифра. Лидирующий 0 заменяется пробелом.
0	Выводится цифра. Лидирующий 0 выводится.
EEEE	Результат выводится в экспоненциальной нотации.
G	Выводится символ-разделитель (обычно запятая).

- Функция **TO_NUMBER**(*символьный_аргумент*) возвращает результат преобразования значения параметра *символьный_аргумент* символьного типа в аргумент типа NUMBER. Параметр *символъный_аргумент* может представлять числа в любой допустимой Oracle нотации.
- Функция **TO_DATE**(*символьный_аргумент* [*формат*]) возвращает результат преобразования значения параметра *символьный_аргумент* символьного типа в тип DATE. Если параметр *формат* опущен, *символьный_аргумент* должен соответствовать формату даты, принятой в системе по умолчанию. Наиболее употребительные значения параметра *формат* представлены выше в табл. 12. Обычно функцию TO_DATE используют для преобразования даты из используемого некоторым приложением формата в стандартный формат системы.

Таблица 12. Значения параметра ***формат*** для даты

Модель формата	Описание
CC,SCC	век (S префиксует даты до н.э. минусом)
YYYY,SYYYY	год (S префиксует даты до н.э. минусом)
IYYY	год в стандарте ISO
YYY,YY,Y	последние три, две или одна цифра года
IYY,IY,I	то же для года ISO
Y,YYY	год с запятой
YEAR,SYEAR	год прописью (S префиксует даты до н.э. минусом)
RR	последние две цифры года в новом веке
BC,AD	индикатор BC или AD
B.C.,A.D.	индикатор B.C. или A.D.
Q	квартал (1-4)
MM	месяц (1-12)
RM	римский номер месяца (I-XII)
MONTH	имя месяца
MON	сокращенное имя месяца
WW	неделя года (1-53)
IWW	неделя года (1-52 или 1-53) по ISO
W	неделя месяца (1-5)
DDD	день года (1-366)
DD	день месяца (1-31)
D	день недели (1-7)
DAY	имя дня
DY	сокращенное имя дня
J	юлианский день (число дней с 1 января 4712 г. до н.э.)
AM,PM	индикатор полудня
A.M.,P.M.	индикатор полудня с точками
HH,HH12	час дня (1-12)
HH24	час суток (0-23)
MI	минута (0-59)
SS	секунда (0-59)
SSSSS	секунд после полуночи (0-86399)