ОБЗОР PL/SQL

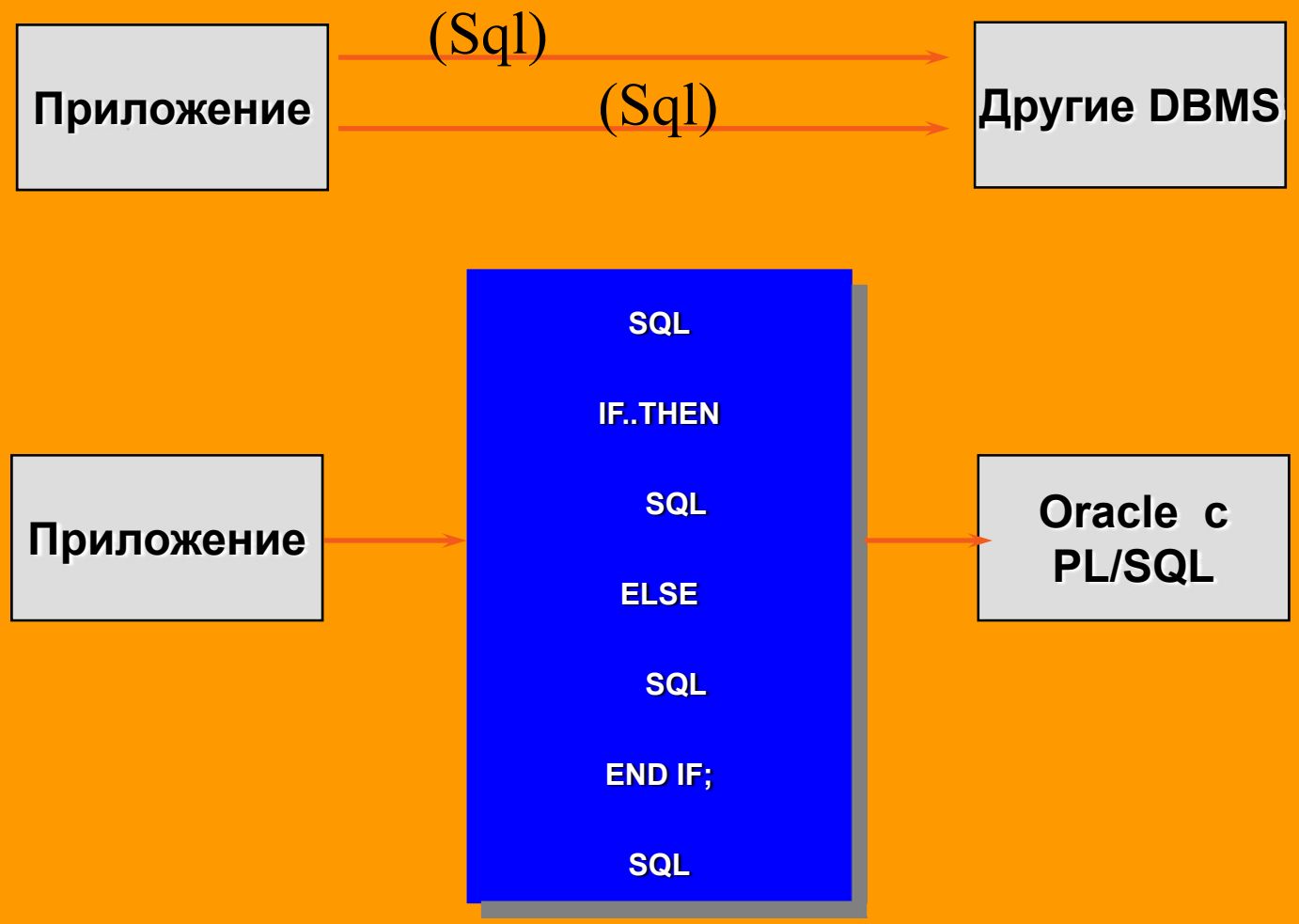
Общие понятия

PL/SQL – это дополнение стандартного языка SQL многими средствами, присущими современным языкам программирования.

PL/SQL обладает многими преимуществами, недоступными в SQL:

- Модульная разработка программ.
- Объявление переменных.
- Управляющие структуры.
- Обработка ошибок.
- Переносимость.
- Интеграция.
- Повышенная производительность

Повышение производительности PL/SQL



Блочная структура

DECLARE – необязательно

- Переменные, константы, курсоры, исключения пользователя.

BEGIN – обязательно

- операторы SQL.
- операторы PL/SQL.

EXCEPTION – необязательно

- Действия, выполняемые при возникновении ошибки.

END; – необязательно

Пример блока PL/SQL

```
DECLARE
  v_product_id      s_product.id%TYPE;
BEGIN
  SELECT id
   INTO v_product_id
   FROM s_product
   WHERE id = &p_product_id;
  DELETE FROM s_inventory
   WHERE product_id = v_product_id;
  COMMIT;
EXCEPTION
  WHEN OTHERS THEN
   ROLLBACK;
   INSERT INTO exception_table (message)
       VALUES ('Some error occurred in the
   database.');
   COMMIT;
END;
```

Типы блоков PL/SQL

анонимный

```
[DECLARE]


BEGIN
  --statements

[EXCEPTION]

END;
```

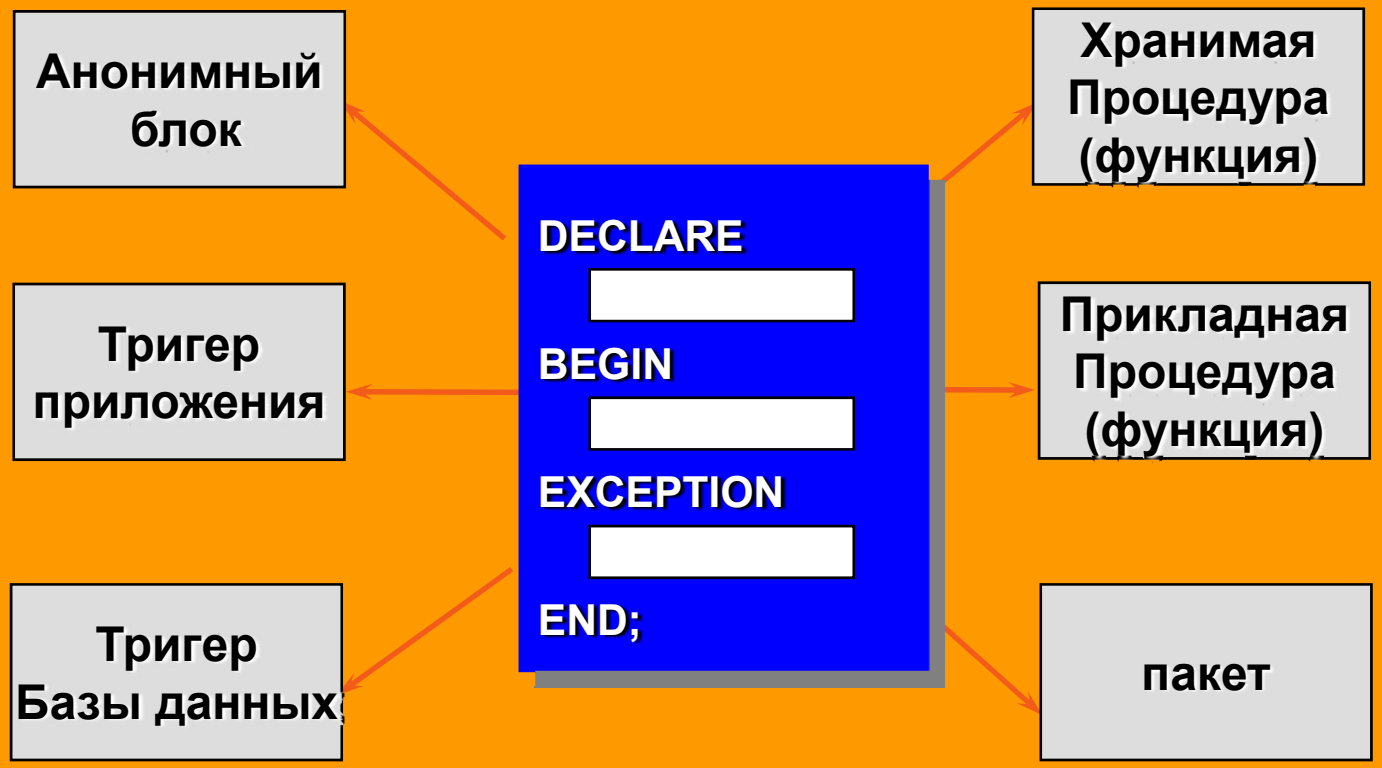
процедура

```
PROCEDURE name
IS

BEGIN
  --statements

[EXCEPTION]

END;
```

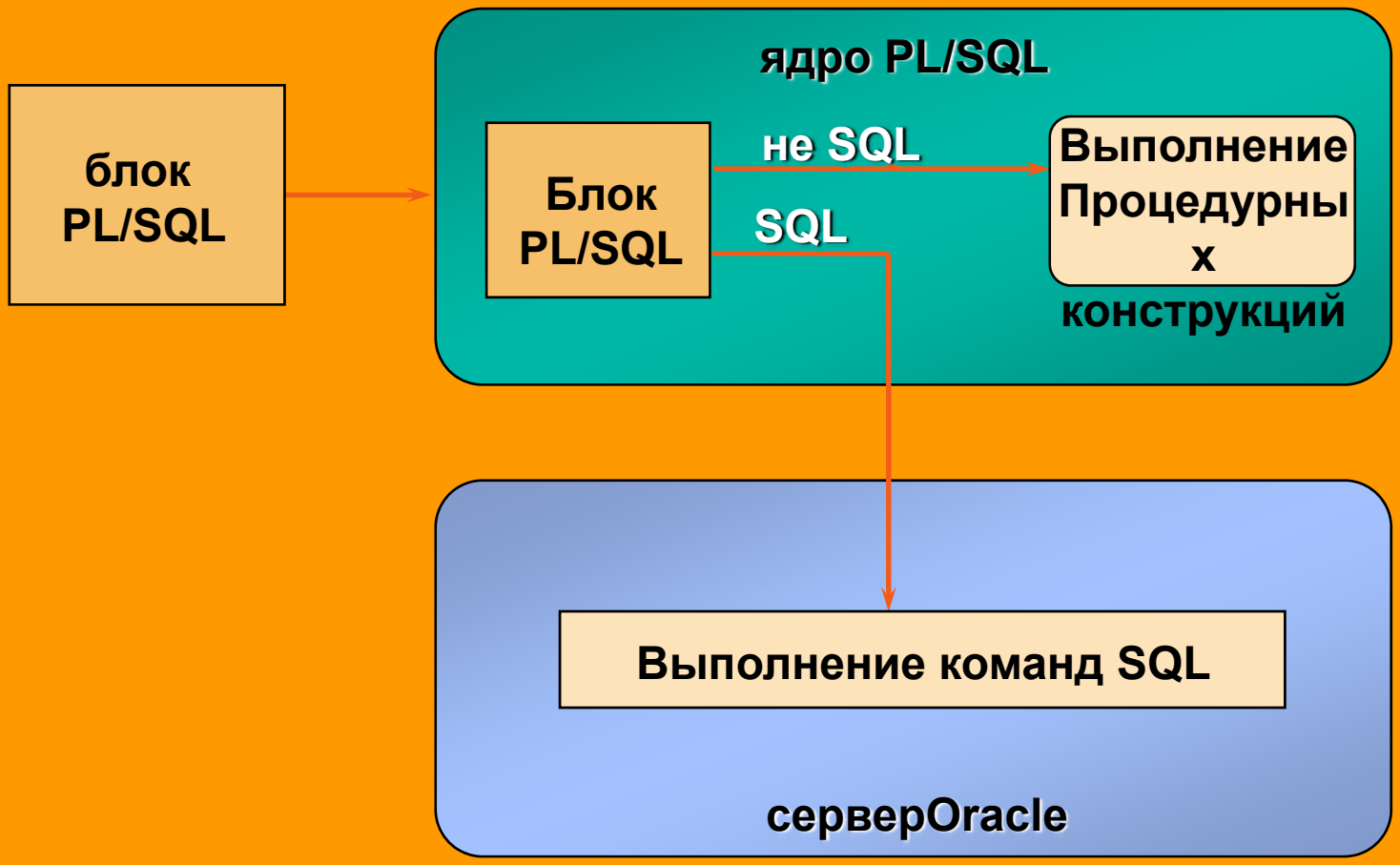
функция

```
FUNCTION name
RETURN datatype
IS
BEGIN
  --statements
  RETURN value;
[EXCEPTION]

END;
```

Программные конструкции PL/SQL



Среда PL/SQL


блок PL/SQL
ядро PL/SQL
Блок PL/SQL
не SQL
SQL
Выполнение Процедурных конструкций
Выполнение команд SQL
серверOracle

Продукт Procedure Builder

- Графическая среда для разработки кода PL/SQL
- Встроенные редакторы
- Компиляция, тестирование и отладка кода
- Разбиение приложения на компоненты позволяет перемещать программные единицы мышью между клиентом и сервером

Общие сведения о Procedure Builder

компоненты	описание
Object Navigator (навигатор объектов)	управление конструкциями PL/SQL. Действия по отладке.
PL/SQL Interpreter (Интерпретатор PL/SQL)	Отладка кода PL/SQL. Анализ кода PL/SQL в реальном времени.
Program Unit Editor (Редактор программных единиц)	Создание и редактирование исходного кода PL/SQL.
Stored Program Unit Editor (Редактор храни-Мых программных единиц)	Создание и редактирование исходного кода PL/SQL на стороне сервера.
Database Trigger Editor (Редактор тригеров Баз данных)	Создание и редактирование тригеров баз данных.

РАЗРАБОТКА ПРОСТОГО БЛОКА

Структура простого блока

```
DECLARE

BEGIN

EXCEPTION

END;
```

Объявление скалярных переменных

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

- Не имеют компонент
- Хранят единственное значение
- Основные типы:
  - BINARY_INTEGER
  - NUMBER [(*m*,*n*)]
  - CHAR [(*m*)]
  - LONG
  - LONG RAW
  - VARCHAR2(*m*)
  - DATE
  - BOOLEAN

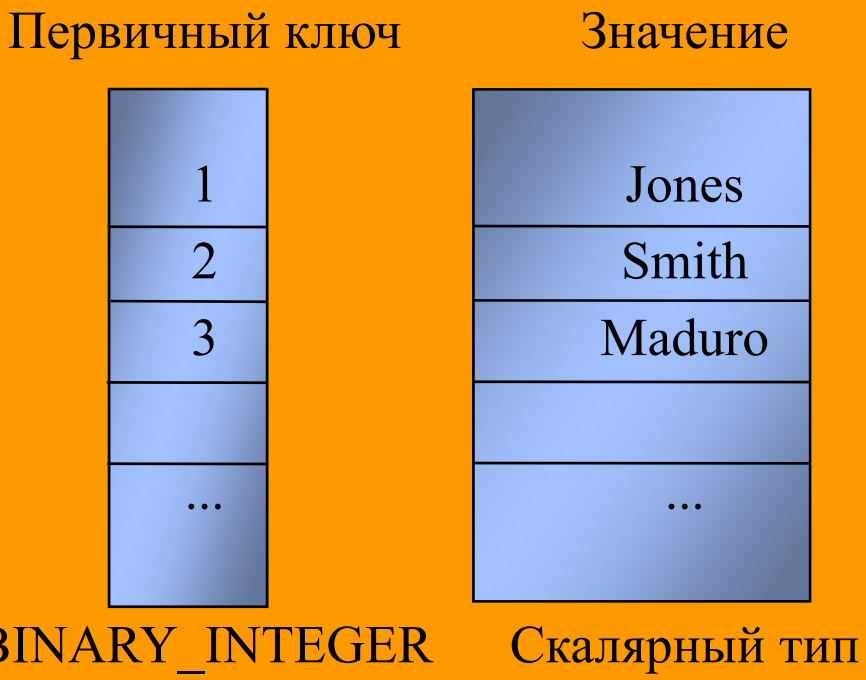
Объявление скалярных переменных

```
v_gender        CHAR(1);
v_count         BINARY_INTEGER := 0;
v_total_sal     NUMBER(9,2) := 0;
v_order_date    DATE := SYSDATE + 7;
c_tax_rate      CONSTANT NUMBER(3,2) := 8.25;
v_valid         BOOLEAN NOT NULL := TRUE;
```

```
...
  v_last_name           s_emp.last_name%TYPE;
  v_first_name          s_emp.first_name%TYPE;
  v_balance             NUMBER(7,2);
  v_minimum_balance     v_balance%TYPE := 10;
...
```

Составные типы данных

Таблицы PL/SQL



Записи

Создание таблицы PL/SQL

```
  TYPE type_name IS TABLE OF datatype
    [NOT NULL] INDEX BY BINARY_INTEGER;
  identifier    type_name;
```

```
...
  TYPE name_table_type IS TABLE OF
VARCHAR2(25)
     INDEX BY BINARY_INTEGER;
  first_name_table name_table_type;
  last_name_table name_table_type;
...
```

Создание записи

```
  TYPE type_name IS RECORD
   (field_name1 field_type
   [NOT NULL {:=|DEFAULT} expr],
    field_name2 field_type
   [NOT NULL {:=|DEFAULT} expr],...);
  identifier       type_name;
```

```
...
  TYPE emp_record_type IS RECORD
    (last_name    VARCHAR2(25),
     first_name   VARCHAR2(25),
     gender       CHAR(1));
  employee_recordemp_record_type;
...
```

```
...
  dept_record      s_dept%ROWTYPE;
  emp_record       s_emp%ROWTYPE;
...
```

Область видимости переменных

```
...
  x  BINARY_INTEGER;
BEGIN
  ...
  DECLARE
    y  NUMBER;
  BEGIN
    ...
  END;
  ...
END;
```

Область видимости x

Область видимости y

Присвоение значений переменным

```
identifier := expr;
plsql_table_name (primary_key_value) := expr;
plsql_record_name.field_name := expr;
```

```
v_max_sal     := v_sal;
last_name_table (3) := 'Maduro';
emp_record.last_name := 'Maduro';
emp_record.first_name  := 'Elena';
emp_record.gender := 'F';
```

Присвоение значений переменным

```
DECLARE
  v_weight                NUMBER(3) := 600;
  v_message               VARCHAR2(255) := 'Product 10012';
BEGIN
                          Sub-Block
    DECLARE
      v_weight    NUMBER(3) := 1;
      v_message   VARCHAR2(255) := 'Product 11001';
      v_new_locn  VARCHAR2(50) := 'Europe';

    BEGIN

     v_weight    := v_weight + 1;
     v_new_locn  := 'Western '|| v_new_locn;
    END;

     v_weight    := v_weight + 1;
     v_message   := v_message || ' is in stock';
     v_new_locn  := 'Western '|| v_new_locn;
END;
```

Операторы в PL/SQL

- Логические
- Арифметические
- Конкатенации
- Возведения в степень (**)
- Скобки

} Как в SQL

Как в SQL

```
v_count   := v_count + 1;
```

```
v_equal      := (v_n1 = v_n2);
```

```
v_valid      := (v_emp_id IS NOT NULL);
```

Функции в PL/SQL

- Доступны:
  - Числовые
  - Символьные
  - Преобразования типов
  - Работы с датами
- Недоступны:
  - GREATEST
  - LEAST
  - Групповые

} Как в SQL

```
v_mailing_address := v_name||CHR(10)||
   v_address||CHR(10)||v_country||CHR(10)||
   v_zip_code;
```

```
v_last_name  := UPPER (v_last_name);
```

Практическое занятие

1. Какие из приведенных описаний неверны и почему?

```
Declare
V_id      Number (7);
```

```
Declare
V_x, v_y, v_z      VARCHAR (7);
```

```
Declare
V_birtdate          date not null;
```

```
Declare
V_in_stock          boolean := 1;
```

```
Declare
Emp_record          emp_record_type;
```

Практическое занятие

2. Создайте процедуру, принимающую два числа через переменные. Первое нужно разделить на второе и к результату прибавить второе число. Результат должен быть записан в переменную PL/SQL и выведен на экран.

3. Создайте функцию, вычисляющую общее вознаграждение за год. Функции должны передаваться годовая зарплата и процент премиальных. Премиальные необходимо преобразовать из целого числа в десятичное. Если зарплата не определена, то функция должна выдавать значение «ноль». Если премия не определена, функция должна выдавать только зарплату.

МОДУЛЬНОЕ ПРОГРАММИРОВАНИЕ

Программная единица PL/SQL

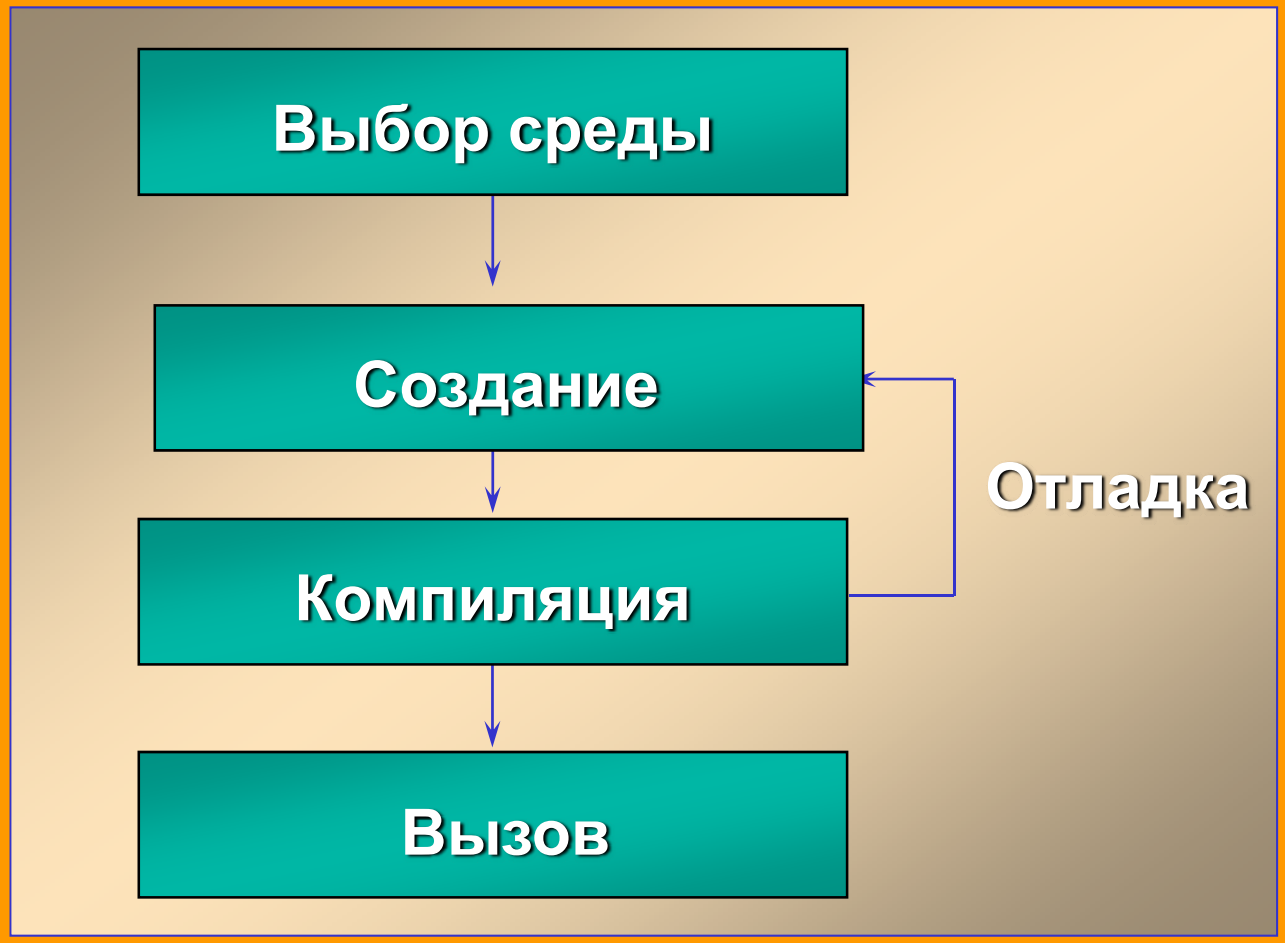
Именованные блоки

Три основных категории

- Процедура
- Функция
- Пакет

Хранятся в базе данных или обрабатываются в прикладных программах

Создание подпрограммы



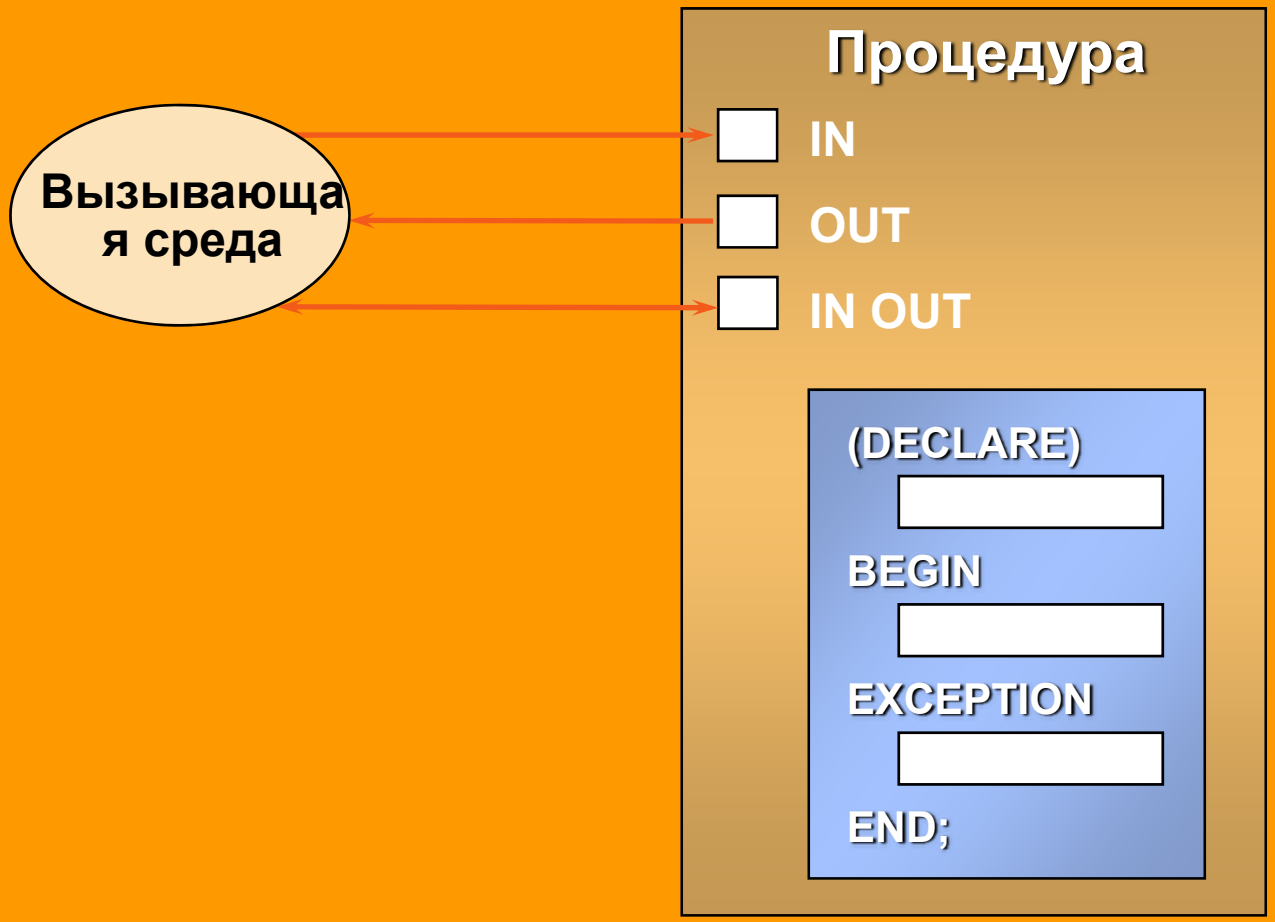
Создание процедуры

```
PROCEDURE name
    [(parameter,...)]
IS
pl/sql_block;
```

```
parameter_name [IN | OUT | IN OUT] datatype
     [{:= | DEFAULT} expr]
```

При создании из SQL*PLUS необходимо пользоваться командой **CREATE OR REPLACE**

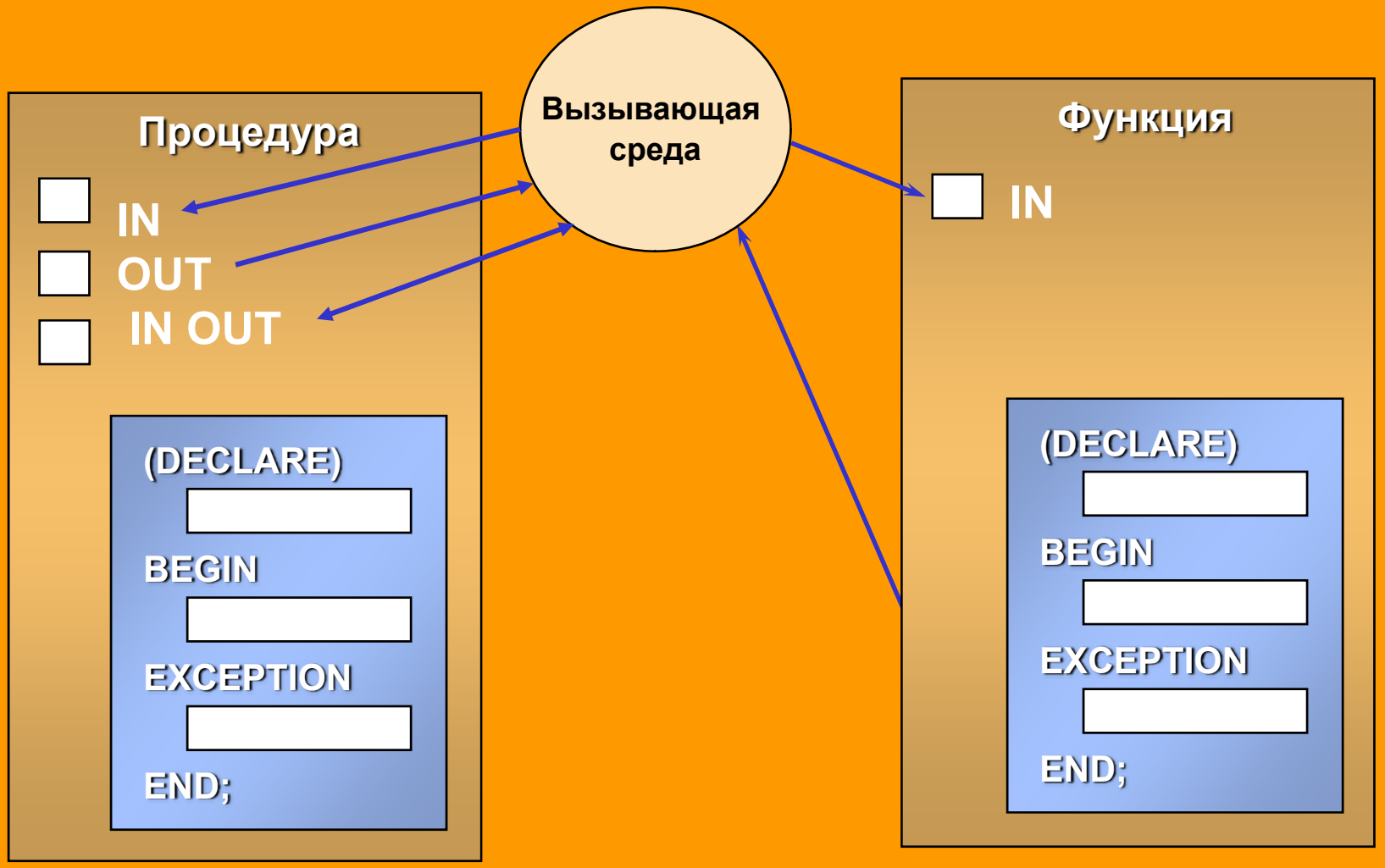
Параметры процедуры



Пример создания процедуры

```
PROCEDURE change_salary
     (v_emp_id IN NUMBER,
      v_new_salary IN NUMBER)
IS
BEGIN
     UPDATE   s_emp
      SET     salary = v_new_salary
      WHERE   id = v_emp_id;
     COMMIT;
END change_salary;
```

Процедура или функция



Создание функции

```
FUNCTION name
      [(parameter,...)]
      RETURN datatype
IS
pl/sql_block;
```

```
FUNCTION tax
      (v_value IN NUMBER)
      RETURN NUMBER
IS
BEGIN
      RETURN (v_value * .07);
END tax;
```

Вызов блока

```
PL/SQL> change_salary (17, 1000);
```

```
PROCEDURE process_sal
     (v_emp_id IN NUMBER,
      v_new_salary IN NUMBER)
IS
BEGIN
      change_salary (v_emp_id, v_new_salary);
      --invoking procedure change_salary
      ...
END;
```

```
PL/SQL> CREATE NUMBER x PRECISION 4
PL/SQL> :x := tax(100);
PL/SQL> TEXT_IO.PUT_LINE (TO_CHAR(:x));
```

Практическое занятие

1. Процедура может содержать параметры IN, OUT и IN **OUT. (**Да/Нет)
2. Процедуру можно использовать в команде SQL. (Да/Нет)
3. По умолчанию используются параметры типа IN **OUT. (**Да/Нет)
4. Функции выполняются как часть выражения. (Да/Нет)
5. Назовите четыре части синтаксиса подпрограммы.
6. Создайте процедуру MY_PROCEDURE для вывода на экран фразы "My procedure works" ("Моя процедура работает").
7. Скомпилируйте код. Для успешной компиляции внесите в код
8. Выполните процедуру из командной строки Интерпретатора

Пакеты PL/SQL

Понятие пакета PL/SQL

ПАКЕТ - это объект базы данных, который группирует логически связанные типы, программные объекты и подпрограммы PL/SQL.

Пакеты обычно состоят из двух частей, спецификации и тела, хотя иногда в теле нет необходимости.

1. **СПЕЦИФИКАЦИЯ пакета** – это интерфейс с вашими приложениями; она объявляет типы, переменные, константы, исключения, курсоры и подпрограммы, доступные для использования в пакете.
2. **ТЕЛО пакета** полностью определяет курсоры и подпрограммы, тем самым реализуя спецификацию пакета.

Формат пакета

В отличие от подпрограмм, пакеты нельзя вызывать, передавать им параметры или вкладывать их друг в друга. В остальном формат пакета аналогичен формату подпрограммы:

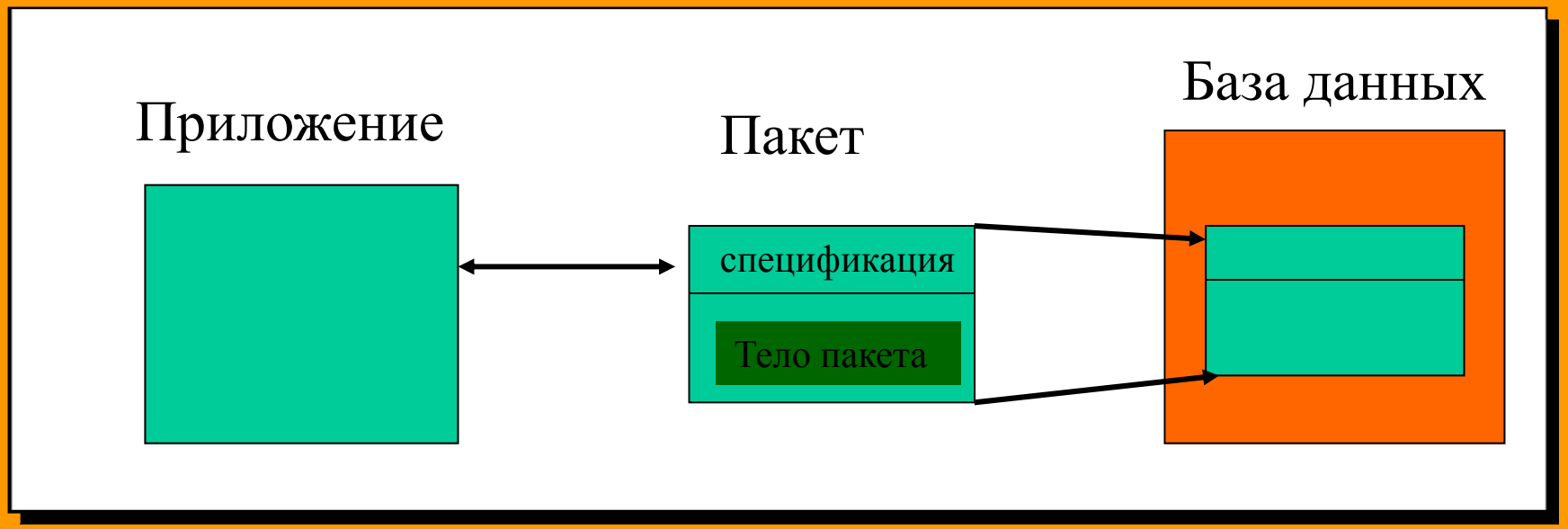
```
PACKAGE имя IS  -- спецификация (видимая часть)
   -- объявления общих типов и объектов
   -- спецификации подпрограмм
END [имя];
```

```
PACKAGE BODY имя IS  -- тело (скрытая часть)
   -- объявления личных типов и объектов
   -- тела подпрограмм
[BEGIN
   -- предложения инициализации]
END [имя];
```

Интерфейс пакета

1. **Спецификация** содержит ОБЩИЕ объявления, которые видимы вашему приложению.
2. **Тело** содержит детали реализации и ЛИЧНЫЕ объявления, которые скрыты от вашего приложения.

Можно отлаживать, развивать или заменять тело пакета, не изменяя интерфейса к этому телу (т.е. спецификации пакета).



Синтаксис создания пакета

Пакеты создаются интерактивно в **SQL*Plus** или **SQL*DBA** с помощью команд **CREATE PACKAGE и CREATE PACKAGE BODY.**

ПРИМЕР: В пакет формируем тип записи, курсор и две процедуры управления кадрами:

```
CREATE PACKAGE emp_actions AS  -- спецификация
        TYPE EmpRecTyp IS RECORD (emp_id INTEGER, salary REAL);
        CURSOR desc_salary (emp_id NUMBER) RETURN EmpRecTyp;

        PROCEDURE hire_employee
           (ename   CHAR,
            job     CHAR,
            mgr     NUMBER,
            sal     NUMBER,
            comm    NUMBER,
            deptno  NUMBER);

        PROCEDURE fire_employee (emp_id NUMBER);
    END emp_actions;
```

Синтаксис создания пакета

```
CREATE PACKAGE BODY emp_actions AS  -- тело
        CURSOR desc_salary (emp_id NUMBER) RETURN EmpRecTyp IS
           SELECT empno, sal FROM emp ORDER BY sal DESC;

        PROCEDURE hire_employee
           (ename   CHAR,
            job     CHAR,
            mgr     NUMBER,
            sal     NUMBER,
            comm    NUMBER,
            deptno  NUMBER) IS
        BEGIN
           INSERT INTO emp VALUES (empno_seq.NEXTVAL, ename, job,
              mgr, SYSDATE, sal, comm, deptno);
        END hire_employee;

        PROCEDURE fire_employee (emp_id NUMBER) IS
        BEGIN
           DELETE FROM emp WHERE empno = emp_id;
        END fire_employee;
     END emp_actions;
```

Преимущество использования пакетов

- Модульность
- Облегчение
- Проектирования приложений
- Скрытие информации
- Расширенная функциональность
- Лучшая производительность

Видимыми и доступными для приложений являются лишь объявления в спецификации пакета. Детали реализации в теле пакета скрыты и недоступны. Поэтому вы можете исправлять тело (реализацию), не перекомпилируя вызывающих программ.

Спецификация пакета

Если спецификация пакета объявляет лишь типы, константы, переменные и исключения, тело пакета не нужно.

Пример пакета, состоящего только из спецификации:

```
PACKAGE trans_data IS
        TYPE TimeTyp IS RECORD
           (minute  SMALLINT,
            hour    SMALLINT);
        TYPE TransTyp IS RECORD
           (category  VARCHAR2,
            account   INTEGER,
            amount    REAL,
            time      TimeTyp);
        minimum_balance     CONSTANT REAL := 10.00;
        number_processed    INTEGER;
        insufficient_funds  EXCEPTION;
     END trans_data;
```

Спецификация пакета

Пакет **trans_data** не нуждается в теле, потому что типы, константы, переменные и исключения не требуют реализации.

Такие пакеты позволяют определять глобальные переменные – для использования подпрограммами и триггерами – которые существуют на протяжении всей сессии.

Обращение к содержимому пакета

Для обращения к типам, объектам и подпрограммам, объявленным в спецификации пакета, используются квалифицированные ссылки:

имя_пакета.имя_типа
имя_пакета.имя_объекта
имя_пакета.имя_подпрограммы

Вы можете обращаться к содержимому пакета из триггеров базы данных, хранимых подпрограмм, строенных блоков PL/SQL, а также анонимных блоков PL/SQL, посылаемых в ORACLE интерактивно через SQL*Plus или SQL*DBA.

Обращение к содержимому пакета

Пример обращения к пакетированной переменной **miminum_balance**, которая объявлена в пакете **trans_data**:

```
DECLARE
        new_balance  REAL;
        ...
    BEGIN
        ...
        IF new_balance <
trans_data.minimum_balance THEN
            ...
        END IF;
        ...
```

Тело пакета

Тело пакета реализует спецификацию пакета. Оно содержит определения всех курсоров и подпрограмм, объявленных в спецификации пакета.

При этом любая подпрограмма, определенная в теле пакета, доступна извне пакета лишь в том случае, если ее спецификация также появляется в спецификации пакета.

Тело пакета может также содержать личные объявления, которые определяют типы и объекты, необходимые для внутренней работы пакета. Сфера таких объявлений локальна в теле пакета. Поэтому объявленные здесь типы и объекты недоступны нигде, кроме тела пакета.

В отличие от спецификации пакета, декларативная часть тела пакета может содержать тела подпрограмм.

Тело пакета

За декларативной частью тела пакета может следовать необязательная часть инициализации, которая обычно содержит предложения, инициализирующие некоторые из переменных, ранее объявленных в пакете.

Часть инициализации пакета не играет большой роли, потому что, в отличие от подпрограмм, пакет нельзя вызывать или передавать ему параметры. Следовательно, часть инициализации пакета отрабатывает лишь один раз, при первом обращении к пакету.

Взаимодействие с Oracle

Команды SQL в PL/SQL

1. Извлечение строк данных из базы данных производится командой SELECT.
2. Изменение строк базы данных производится командами DML.
3. Управление транзакциями осуществляется командами COMMIT и ROLLBACK.
4. Пакет DBMS_SQL позволяет выполнять команды DML И DCL.

Выборка данных: синтаксис

Для выборки данных из базы данных используется команда **SELECT**

```
SELECT список_выборки
INTO имя_переменной| имя_записи
FROM таблица
WHERE условие;
```

- Предложение INTO обязательно.
- Должна быть возвращена только одна строка.
- Доступен полный синтаксис SELECT.

Выборка данных: пример

Выборка даты размещения заказа и даты отгрузки

```
PROCEDURE ship_date
     (v_ord_id IN NUMBER)
IS
     v_date_ordered s_ord.date_ordered%TYPE;
     v_date_shipped s_ord.date_shipped%TYPE;
BEGIN
     SELECT date_ordered, date_shipped
     INTO v_date_ordered, v_date_shipped
     FROM s_ord
     WHERE id=v_ord_id;
     ...
END ship_date;
```

Выборка данных: пример

Вывод суммы заработной платы всех сотрудников указанного отдела.

```
FUNCTION sum_emp
     (v_dept_id IN NUMBER)
  RETURN NUMBER
IS
     v_sum_salary s_emp.salary%TYPE;
BEGIN
     SELECT SUM(salary) - групповая функция
     INTO v_sum_salary
     FROM s_emp
     WHERE dept_id=v_dept_id;
  RETURN (v_sum_salary);
END sum_emp;
```

Выборка данных: пример

Выборка всей информации об указанном отделе

```
PROCEDURE all_dept
     (v_dept_id IN NUMBER)
IS
     dept_record s_dept%ROWTYPE;
BEGIN
     SELECT *
     INTO dep_record --запись PL/SQL
     FROM s_dept
     WHERE id=v_dept_id;
     ...
END all_dept;
```

Исключения для команды SELECT

- Предложение SELECT в PL/SQL должны возвращать только одну строку.
- Возврат нулевого количества строк или нескольких строк рассматривается как исключение.
- Исключения для команды SELECT:
  - TOO_MANY_ROWS (слишком много строк)
  - NO_DATA_FOUND (данные не обнаружены)

Манипулирование данными

Изменения в базу данных вносятся с помощью команд DML.

- INSERT
- UPDATE
- DELETE

Команды COMMIT и ROLLBACk

- Транзакция начинается с первой команды DML, следующей за COMMIT или ROLLBACK.
- Транзакция завершается явно командой COMMIT или ROLLBACK.

Управление транзакциями: пример

```
BEGIN
     INSERT INTO temp(num_col1, num_col2, char_col)
        VALUES (1,1,'ROW 1');
     SAVEPOINT a;
     INSERT INTO temp(num_col, num_col2, char_col)
        VALUES (2,2,'ROW 2');
     SAVEPOINT b;
     INSERT INTO temp(num_col, num_col2, char_col)
        VALUES (3,3,'ROW 3');
     SAVEPOINT c;
ROLLBACK TO SAVEPOINT b;
END;
```

Практическое занятие

1. Создайте процедуру для включения нового отдела в таблицу S_DEPT.
2. Создайте процедуре, обновляющую номер региона для существующего отдела.
3. Создайте процедуру для удаления отдела, созданного в упражнении 1.
4. Создайте процедуру NEW_EMP для включения записи о новом служащем в таблицу S_EMP.

УПРАВЛЕНИЕ ПОТОКОМ В БЛОКАХ PL/SQL

Управление потоком операций в PL/SQL

Логический поток операций можно изменять с помощью управляющих структур:

- Структуры условного управления (оператор IF)
- Циклы
  - простой цикл
  - цикл FOR
  - цикл WHILE
  - оператор EXIT

Оператор IF

Действия можно выполнять выборочнов зависимости от определенных условий:

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

- ELSIF – одно слово
- END IF – два слова
- Предложение ELSE может быть только одно

Оператор IF. Примеры

```
...
IF V_last_name ='Dumas' THEN
     v_job:='Sales Representative';
     v_region_id:=35;
END IF;
...
```

```
...
IF v_date_shipped - v_date_ordered <5 THEN
     v_ship_flag:='Acceptable';
ELSE
     v_ship_flag:='Unacceptable';
END IF;
...
```

Оператор IF-THEN-ELSIF: пример

Вычисление одного значения на основе другого.

```
...
IF v_start > 100 THEN
     RETURN (2 * v_start);
ELSIF v_start>=50 THEN
     RETURN (.5 * v_start);
ELSE
     RETURN (.0 * v_start);
END IF;
...
```

Задание логических условий

- Неопределенные значения можно обрабатывать с помощью оператора IS NULL
- Выражение, содержащее неопределенные значения, дает результат NULL
- Результатом конкатенации выражения с неопределенным значением является пустая строка

Таблицы истинности

Построение простого логического условия с оператором сравнения

AND	TRUE	FALSE	NULL
TRUE	TRUE	FALSE	NULL
FALSE	FALSE	FALSE	FALSE
NULL	NULL	FALSE	NULL

NOT	
TRUE	FALSE
FALSE	TRUE
NULL	NULL

OR	TRUE	FALSE	NULL
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	NULL
NULL	TRUE	NULL	NULL

Логические условия

Каково значение v_falg в каждом случае?

```
V_flag:=v_reorder_flag AND v_anailable_flag
```

V_reorder_flag	V_anailable_flag	VALUE
TRUE TRUE NULL NULL	TRUE FALSE TRYE FALSE	

Циклы

• Цикл позволяет выполнить последовательность предложений несколько раз

• Три типа циклов:

- простой цикл

- цикл FOR

- цикл WHILE

Простой цикл: синтаксис

Многократное выполнение команд с помощью простого цикла

```
LOOP                              --ограничитель
    statement1;                   -- оператор
    …
    EXIT [WHEN condition]; --оператор EXIT
END LOOP;                         -- ограничитель
```

Без предложения EXIT цикл был бы бесконечным

Простой цикл: пример

Включение первых десяти товарных позиций в заказ номер 101.

```
...
  v_ord_id s_item.ord_id%TYPE:=101;
  v_counter NUMBER(2):=1;
BEGIN
...
  LOOP
     INSERT INTO s_item (ord_id, item_id)
     VALUES (v_ord_id, v_counter);
  v_counter:= v_counter+1;
  EXIT WHEN v_counter>10;
  END LOOP;
...
```

Цикл FOR: синтаксис

Цикл FOR – это быстый способ проверки количества повтора цикла.

```
FOR индекс in [REVERSE]
    нижняя_граница..верхняя_граница LOOP
  предложение1;
  предложение2;
  ...
END LOOP;
```

Объявление индекса (переменной цикла) не требуется; индекс описывается неявно.

Цикл FOR: пример

Пример

- Вывод количества выполнений цикла и последнего значения индекса.

Указания

- Ссылаться на индекс можно только внутри цикла ; вне цикла он не определен.
- Для ссылки на текущее значение индекса можно использовать выражение.
- Нельзя ссылаться на индекс в качестве переменной в левой части оператора присваивания.

Цикл WHILE: синтаксис

Цикл WHILE используется для повторения цикла в течение всего времени, пока выполняется условие.

```
WHILE condition LOOP          -- Условие оценивается
  statement1;                 -- в начале каждой
  statement2;                 -- итерации
  ...
END LOOP;
```

Цикл WHILE: пример

Включение первых десяти товарных позиций в заказ номер 101.

```
...
  v_ord_id s_item.ord_id%TYPE:=101;
  v_counter NUMBER(2):=1;
BEGIN
...
  WHILE v_counter<= 10 LOOP
     INSERT INTO s_item (ord_id, item_id)
     VALUES (v_ord_id, v_counter);
  v_counter:= v_counter+1;
  END LOOP;
...
```

Вложенные циклы и метки

- Возможно несколько уровней вложенных циклов
- Для различения циклов и блоков используются метки
- Выход из внешнего цикла осуществляется с помощью оператора EXIT, содержащего ссылку на метку

Вложенные циклы и метки: пример

Выход из внешнего цикла по значениям, вычисленным во внутреннем блоке

```
...
  BEGIN
  <<Outer_loop>> LOOP
  v_counter:=v_counter+1;
  EXIT WHEN v_counter > 10;
  <<Inner_loop>> LOOP
      ...
      EXIT Outer_loop WHEN total_done = 'Yes';
                        -- Leave both loop
      EXIT WHEN inner_done = 'Yes';
                        -- Leave inner loop only
      ...
      END LOOP Inner_loop;
      ...
END LOOP Outer_loop;
END;
```

Практическое занятие

1. **Создайте процедуру SET_COMM, которая устанавливает процент комиссионных для служащего в зависимости от общего объема продаж**

а) Для подготовке к упражнению отмените действие ограничения для столбца COMISSION_PCT в таблице S_EMP. Проценты комиссионных в упражнении не будут удовлетворять ограничению.

```
 > ALTER TABLE s_emp
+> DROP CONSTRAINT s_emp_comission_pct_ck;
```

б) Создайте параметр, чтобы можно было ввести номер служащего.

в) Подсчитайте сумму всех заказов, размещенным этим служащим.

г) Если эта сумма меньше 10000 установите процент комиссионных «10».

д) Если сумма находится в диапазоне от 10000 до 1000000, установите процент комиссионных «15».

е) Если в таблице S_ORD нет заказов, размещенных данным служащим, установите процент комиссионных «0».

Практическое занятие

2. Создайте процедуру CUST_UPDATE с циклом для обработки кредитного рейтинга всех заказчиков по регионам. Не фиксируйте изменения

а) Если номер региона четный, установите кредитный рейтинг “Exellent”, даже если он уже был таким; в противном случае установите кредитный рейтинг “Good”.

б) Когда строки будут обновлены, определите количество обновленных строк. В зависимости от того, каким будет это количество, выдайте на экран следующую информацию:

- Если обновлено менее трех срок, сообщение : “Fewer then 3 customer records updated for region number X”, где X – номер региона.
- В противном случае – “Y rows updated for region number X”, где Y – количество обновленных строк, X – номер региона.

в) Отмените изменения. Задайте точку останова на проверке количества обновленных строк.

Практическое занятие

3. Создайте процедуру EMP_MESSAGE, которая выбирает фамилию служащего, дату начала работы и заработную плату по номеру служащего, задаваемому во времени выполнения. Выдайте на экран сообщение по любой комбинации нижеуказанных критериев (используйте вложенные операторы IF)

Критерий	**Сообщение**
Зарплата более 1200	Salary >1200
Фамилия содержит “R”	Name contains “R”
Дата начала работы приходится на март	March start date
Ничего из вышеуказанного	**None**

Обработка запросов с использованием курсоров

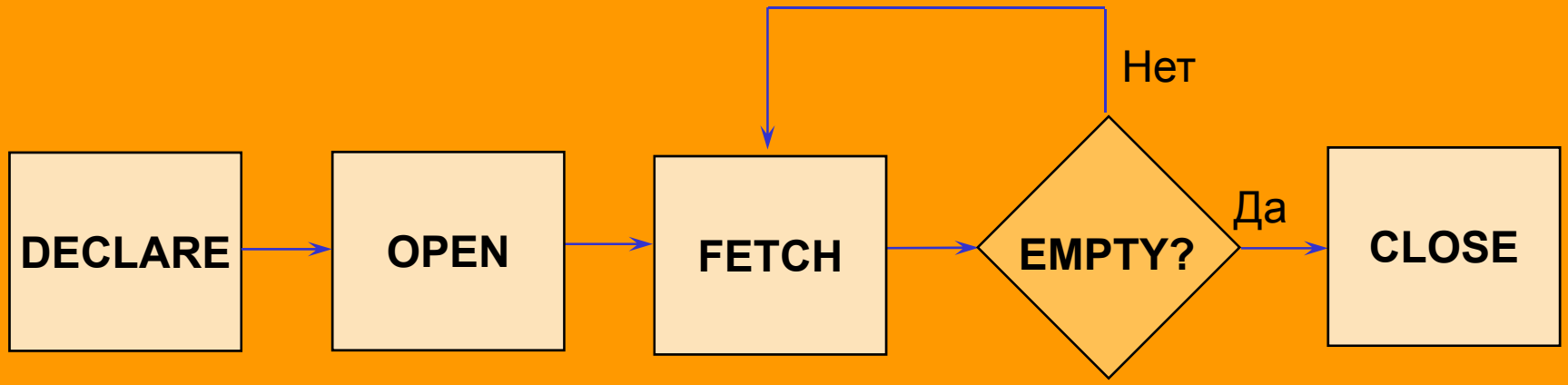
Что такое курсор?

- Каждая команда SQL, выполняемая на сервере Oracle, имеет свой курсор.
- Два типа курсоров:
  - Неявные курсоры: создаются для всех команд DML и команд SELECT PL/SQL.
  - Явные курсоры: создаются программистом. Имя присваивает программист.

Функции явного курсора

- Поочередная обработка строк, возвращаемых запросом.
- Отслеживание текущей обрабатываемой строки.
- Ручное управление курсорами в блоке PL/SQL.

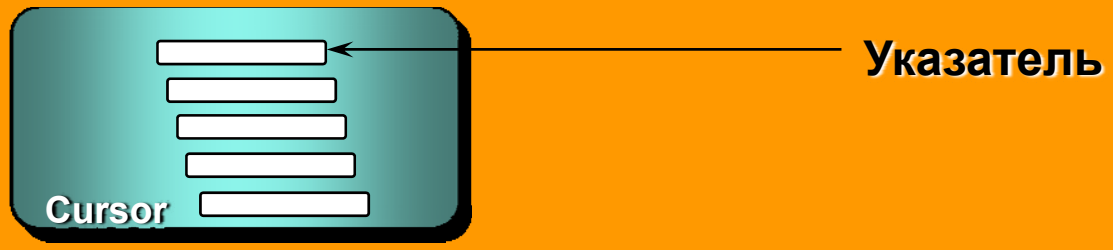
Управление явными курсорами



- Создание именованной рабочей области SQL
- Выявление активного набора строк
- Загрузка текущей строки в перемен-ные
- Проверка на наличие строки
- Возврат к FETCH если строка обнаружена
- Освобож-дение активного набора строк

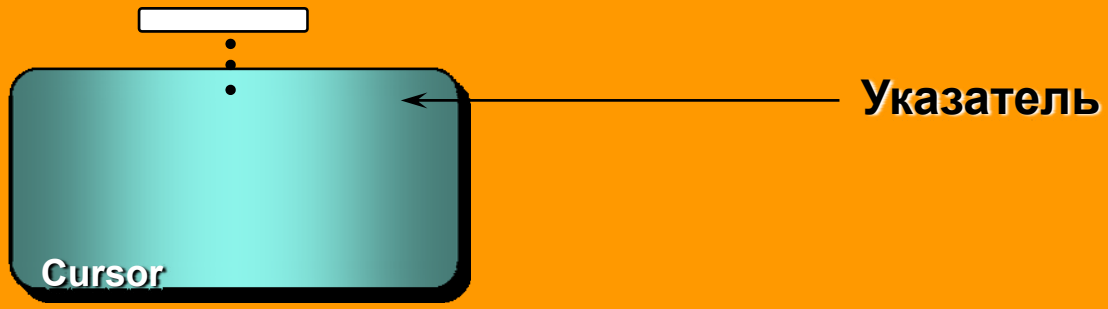
Управление явными курсорами

Открытие курсора



Выборка строки из курсора

Выборка до тех пор, пока не останется строк



Объявление курсора: синтаксис

```
DECLARE
  CURSOR cursor_name IS
      select_statement;
```

Не включайте выражение INTO в описание курсора.

Объявление курсора: пример

```
DECLARE
   . . .
   v_ord_id          s_item.ord_id%TYPE;
   v_product_id      s_item.product_id%TYPE;
   v_item_total      NUMBER (11,2);
   CURSOR item_cursor IS
     SELECT   product_id, price*quantity
     FROM     s_item
     WHERE    ord_id = v_ord_id;
BEGIN
. . .
```

Открытие курсора: синтаксис

```
OPEN cursor_name;
```

Выборка данных из курсора: синтаксис

```
FETCH cursor_name INTO variable1, variable2,...;
```

- Значения текущей строки выбираются в выходные переменные.
- Включается столько переменных, сколько столбцов в запросе.
- Последовательность имен переменных должна соответствовать последовательности столбцов.
- Проверьте, есть ли строки в курсоре.

Выборка данных из курсора: пример

```
FETCH item_cursor
    INTO v_product_id, v_item_total;
```

Закрытие курсора: синтаксис

```
CLOSE cursor_name;
```

Атрибуты явного курсора

Информацию о состоянии курсора можно получить с помощью атрибутов курсора.

Атрибут	Тип	Описание
%ISOPEN	Boolean	Истинно (TRUE), если курсор открыт.
%NOTFOUND	Boolean	Истинно (TRUE), если последняя команда FETCH не вернула строку.
%FOUND	Boolean	Истинно (TRUE), пока последняя команда FETCH возвращает строку.
%ROWCOUNT	Number	Общее количество строк, выбранных на данный момент.

Управление многократной выборкой

- Для обработки нескольких строк из явного курсора можно организовать цикл.
- При каждом выполнении цикла извлекается одна строка.
- Проверку на неудачную выборку можно сделать с помощью атрибута %NOTFOUND.
- Успех каждой выборки можно проверить с помощью атрибутов явного курсора.

Атрибут %ISOPEN: пример

- Выборка строк возможна только при открытом курсоре.
- Прежде, чем выполнять операцию FETCH, проверьте, открыт ли курсор, с помощью атрибута %ISOPEN.

```
IF item_cursor%ISOPEN THEN
  FETCH item_cursor INTO v_quantity,
v_price;
ELSE
  OPEN item_cursor;
END IF;
```

Атрибуты %NOTFOUND и %ROWCOUNT: пример

- Выбрать точное количество строк можно с помощью атрибута курсора %ROWCOUNT.
- Момент выхода из цикла определяется по атрибуту курсора %NOTFOUND.

```
LOOP
  FETCH item_cursor
     INTO v_product_id, v_item_total;
  EXIT WHEN item_cursor%ROWCOUNT > 5
     OR item_cursor%NOTFOUND;
  v_order_total := v_order_total + v_item_total;
. . .
END LOOP;
```

Курсоры и записи: пример

Строки из активного набора строк удобно обрабатывать, выбирая значения в запись (RECORD) PL/SQL.

```
  CURSOR emp_cursor IS
    SELECT  id, salary, start_date, rowid
    FROM    s_emp
    WHERE   dept_id = 41;
  emp_recordemp_cursor%ROWTYPE;
BEGIN
  OPEN emp_cursor;
. . .
  FETCH emp_cursor INTO emp_record;
```

Курсоры с параметрами: синтаксис

```
CURSOR cursor_name
    [(parameter_name datatype, ...)]
  IS
  select_statement;
```

Курсоры с параметрами: пример

```
CURSOR emp_cursor
        (v_dept NUMBER, v_job VARCHAR2) IS
 SELECT      last_name, salary, start_date
 FROM        s_emp
 WHERE       dept_id = v_dept
 AND         title = v_job;
```

Циклы FOR с курсорами: синтаксис

```
FOR record_name IN cursor_name LOOP
  statement1;
  statement2;
  . . .
END LOOP;
```

Циклы FOR с курсорами: пример

```
FOR item_record IN item_cursor LOOP
        -- неявное открытие и неявная выборка
  v_order_total := v_order_total +
     (item_record.price * item_record.quantity);
  i := i + 1;
  product_id_table (i) := item_record.product_id;
  order_total_table (i) := v_order_total;
END LOOP;                    -- неявное закрытие
```

Предложение WHERE CURRENT OF

- Требует предварительной блокировки строк с помощью предложения FOR UPDATE в запросе.

```
SELECT...FROM...FOR UPDATE [OF column-reference][NOWAIT]
```

- Используется для ссылки на текущую строку явного курсора.
- Если используется предложение FOR UPDATE, фиксация транзакций (COMMIT) между выборками из явного курсора не допускается.

```
...
  CURSOR emp_cursor IS
    SELECT ...
    FOR UPDATE;
BEGIN
  ...
  FOR emp_record IN emp_cursor LOOP
    UPDATE ...
      WHERE CURRENT OF emp_cursor;
    ...
  END LOOP;
  COMMIT;
END;
```

Курсорная переменная

Понятие курсорной переменной

Курсорная переменная (cursor variable) может быть связана с различными операторами во время выполнения программы.

Курсорные переменные аналогичны переменным PL/SQL, в которых могут содержаться различные значения.

Объявление курсорной переменной

Курсорные переменные имеют ссылочный тип. С помощью та кого типа можно именовать области хранения данных во время выполнения программы. Чтобы воспользоваться ссылочным типом, необходимо сначала объявить переменную, а затем выделить область памяти.

```
TYPE имя_типа IS REF CURSOR [RETURN возвращаемый_тип];
```

имя_типа — это имя нового ссылочного типа, а *возвращаемый_тип* - тип записи, указывающий типы списка выбора, которые в итоге будут возвращаться курсорной переменной.

Пример объявления курсорной переменной

```
DECLARE
   - Описание при помощи %ROWTYPE
TYPE t_StudentsRef IS REF CURSOR
RETURN students%ROWTYPE;
   - Определяем новый тип записи,
TYPE t_NameRecord IS RECORD (
        first_name students.first_name%TYPE,
        last_name students. last_name%TYPE);
   - переменную этого типа
v_NameRecord t_NameRecord;
    - и курсорную переменную, использующую этот тип записи.
TYPE t_NamesRef IS REF CURSOR
        RETURN t_NameRecord;
    - При помощи %TYPE можно объявить еще один тип.
TYPE t_NamesRef2 IS REF CURSOR
        RETURN t_NameRecord%TYPE;
     - Объявим курсорные переменные.
v_StudentCV t_StudefitsRef;
v_NameCV t_NamesRef;
```

Ограниченные и неограниченные курсорные переменные

Ограниченные курсорные переменные (constrained) объявляются только для конкретного возвращаемого типа. Переменная должна открываться для такого запроса, список выбора которого соответствует типу, возвращаемому курсором. В противном случае возникает предопределенная исключительная ситуация ROWTYPE_MISMATCH.
Для *неограниченных курсорных переменных* (unconstrained) предложение RETURN отсутствует. Такая переменная может быть открыта для любого запроса.

```
DECLARE
    - Определим неограниченный ссылочный тип
TYPE t_FlexibleRef IS REF CURSOR;
    - и переменную этого типа.
v_CursorVar t_FlexibleRef;
```

Открытие курсорной переменной для запроса

OPEN *курсорная_переменная* FOR *оператор_select;*

где *курсорная_переменная* — это ранее объявленная курсорная переменная,
а *оператор_sekct* — требуемый запрос.

После выполнения OPEN...FOR можно считывать информацию из курсорной переменной.

Использование курсорной переменной. Пример.

```
PROCEDURE SEL_TEST
  ( in_last_name              IN     VARCHAR2 DEFAULT NULL,
   in_date_birth              IN   VARCHAR2 DEFAULT NULL )
 IS
 TYPE ref_cursor IS REF CURSOR;
   v_sql_stmt   VARCHAR2 (4000);
   v_comma      CHAR (1)        := NULL;
   out_cur     ref_cursor;
 BEGIN
   v_sql_stmt := 'select * FROM PERSON ';
  IF (in_last_name IS NOT NULL) or (in_last_name IS NOT NULL) THEN
         v_sql_stmt := v_sql_stmt||'WHERE';
   END IF;
   IF in_last_name IS NOT NULL THEN
          v_sql_stmt :=
           v_sql_stmt || v_comma || ' last_name='' || in_last_name
            || '';
    v_comma := ‘ and ';
   END IF;
```

Использование курсорной переменной. Пример. (Продолжение)

```
IF in_date_birth IS NOT NULL THEN
   v_sql_stmt :=
       v_sql_stmt
     || v_comma
     || ' date_birth=to_date('''
     || in_date_birth
     || ''',''DD-MM-YYYY'')';
   v_comma := ',';
 END IF;

 OPEN out_cur
 FOR v_sql_stmt;

 ….

END;
```

Ограничения на использование курсорных переменных

- Курсорные переменные нельзя объявлять в модуле. Сам тип можно, но переменную нельзя.
- Удаленные подпрограммы не могут возвращать значение курсорной переменной. Курсорные переменные могут передаваться между клиентской и серверной стороной PL/SQL (например, из клиента Oracle Form), но не между двумя серверами.
- Сборные конструкции PL/SQL (индексные таблицы, вложенные таблицы и изменяемые массивы) не могут хранить курсорные переменные. Аналогично, таблицы и представления базы данных не могут хранить столбцы REF CURSOR.
- Запрос, связанный с курсорной переменной в операторе OPEN...FOR, не может быть FOR UPDATE. Это ограничение снято в Oracle8i и выше.

Задание

1. Создайте процедуру TOP_DOGS1 для определения самых высокооплачиваемых служащих
   a. Для этого упражнения создайте новую таблицу с данными о служащих и их заработной плате.
   b. Включите параметр, чтобы пользователь мог ввести нужное количество самых высокооплачиваемых служащих (n).
   c. Создайте цикл FOR с курсором для выборки из таблицы S_EMP фамилий и заработной платы n самых высокооплачиваемых служащих.
   d. Сохраните фамилию и заработную плату в таблице TOP_DOGS.
   e. Предполагается что двух служащих с одинаковой заработной платой не существует.
   f. Проверьте особые случаи – например, с n=0 и с n, превышающим количество служащих в таблице S_EMP.
   g. После каждого теста удаляйте данные из таблицы TOP_DOGS.

Задание

2. Создайте хранимую процедуру ADD_STARS, которая в новом столбце STARS проставляет для каждого служащего по призовой звёздочке за каждый процент заработанных комиссионных. Используйте курсор и цикл WHILE.
   a. Для этого упражнения создайте в таблице S_EMP новый столбец для хранения звёздочек (*)
   b. Определите процент комиссионных для каждого служащего, округлив его до ближайшего целого числа. Рассмотрите случай, когда служащий не получает комиссионных.
   c. Добавляйте звёздочку в строку звёздочек за каждый процент комиссионных. Если, например, служащий получает 10 процентов комиссионных, символьная строка в столбце STARS должна содержать десять звёздочек.
   d. Проставьте соответствующее количество звёздочек для каждого служащего в столбце STARS.

Задание

3. Скопируйте процедуру TOP_DOGS1 из упражнения 1 и назовите новый вариант TOP_DOGS2. Измените процедуру TOP_DOGS2 с учётом случая, когда несколько служащих из упражнения 1 имеют одинаковую заработную плату. Для каждой фамилии в списке должны быть перечислены все служащие с такой же заработной платой.

   Выполните процедуру TOP_DOGS2. В качестве n введите число 6, 7 или 8. В этом случае в выходных данных должны появиться фамилии Ngao, Dumas и Quick-To_See. Если же n равно 9, 10 или 11, должны появиться фамилии Nagayama, Magee и Maduro.

Не забывайте полностью удалять данные из таблицы TOP_DOGS после каждого теста.

Задание

4. Напишите процедуру для печати фамилии служащих, чья заработная плата лежит в диапазоне плюс минус $100 от введённого значения.
   a. Если служащего с такой заработной платой нет, пользователь должен получить соответствующее сообщение. Используйте исключение.
   b. Если служащих с такой зарплатой более 3, сообщение должно указывать, сколько сотрудников попадёт в этот диапазон зарплат.
5. Создайте процедуру для определения:
   a. Сотрудников, работающих больше заданного числа лет.
   b. Для этих сотрудников определите менеджеров. Выведите без повторений полученный список менеджеров с указанием рядом с каждым именем через запятую имён сотрудников из начального списка.
   c. Определите и выведите на экран менеджеров, принявших суммарные заказы более заданной суммы.

Задание

6. Напишите процедуру, определяющую рейтинг лучших товаров по:

- суммарной цене приобретённого за указанный срок товара;
- количеству единиц приобретённого за указанный срок товара;
- количеству заказов, в которые входил приобретённого за указанный срок товар.

Т.е. первым выдаётся на печать товар, у которого максимальный показатель по выбранному критерию, затем товар с более низким показателем и т.д.

Какой из критериев использовать определяется по входным параметрам. Если указано несколько критериев, то их приоритет соответствует списку, указанному выше.

Задание

7. Скопируйте таблицу s_emp в s_emp_copy. Для s_emp_copy напишите пакет функций:

• Вставка строки в таблицу. Если отдел не введён то сотрудник заносится в самый малочисленный отдел.

•Выбор строки. Если указан id, то выводится строка с указанным ключом. Если нет, то происходит поиск по фамилии, имени, дате начала работы, году начала работы, отделу и заработной плате.

•Удаление. Если указана id, то удаляется строка с указанным ключом. Если нет, то происходит поиск по фамилии, имени, дате начала работы, году начала работы, отделу и заработной плате.

Разработать структуру логирования информации для таблицы s_emp_copy с возможностью восстановления данных за нужный период.

Задание

8. Напишите процедуру, осуществляющую поиск в зависимости от входных параметров:

1. *Указан интервал времени.* Поиск заказчиков у которых время, прошедшее от заказа до получения товара больше чем указанный интервал.

2.Наименование товара. Список заказчиков, закупавших товар с таким наименованием (при этом не поиск должен работать независимо от указанного регистра или числа пробелов).

3.Сумма. Заказчиков, приобретших товар на сумму более указанной.

Добавить в качестве входного параметра флаг. В первом случае он позволяет выбирать заказчиков по условиям независимо. Выбранных заказчиков без повторения поместить в специальную таблицу и в отдельном столбце в ней ставить столько плюсиков, сколько пунктов для заказчика истинно.

Второе значение флага позволяет выбирать заказчиков удовлетворяющих всем указанным пунктам.

Обработка исключений

Общие понятия

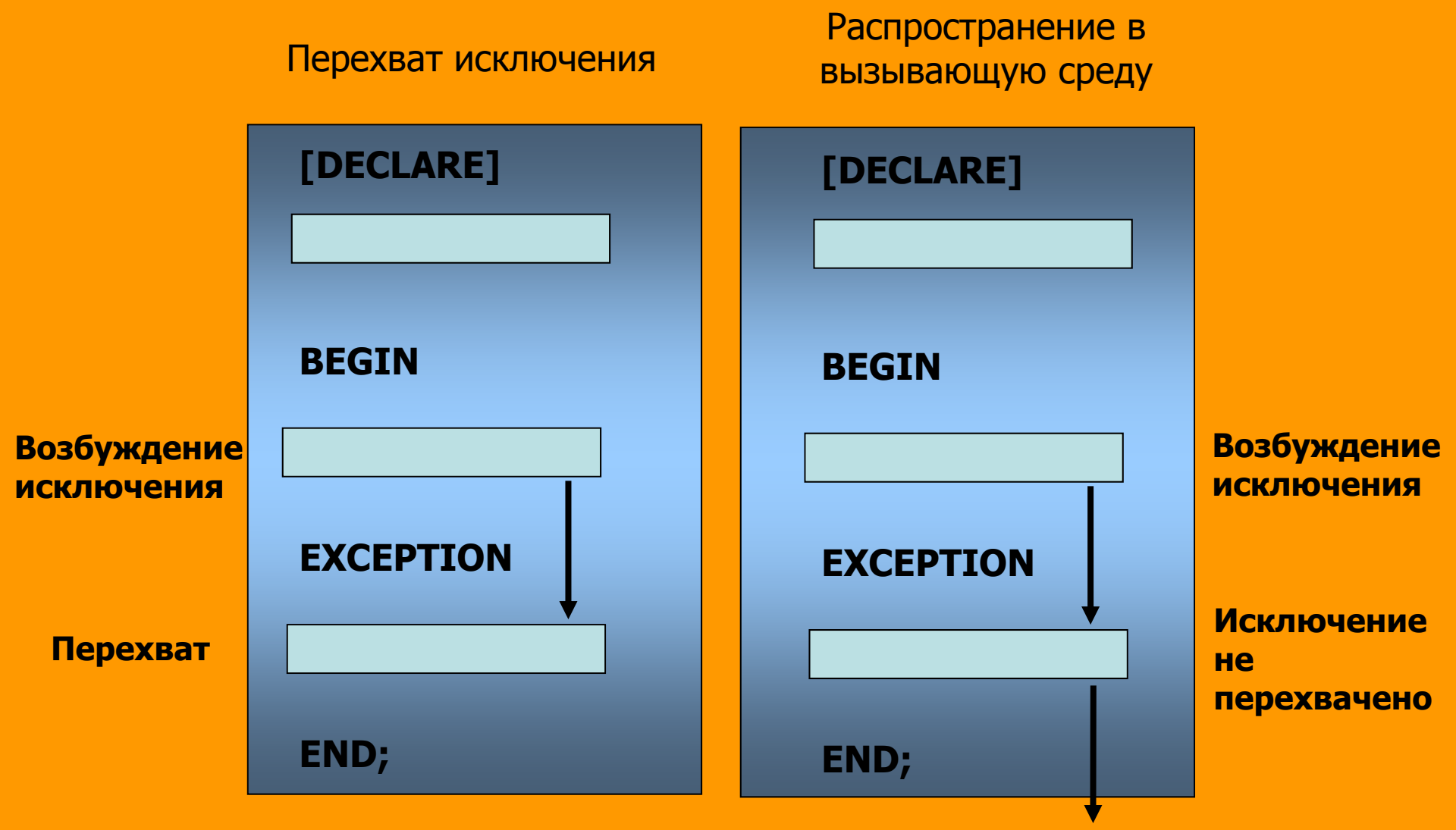
- Что такое исключение?

  Переменная в PL/SQL, возбуждаемая во время выполнения.

- Как возникает исключение?
  - Возбуждается сервером.
  - Возбуждается явно.

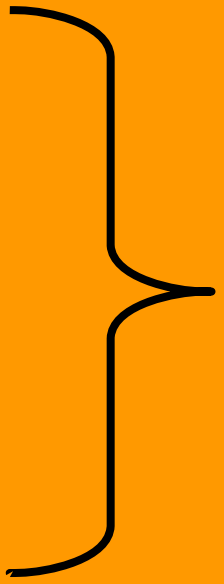
- Как его обрабатывать?
  - Перехватывать с помощью обработчика исключений.
  - Распространить в вызывающую среду.

Обработка исключений



Типы исключений

- Предопределенные, возбуждаемые сервером.
- Непредопределенные, возбуждаемые сервером.



Возбуждаются неявно

- пользовательские

Возбуждаются явно

Перехват исключений

```
EXCEPTION
   When исключение1 [or исключение2 …] Then
         оператор1;
         оператор1;
         ...

   [When исключение3 [or исключение4 …] Then
         оператор1;
         оператор1;
         ...

   When OTHERS Then
         оператор1;
         оператор1;
         ...]
```

Перехват предопределенных исключений

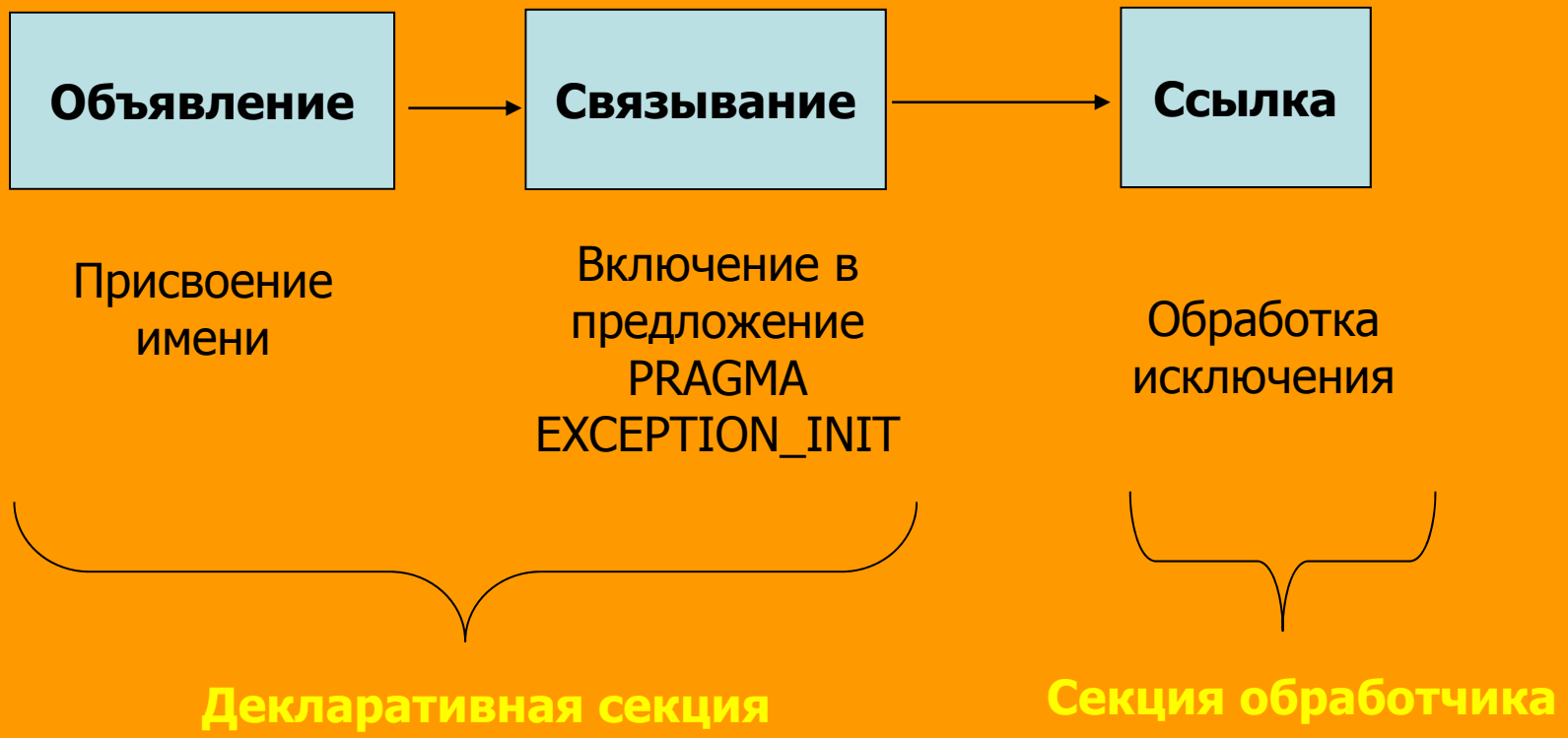
- В секции обработки ссылка на стандартное имя.

- Некоторые стандартные имена исключений:
  - NO_DATA_FOUND
  - TOO_MANY_ROWS
  - INVALID_CURSOR
  - ZERO_DIVIDE
  - DUP_VAL_OF_INDEX

Предопределенные исключения: пример

```
Procedure test_exception_1 (v_product_id IN s_product.id%type) IS
V_id              s_product.id%type;
BEGIN
Select id into v_id       from s_product where id=v_product_id;
Delete from s_invertory where product_id=v_product_id;
Commit;
EXCEPTION
When NO_DATA_FOUND Then
    rollback;
    text_io.put_line('Product number is invalid');
When TOO_MANY_ROWS Then
    rollback;
    text_io.put_line('Data corraption in table S_PRODUCT');
When OTHERS Then
    rollback;
    text_io.put_line('Other error occurred');
END;
```

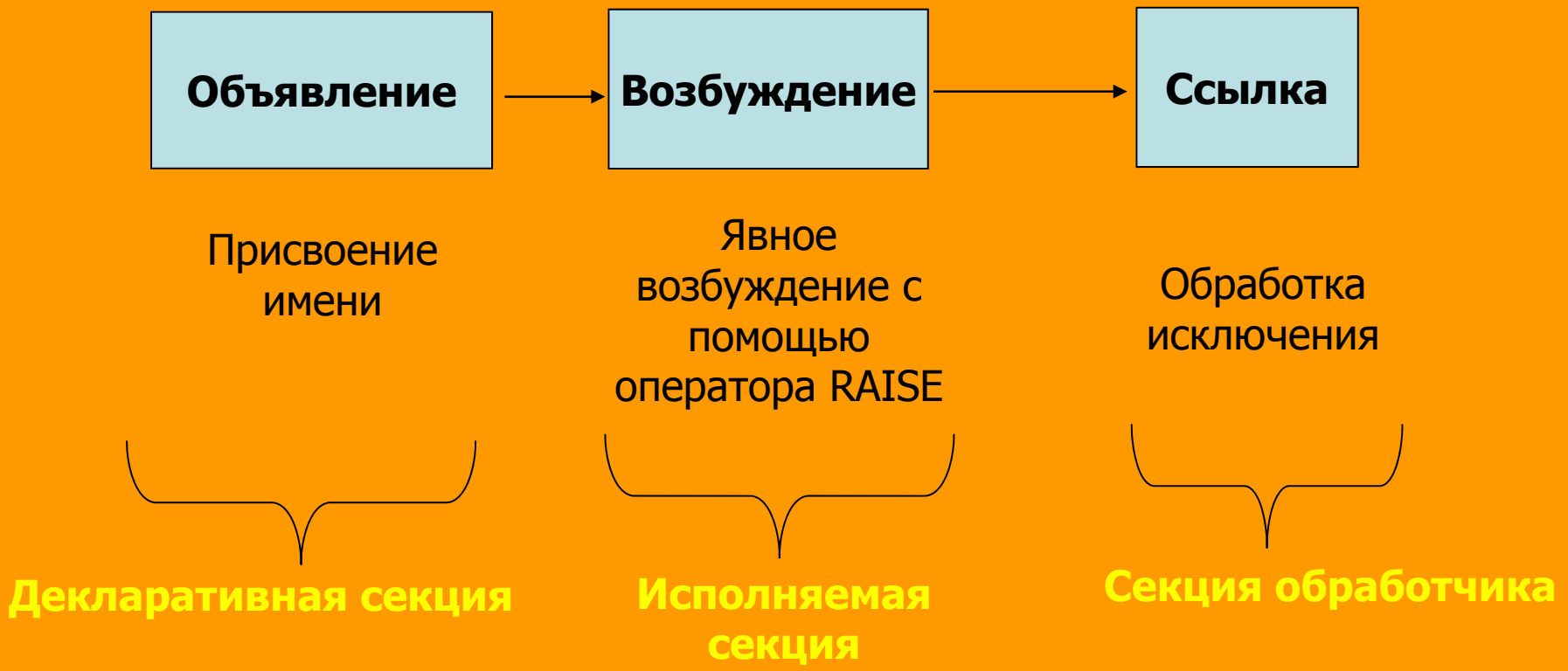
Перехват непредопределенных исключений



Непредопределенное исключение: пример

```
DECLARE
E_product_r      EXCEPTION;
PRAGMA EXCEPTION_INIT (E_product_r, -2222);
…
BEGIN
…
EXCEPTION
When E_product_r Then
Text_io.put_line('Integrity constraint violated');
…
END;
```

Перехват пользовательских исключений



Пользовательское исключение: пример

```
DECLARE
E_product_r      EXCEPTION;
…
BEGIN
…
RAISE E_product_r;
…
EXCEPTION
When E_product_r Then
        Text_io.put_line('Integrity constraint violated');
…
END;
```

Функции перехвата исключений

- Обработчик исключения WHEN OTHERS
  - Перехватывает все необработанные исключения.
  - Должен быть последним обработчиком.
- SQLCODE
  - Возвращает числовой код ошибки.
  - Значения:
    - 0 Исключений не было
    - 1 Пользовательское исключение
    - +100 Исключение NO_DATA_FOUND
    - -n Номер ошибки сервера
- SQLERRM
  - Возвращает стандартное сообщение об ошибке.

Функции перехвата исключений: пример

```
…
BEGIN
…
EXCEPTION
When OTHERS Then
        rollback;
        Text_io.put_line('Unknown error: '||SQLCODE||' '||SQLERRM);
…
END;
```

Распространение исключений в вызывающую среду

```
DECLARE
E_product_r          EXCEPTION;
E_integrity          EXCEPTION;
PRAGMA EXCEPTION_INIT (E_product_r, -2222);
…
BEGIN
        FOR I IN emp_cursor LOOP
                begin
                select …;
                update …;
                …
                RAISE E_integrity;
                …
                exception
                WHEN E_product_r THEN …
                 WHEN E_integrity THEN
                end;
        END LOOP;
EXCEPTION
When  NO_DATA_FOUND Then …
When OTHERS Then …
…
END;
```

Практическое занятие

1. Создайте процедуру, которая бы обновляла номера регионов для служащих отделов. (Входные данные – название региона и новый номер отдела для данного служащего).

а) Напишите обработчик исключений, который будет выдавать сообщение о том, что указанный регион не существует

б) Напишите обработчик исключений, который бы выдавал пользователю сообщение о том, что для указанного региона уже есть отдел с таким названием

в) Напишите обработчик исключений, который выдавал бы пользователю сообщение о том, что указанный номер отдела не существует (используйте атрибут SQL%NOTFOUND и возбудите исключение вручную)

2. Напишите процедуру для вывода на фамилии и названия отдела для служащих, чья заработная плата лежит в диапазоне плюс-минус 100$ от введенного значения.

а) Напишите обработчик исключений, который будет выдавать сообщение о том, что служащих с такой зарплатой нет.

б) Напишите обработчик исключений, который будет выдавать сообщение о том, что служащих с такой зарплатой несколько. Сообщение должно указывать, сколько сотрудников попадает в этот диапазон зарплат.

Встроенный динамический SQL

Динамический SQL

PL/SQL использует раннее связывание для выполнения операторов SQL. Следствием этого является то, что только операторы DML могут непосредственно включаться в блоки PL/SQL. Однако можно решить эту проблему с помощью динамического SQL.

Динамический SQL разбирается и исполняется во время выполнения, а не синтаксического разбора блока PL/SQL.

Динамический SQL

Существуют два способа выполнения динамического SQL в PL/SQL.

1. Первый применяет модуль DBMS_SQL.
2. Второй способ был введен в Oracle8i и предлагает использование встроенного динамического SQL. Встроенный динамический SQL является составной частью самого языка. Вследствие этого он значительно проще в применении и быстрее, чем модуль DBMS_SQL.

EXECUTE IMMEDIATE.

Базовым оператором, используемым в не содержащих запросов операторах (DML и DDL) и блоках PL/SQL, является оператор EXECUTE IMMEDIATE.

Выполняемая строка может задаваться как литерал, заключенный в одиночные кавычки или как переменная типа символьной строки PL/SQL.

Завершающая точка с запятой не нужна для операторов DML и DDL, но указывается для анонимных блоков.

EXECUTE IMMEDIATE. Пример.

В этом примере показаны различные способы использования EXECUTE IMMEDIATE: для выполнения DDL, DML и анонимных блоков PL/SQL.

```
BEGIN
EXECUTE IMMEDIATE
                    'CREATE TABLE execute_table (call VARCHAR2(10))';
FOR v_Counter IN 1..10 LOOP
        v_SQLString :=
                    'INSERT INTO execute_table
                    VALUES ("Row' || v_Counter || '")';
        EXECUTE IMMEDIATE v_SQLString;
END LOOP;
v_PLSQLBlock :=
        'BEGIN
                    FOR v_Rec IN (SELECT * FROM execute_table) LOOP
                    DBMS_OUTPUT.PUT_LINE(v_Rec.call);
                    END LOOP;
        END;';
EXECUTE IMMEDIATE v_PLSQLBlock;
EXECUTE IMMEDIATE 'DROP TABLE execute_table ';
END;
```

EXECUTE IMMEDIATE.

EXECUTE IMMEDIATE используется также для выполнения операторов со связанными переменными.

В этом случае выполняемая строка содержит *специальные позиции,* помеченные двоеточием.

Позиции предназначены для размещения переменных PL/SQL, которые указываются в предложении USING оператора EXECUTE IMMEDIATE.

EXECUTE IMMEDIATE. Пример 2..

```
BEGIN
- Вставим ECN 103 в таблицу classes, используя строку символов
- для оператора SQL.
v_SQLString :=
        'INSERT INTO CLASSES (department, course, description,
                max_students, current_students,
                num_credits)
        VALU€S(:dep, :course, :descr, :max_s, :cur_s, :num_c)';
EXECUTE IMMEDIATE v_SQLString USING
        'ECN', 103, 'Economics 103', 10, 0, 3;
- Зарегистрируем всех выбравших Economics в новой группе.
FOR v_StudentRec IN c_EconMajor LOOP
        - Здесь мы имеем литеральный оператор SQL, а переменные PL/SOL
        - находятся в предложении USING.
ECECUTE IMMEDIATE
        'INSERT INTO registered_students
                (student_ID, department, course, grade)
        VALUES (:id, :dep, :course, NULL)'
USING v_Studentflec.ID, 'ENC', 103;
END;
```

OPEN FOR

Запросы выполняются с помощью оператора OPEN FOR аналогично курсорным переменным. Различие состоит в том, что строка, содержащая запрос, может быть переменной PL/SQL, а не литералом.

К получаемой курсорной переменной можно обращаться так же, как и к любой другой переменной.

Для связывания используется предложение USING, так же как в операторе EXECUTE IMMEDIATE.

```
BEGIN
v_SQLStatement := 'SELECT * FROM students ' || p_WhereClause;
OPEN v_ReturnCursor FOR v_SQLStatement;

v_SQLStatement := 'SELECT * FROM students WHERE major = :m';
OPEN v_ReturnCursor FOR v_SQLStatement USING p_Major;

END;
```

Массовые соединения

Операторы SQL в блоках PL/SQL пересылаются системе поддержки SQL, которая в свою очередь может передавать данные назад системе поддержки PL/SQL (как результат запроса).

Во многих случаях данные, которые вносятся или обновляются в базе данных, помещаются сначала в сборную конструкцию PL/SQL, и затем эта сборная конструкция просматривается с помощью цикла FOR для отправки информации системе поддержки SQL. Это приводит к переключению контекста между PL/SQL и SQL для каждой строки в сборной конструкции.

Oracle8i и выше позволяет передавать все строки сборной конструкции системе поддержки SQL с помощью одной операции, оставляя только одно переключение контекста. Это называется *массовым соединением,* оно выполняется с помощью оператора FORALL.

Массовые соединения. Пример.

```
DECLARE
        TYPE t_Numbers IS TABLE OF temp_table.num_col%TYPE;
        TYPE t_Strings IS TABLE OF temp_table.char_col%TYPE;
        v_Numbers t_Numbers := t_Numbers(1);
        v_Strings t_Strings := t_Strings(1);
- Печатаем общее число строк таблицы temp_table.
PROCEDURE PrintTotalRows (p_Message IN VARCHAR2) IS
        v_Count NUMBER:
BEGIN
        SELECT COUNT(*)
                INTO v_Count
                FROM temp_table;
        DBMS_OUTPUT.PUT_LINE(p_Message || ': Count is ' || v_Count);
END PrintTotalRows;
BEGIN
DELETE FROM temp_table;
-- Заполняем вложенные таблицы PL/SQL, используя 1000 значений.
v_Numbers.EXTEND(1000);
v_Strings.EXTEND(1000);
FOR v_Count IN 1..1000 LOOP
        v_Numbers(v_Count) := v_Count;
        v_Strings(v_count) := 'Element #' || v_Count;
END LOOP;
```

Массовые соединения. Пример (продолжение).

```
-- Внесем в базу данных все 1000 элементов с помощью оператора FORALL.
FORALL v_Count IN 1..1000
        INSERT INTO temp_table VALUES
                (v_Numbers(v_Count), v_Strings(v_Count));
- Теперь должно быть 1000 строк.
        PrintTotalRows('After first insert');
-- Снова внесем в базу данных элементы с 501 по 1000.
FORALL v_Count IN 501..1000
        INSERT INTO temp_table VALUES
        (v_Numbers(v_count), v_Strings(v_Count));
-- Теперь у нас должно быть 1500 строк.
        PrintTotalRows('After second insert');
-- Обновим все строки.
FORALL v_Count IN 1..1000
        UPDATE temp_table
                SET char_col = 'Changed!'
                WHERE num_col = v_Numbers(v_Count);
- Несмотря на то, что имеется только 1000 элементов, этот оператор
- обновляет 1500 строк, так как предложение WHERE соответствует
- 2 строкам для каждой из последних 500 строк.
DBMS_OUTPUT.PUT_LINE(
        'Update processed ' || SQL%ROWCOUNT || 'rows.');
```

Массовые соединения. Пример (продолжение).

```
-- Аналогично, этот DELETE удалит 300 строк.
FORALL V_Count IN 401..600
        DELETE FROM tempjtable
        WHERE nun_col = v_Numbers(v_Count);
-- Поэтому должно остаться 1200 строк.
PrintTotalRows('After delete');
END;
```

Результатом выполнения примера будет следующее:

```
After first insert: Count is 1000
After second insert: Count is 1500
Update processed 1500 rows.
After delete: Count is 1200
```

FORALL синтаксически аналогичен циклу FOR. Он может использоваться для сборных конструкций любого типа и для операторов INSERT, DELETE и UPDATE. Определяемый в FORALL диапазон должен быть непрерывным, и все элементы в этом диапазоне должны существовать.

Особенности использования транзакций

Если в массовой операции DML при обработке одной из строк возникает ошибка, то откатывается только эта строка. Предыдущие строки будут обработаны.

В Oracle9i можно указать в операторе FORALL новую конструкцию SAVE EXCEPTIONS. При этом любая ошибка, возникшая во время пакетной обработки, будет сохранена, а обработка будет продолжена.

Для просмотра исключений можно использовать новый атрибут SQL%BULK_EXCEPTIONS, который действует как таблица PL/SQL.

DBMS_SQL

DBMS_SQL используется для выполнения динамического SQL в PL/SQL. Он не встроен непосредственно в язык и поэтому менее эффективен, чем встроенный динамический SQL (который доступен в Oracle8г и выше).

Модуль DBMS_SQL позволяет непосредственно управлять обработкой операторов в курсоре, выполнять синтаксический разбор оператора, связывать входные переменные и определять выходные переменные.

DBMS_SQL. Пример.

```
CREATE OR REPLACE PROCEDURE UpdateClasses(
/* Использует DBMS_SQL для обновления таблицы учебных групп, задания
 числа зачетов для всех групп на указанном факультете.
*/
p_Department IN classes.department%TYPE,
p_NewCredits IN classes. num_credits%TYPE,
p_RowsUpdated OUT INTEGER) AS
v_CursorID INTEGER;
v_UpdateStmt VARCHAR2(100);

BEGIN
- Откроем курсор для обработки.
v_CursorID := DBMS_SQL.OPEN_CURSOR;
- Определим строку SQL.
v_UpdateStmt :=
        'UPDATE classes
                SET num_credits = :nc
                WHERE department = :dept';
```

DBMS_SQL. Пример. (продолжение)

```
  -- Выполним синтаксический разбор оператора.
  DBMS_SQL.PARSE(v_CursorID, v_UpdateStrat, DBMS_SQL.NATIVE);
  -- Свяжем p_NewCredits с позицией :nc. Эта перегруженная версия
  -- BIND_VARIABLE привяжет p_NewCredits как NUMBER,
  -- поскольку он так объявлен.
  DBMS_SQL.BIND_VARIABLE(v_CursorID, ':nc', p_NewCredits);
  -- Свяжем p_Department с позицией :dept. Эта перегруженная версия
  -- BIND_VARIABLE привяжет p_Department как CHAR, поскольку он
  -- так объявлен.
  DBMS_SQL.BIND_VARIABLE_CHAR(v_CursorID, ':dept ', p_Department);
  -- Выполним оператор
  p_RowsUpdated := DBMS_SQL.EXECUTE(v_CursorID);
  -- Закроем курсор
  DBMS_SQL.CLOSE_CURSOR(v_CursorID);
EXCEPTION
    WHEN OTHERS THEN
     -- Закроем курсор и снова инициируем ошибку.
    DBMS_SQL.CLOSE_CURSOR(v_CursorID);
    RAISE;
END UpdateClasses;
```

Задания

1. Напишите процедуру, возвращающую список товаров, количество каждого товара и его цену. Процедура должна иметь два входных параметра - название фирмы и название товара. Если указано название фирмы, выдаётся список всех купленных ею товаров. Если указано название фирмы и дополнительно указано наименование товара, то выдаётся товар, купленный фирмой, наименование которого совпадает с заданным наименованием. Если указано только наименование товара, то выдаётся указанный товар.

   Процедуру написать с использованием позиций предназначенных для размещения переменных PL/SQL, указываемых в предложении USING оператора EXECUTE IMMEDIATE.
2. Выполните первое задание используя DBMS_SQL.

Триггеры баз данных

Понятие триггера

ТРИГГЕР БАЗЫ ДАННЫХ - это хранимая программная единица PL/SQL, ассоциированная с конкретной таблицей базы данных.

В отличие от подпрограмм, которые должны вызываться явно, триггер базы данных вызывается неявно и не имеет атрибутов.

Акт выполнения триггера называется его *активизацией* (firing). Событием, запускающим триггер, является операция DML (INSERT, UPDATE или DELETE), выполняемая над таблицей или представлением базы данных. В Oracle8i эти функции расширены: триггер может срабатывать на системное событие, например на запуск или останов базы данных, а также на определенные виды операций DDL.

Триггеры данных используются для

- контроля за информацией, хранимой в таблице, посредством регистрации вносимых изменений и пользователей, производящих эти изменения.
- реализации сложных ограничений целостности данных, которые невозможно реализовать через декларативные ограничения, устанавливаемые при создании таблицы.
- автоматического оповещения других программ о том, что необходимо делать в случае изменения информации, содержащейся в таблице.
- осуществления сложных процедур защиты;
- поддержки дублированных таблиц.

Типы триггеров

Триггер DML активизируется оператором DML, и тип триггера определяется типом этого оператора. Триггеры DML задаются для операций ввода, обновления и удаления информации (INSERT, UPDATE, DELETE). Они активизируются до или после операции, на уровне строки или оператора.

В Oracle8i предлагается еще один вид триггеров. *Триггеры замещения* (instead of) можно создавать только для представлений (либо объектных, либо реляционных). В отличие от триггеров DML, которые выполняются в дополнение к операторам DML, триггеры замещения выполняются вместо операторов DML, вызывающих их срабатывание. Триггеры замещения должны быть строковыми триггерами.

В Orac1e8i и выше существует третий тип триггеров. *Системный триггер* активизируется не на операцию DML, выполняемую над таблицей, а на системное событие, например, на запуск или останов базы данных. Системные триггеры срабатывают и на операции DDL, такие как создание таблицы.

Привилегии для создания триггера

Для создания триггера базы данных необходимо иметь привилегии CREATE TRIGGER, а также либо владеть ассоциированной таблицей, либо иметь привилегии ALTER для ассоциированной таблицы, либо иметь привилегии ALTER ANY TABLE.

Триггер базы данных состоит из трех частей: события триггера, необязательного ограничения триггера и действия триггера. Когда происходит событие триггера, триггер базы данных возбуждается, и анонимный блок PL/SQL выполняет предписанное действие. Триггеры базы данных возбуждаются с привилегиями владельца, а не текущего пользователя. Поэтому владелец должен иметь должный доступ ко всем объектам, вовлекаемым в действие триггера.

Создание триггера

CREATE [OR REPLACE] TRIGGER ***имя_триггера***

[BEFORE | AFTER | INSTEAD OF] ***активизирующее_событие ссылочное_предложение***

[WHEN ***условие_срабатывания*****]**
[FOR EACH ROW]
тело_триггера;

Внимание! Тело триггера не может превышать 32 Кбайт. Если триггер больше, то его следует уменьшить, перенеся часть программного текста в отдельно компилируемые модули или хранимые процедуры и вызывая их в теле триггера. Ограничение размера тела триггеров обусловлено частотой их выполнения.

Создание триггера DML

Триггер DML активизируется операцией INSERT (ввод), UPDATE (обновление) или DELETE (удаление), выполняемой над таблицей базы данных. Триггеры могут активизироваться до (BEFORE) или после (AFTER) операции и действовать на уровне строки или оператора. Тип триггера определяется комбинацией этих факторов. Существует 12 возможных видов: 3 оператора x 2 момента времени x 2 уровня. Ниже приведены примеры правильных триггеров DML:

- До выполнения операции обновления на операторном уровне
- После выполнения операции ввода на уровне строк
- До выполнения операции удаления на уровне строк

Кроме того, триггер может активизироваться несколькими типами операторов DML, выполняемых над конкретной таблицей: например, INSERT и UPDATE. Код триггера выполняется вместе с активизирующим оператором как часть одной транзакции.

Виды триггеров DML

Категория	Значение	Комментарии
Оператор	INSERT, DELETE или UPDATE	Определяет, какой оператор DML активизирует триггер.
Момент времени	BEFORE или AFTER	Определяет момент активизации триггера: до или после выполнения оператора.
Уровень	Строка или оператор	Если триггер является строковым, то он активизируется один раз для каждой из строк, на которые воздействует оператор, вызывающий срабатывание триггера. Если триггер является операторным, то он активизируется один раз до или после оператора. Строковые триггеры идентифицируются предложением FOR EACH ROW (для каждой строки) в описании триггера.

Порядок активизации триггеров DML

Триггеры активизируются при выполнении оператора DML.

Алгоритм выполнения оператора DML таков:

1. Выполняются операторные триггеры BEFORE (при их наличии).
2. Для каждой строки, на которую воздействует оператор:
   A. Выполняются строковые триггеры BEFORE (при их наличии).
   B. Выполняется собственно оператор.
   C. Выполняются строковые триггеры AFTER (при их наличии).
3. Выполняются операторные триггеры AFTER (при их наличии).

Создание триггера DML. Пример 1.

```
CREATE SEQUENCE trig_seq
      START WITH 1
      INCREMENT BY 1;

CREATE OR REPLACE PACKAGE TrigPackage AS
      - Глобальный счетчик для использования в триггерах
      v_Counter NUMBER;
END TrigPackage;

CREATE OR REPLACE TRIGGER classesBStatement
      BEFORE UPDATE ON classes
BEGIN
      - Сначала сбросим счетчик.
      TrigPackage.v_Counter := 0;
      INSERT INTO temp_table (num_col, char_col)
            VALUES (trig_seq.NEXTVAL,
            'Before Statement: counter = ' | |
TrigPackage.v_Counter);
      - А теперь увеличим его значение для следующего
триггера.
      TrigPackage.v_Counter := TrigPackage.v_Counter+ 1;
END ClassesBStatement;
```

Создание триггера DML. Пример 1(продолжение)

```
CREATE OR REPLACE TRIGGER ClassesAStatementl
     AFTER UPDATE ON classes
BEGIN
     INSERT INTO temp_table (num_col. char_col)
            VALUES (trig'_seq.NEXTVAL,
            'After Statement 1: counter = ' |
TrigPackage.v_Counter);
     - Увеличим для следующего триггера.
     TrigPackage.v_Counter := TrigPackage.v_Counter+ 1;
END ClassesAStatementl;

CREATE OR REPLACE TRIGGER ClassesAStatement2
     AFTER UPDATE ON classes
BEGIN
     INSERT INTO temp_table (num_col, char_col)
            VALUES (trig_seq.NEXTVAL,
            'After Statement 2: counter = ' ||
TrigPackage.v_Counter);
     - Увеличим для следующего триггера.
     TrigPackage.v_Counter := TrigPackage.v_Counter+ 1;
END ClassesAStatement2;
```

Создание триггера DML. Пример 1(продолжение)

```
CREATE OR REPLACE TRIGGER ClassesBRowl
     BEFORE UPDATE ON classes
     FOR EACH ROW
BEGIN
     INSERT INTO temp_table (num_col, char_col)
            VALUES (trig_seq.NEXTVAL,
            'Before Row 1: counter = ' ||
TrigPackage.v_Counter);
     - Увеличим для следующего триггера.
     TrigPackage.vJDounter := TrigPackage.v_Counter+ 1;
END ClassesBRowl;

CREATE OR REPLACE TRIGGER ClassesBRow2
     BEFORE UPDATE ON classes
     FOR EACH ROW
BEGIN
     INSERT INTO temp_table (num_col, char__col)
            VALUES (trig_seq.NEXTVAL,
            'Before Row 2: counter = ' |
TrigPackage.v_Counter);
     - Увеличим для следующего триггера.
     TrigPackage.v_Counter := TrigPackage.v_Counter+ 1;
END ClassesBRow2;
```

Создание триггера DML. Пример 1(продолжение)

```
CREATE OR REPLACE TRIGGER ClassesBRowS
     BEFORE UPDATE ON classes
     FOR EACH ROW
BEGIN
     INSERT INTO temp_table (num_col, char_col)
            VALUES (trig_seq.NEXTVAL,
            'Before Row 3: counter = ' ||
TrigPackage.v_Counter);
     - Увеличим для следующего триггера.
     TrigPackage.v_Counter := TrigPackage,v_Counter+ 1;
END ClassesBRowS;

CREATE OR REPLACE TRIGGER ClassesARow
     AFTER UPDATE ON classes
     FOR EACH ROW
BEGIN
     INSERT INTO temp_table (num_col, char_col)
     VALUES (trig_seq.NEXTVAL,
            'After Row: counter = ' ||
TrigPackage.v_Counter);
     - Увеличим для следующего триггера.
     TrigPackage.v_Counter := TrigPackage.v_Counter+ 1;
END ClassesARow;
```

Создание триггера DML. Пример 1(продолжение)

Выполним следующий оператор UPDATE:

```
UPDATE classes
        SET num_credits = 4
        WHERE department IN ('HIS', ' C S ' ) ;
```

Этот оператор воздействует на четыре строки. Операторные триггеры BEFORE и AFTER выполняются по разу, а строковые триггеры BEFORE и AFTER — по четыре раза.

При активизации каждого из триггеров будут видны изменения, сделанные предыдущими триггерами, а также изменения в базе данных, внесенные оператором. Порядок, в котором активизируются триггеры одного вида, не определен. Из приведенного примера следует, что каждый триггер видит изменения, вносимые более ранними триггерами. Если порядок важен, следует объединить все операции в один триггер.

Создание триггера DML. Пример 1(продолжение)

```
SQL> SELECT * FROM temp_table
     2 ORDER BY num_col;
```

NUM_COL	CHAR_COL
1	Before Statement: counter = 0
2	Before Row 3: counter = 1
3	Before Row 2: counter = 2
4	Before Row 1: counter = 3
5	After Row : counter = 4
6	Before Row 3: counter = 5
7	Before Row 2: counter = 6
8	Before Row 1: counter = 7
9	After Row : counter = 8
10	Before Row 3: counter = 9
11	Before Row 2: counter = 10
12	Before Row 1: counter = 11
13	After Row : counter = 12
14	Before Row 3: counter = 13
15	Before Row 2: counter = 14
16	Before Row 1: counter = 15
17	After Row : counter = 16
18	After Statement 2: counter = 17
19	After Statement 1: counter = 18

Идентификаторы корреляции : old и : new

Строковый триггер запускается один раз для каждой строки, обрабатываемой активизирующим оператором. Внутри триггера можно обращаться к данным строки, обрабатываемой в данный момент. Для этого служат два идентификатора корреляции — *:old* и *:new*.

Идентификатор корреляции (correlation identifier) — это переменная привязки PL/SQL особого рода.

Двоеточие перед идентификатором указывает на то, что это переменные привязки (подобны базовым переменным, используемым во встроенном PL/SQL), а не обычные переменные PL/ SQL. Компилятор PL/ SQL рассматривает их как записи типа *активизирующая_таблица%ROWTYPE,* где *активизирующая_таблица* — это таблица, для которой создан триггер. Следовательно, ссылка типа *:new.поле* будет достоверна, если только поле является полем активизирующей таблицы.

Идентификаторы корреляции : parent

В Oracle8i определен еще один идентификатор корреляции — *:parent*. Если триггер создается для вложенной таблицы, *:old* и :new ссылаются на ее строки, а *:parent* — на текущую строку родительской таблицы

Активизирующий оператор	**:old**	**:new**
INSERT	Не определено — во всех полях содержится NULL	Значения, которые будут введены после выполнения оператора
UPDATE	Исходные значения, содержащиеся в строке перед обновлением данных	Новые значения, которые будут введены после выполнения оператора
DELETE	Исходные значения, содержащиеся в строке перед ее удалением	Не определено — во всех полях содержится NULL

Идентификаторы корреляции. Пример.

```
CREATE OR REPLACE TRIGGER GenerateStudentID
        BEFORE INSERT OR UPDATE ON students
        FOR EACH ROW
BEGIN
        /* Заполним поле ID таблицы students следующим значением из
        student_sequence. Поскольку ID - это столбец таблицы students,
        :new.ID является допустимой ссылкой. */
        SELECT student_sequence.NEXTVAL
        INTO :new.ID
        FROM dual;
END GenerateStudentID;
```

GenerateNewStudentID модифицирует значение *:new*.ID. Это одно из полезных свойств *:new* — когда выполнение оператора завершается, используются те значения, которые содержатся в *:new*. Нельзя изменить *:new* в строковом триггере AFTER, так как оператор будет обработан раньше. Вообще говоря, *:new* модифицируется только в строковых триггерах BEFORE; *:old* никогда не модифицируется, а лишь считывается.

Псевдозаписи

Хотя *:new* и *:old* синтаксически рассматриваются в качестве записей типа *активитрующая_таблица%ROWTYPE,* в действительности они записями не являются. Поэтому операции, применимые к записям, не могут быть выполнены над *:new* и *:old*. Например, эти псевдозаписи нельзя присваивать чему-либо в качестве целых записей.

Записи *:new* и *:old* разрешается использовать только в строковых триггерах. Если указать какую-либо из них в операторном триггере, будет выдана ошибка компиляции.

Поскольку операторный триггер выполняется лишь однажды (даже в том случае, когда в операторе обрабатывается несколько строк), псевдозаписи *:old* и *:new* не имеют никакого смысла.

Конструкция REFERENCING

При желании можно воспользоваться конструкцией REFERENCING и указать другие имена для *:old* и *:new*. Эта конструкция размещается после активизирующего события, перед условием WHEN:

REFERENCING [OLD AS *старое_имя*] [NEW AS *новое_имя*]

В теле триггера вместо *:old* и *:new* можно использовать *:старое_имя* и *:новое_имя.* Отметим, что в предложении REFERENCING идентификаторы указываются без двоеточия.

Конструкция REFERENCING. Пример.

```
CREATE OR REPLACE TRIGGER GenerateStudentID
    BEFORE INSERT OR UPDATE ON students
    REFERENCING new AS new _student
    FOR EACH ROW
BEGIN
    /* Заполним поле ID таблицы students
следующим     значением из student_sequence.
Поскольку ID - это столбец  таблицы students,
:new_student.ID является допустимой  ссылкой.
*/
    SELECT student_sequence.NEXTVAL
    INTO :new_student.ID
    FROM dual;
END GenerateStudentID;
```

Предложение WHEN

Предложение WHEN можно использовать только для строковых триггеров. При наличии WHEN тело триггера будет выполняться только для тех строк, которые соответствуют условию, указанному в WHEN. Общий вид предложения WHEN таков

WHEN *условие_триггера*

где *условие_триггера* является логическим выражением, которое проверяется для каждой строки. В условии можно ссылаться на записи *:new* и *:old*, но двоеточие в данном случае не применяется. Двоеточие можно указывать только в теле триггера.

Предложение WHEN. Пример.

```
CREATE OR REPLACE TRIGGER CheckCredits
        BEFORE INSERT OR UPDATE OF current_credits ON students
        FOR EACH ROW
        WHEN (new.current_credits> 20)
BEGIN
        /* Тело триггера */
END;
```

Триггер CheckCredits можно также написать следующим о

```
CREATE OR REPLACE TRIGGER CheckCredits
        BEFORE INSERT OR UPDATE OF current_credits ON Students
        FOR EACH ROW
BEGIN
        IF :new.current_credits > 20 THEN
        /* Тело триггера */
        END IF;
END;
```

Триггерные предикаты: INSERTING, UPDATING и DELETI

Приведенный выше триггер UpdateMajorStats является триггером INSERT, UPDATE и DELETE. Внутри триггера такого типа (который срабатывает на различные виды операторов DML) можно использовать три логические функции, определяющие тип выполняемой операции. Это логические функции (предикаты) INSERTING, UPDATING и DELETING. Их работа описывается в таблице ниже.

Предикат	**Принимаемое значение**
INSERTING	TRUE, если активизирующий оператор INSERT; FALSE в противном случае.
UPDATING	TRUE, если активизирующий оператор UPDATE; FALSE в противном случае.
DELETING	TRUE, если активизирующий оператор DELETE; FALSE в противном случае.

Триггерные предикаты. Пример.

```
CREATE TABLE RS_audit (
        change_type CHAR(1) NOT NULL,
        changed_by VARCHAR2(8) HOT NULL,
        timestamp DATE NOT NULL,
        old_student_id NUMBER(5),
        old_department CHAR(3),
        old_course NUMBER(3),
        old_grade CHAR(1),
        new_student_id NUMBER(5),
        new_department CHAR(3),
        new_course NUMBER(3),
        new_grade CHAR(1)
);
```

Триггерные предикаты. Пример (продолжение).

```
CREATE OR REPLACE TRIGGER LogRSChanges
        BEFORE INSERT OR DELETE OR UPDATE ON registered_students
        FOR EACH ROW
DECLARE
        v_ChangeType CHAR(1);
BEGIN
        /* Используем 'I' для INSERT, '0' для DELETE и 'IT для UPDATE. */
        IF INSERTING THEN
                v_ChangeType : = ' I';
        ELSIF UPDATING THEN
                v_ChangeType := ' U ' ;
        ELSE
                v_ChangeType := ' D ' ;
        END IF;
        /* Запишем в таблицу RS_audit все изменения, внесенные в таблицу
        registered_students. Для генерирования временной метки
        воспользуемся функцией SYSDATE, а для получения идентификатора
        текущего пользователя - функцией USER. */
        INSERT INTO RS_audit
                (changeType, changed_by, timestamp,
                old_student_id, old_department, old_course, old_grade,
                new_student_id, new_department, new_course, new_grade)
        VALUES
                (v_ChangeType, USER, SYSDATE,
                :old.student_id, :old.department, :old. course, :old.grade,
                :new. student_id, :new.department, :new. course, :new.grade);
END LogRSChanges;
```

Создание триггера DML. Пример 2.

Пример иллюстрирует прозрачную журнализацию событий. Триггер базы данных с именем reorder обеспечивает, что товар заказывается заново каждый раз, когда его имеющееся на складе количество (qty_on_hand) падает ниже пороговой точки.

```
CREATE TRIGGER reorder
   /* событие триггера */
   AFTER UPDATE OF qty_on_hand ON inventory  -- таблица
   FOR EACH ROW
   /* ограничение триггера */
   WHEN (new.reorderable = 'T')
BEGIN
   /* действие триггера */
   IF :new.qty_on_hand < :new.reorder_point THEN
      INSERT INTO pending_orders
      VALUES (:new.part_no, :new.reorder_qty, SYSDATE);
   END IF;
END;
```

Создание триггера. Пример 2. (продолжение)

- Имя во фразе ON (в примере, inventory) идентифицирует таблицу базы данных, ассоциированную с триггером базы данных.
- Событие триггера специфицирует предложение манипулирования данными SQL, которое воздействует на таблицу. В данном случае это предложение UPDATE. Если предложение триггера сбивается, оно откатывается.
- По умолчанию, триггер базы данных возбуждается один раз на всю таблицу. Необязательная фраза FOR EACH ROW указывает, что триггер должен возбуждаться один раз на каждую строку.
- Для того, чтобы триггер возбудился, однако, требуется, чтобы булевское выражение в фразе WHEN давало значение TRUE.
- Ключевое слово AFTER указывает, что триггер базы данных возбуждается после того, как обновление выполнено.
- Префикс :new представляет собой коррелирующее имя, которое отсылает к вновь измененному значению столбца. Внутри триггера базы данных вы можете обращаться как к новому, так и к старому (:old) значениям столбцов в измененных строках. Заметьте, что в фразе WHEN двоеточие не используется. Вы можете использовать фразу REFERENCING (здесь не показана), чтобы заменить :new и :old другими коррелирующими именами.

Создание триггера DML. Пример 3.

Как показывает следующий пример, действие триггера может включать вызовы встроенной процедуры ORACLE с именем raise_application_error, которая позволяет выдавать определенные пользователем сообщения об ошибках:

```
CREATE TRIGGER check_salary
    BEFORE INSERT OR UPDATE OF sal, job ON emp
    FOR EACH ROW
    WHEN (new.job != 'PRESIDENT')
DECLARE
    minsal  NUMBER;
    maxsal  NUMBER;
BEGIN
/* Дать интервал окладов для данной должности из справочника */
   SELECT losal, hisal INTO minsal, maxsal FROM sals
   WHERE job = :new.job;
   /* Если оклад вне диапазона, прибавка отрицательна, *
   /* или прибавка выше 10%, возбудить исключение.     */
   IF (:new.sal < minsal OR :new.sal > maxsal) THEN
       raise_application_error(-20225, 'Salary out of range');
    ELSIF (:new.sal < :old.sal) THEN
       raise_application_error(-20230, 'Negative increase');
    ELSIF (:new.sal > 1.1 * :old.sal) THEN
       raise_application_error(-20235, 'Increase exceeds 10%');
    END IF;
END;
```

Задания

1. Для отслеживания изменений информации о сотрудниках, создайте таблицу s_emp_log и напишите триггер, заносящий в s_emp_log информацию о времени изменения и информации о сотруднике до изменения.
2. Для отслеживания удалений информации о сотрудниках, напишите триггер, заносящий в s_emp_log удалённую информацию, а также время удаления.
3. Напишите триггер для таблице s_item, позволяющий контролировать изменение цены товара. Если новая цена отличается от старой более чем на 30%, выдаётся соответствующее сообщение и запрещается изменения данных. Реализовать используя исключения.
4. Требуется отслеживать статистические показатели, касающиеся продуктов. Т.е. для каждого существующего товара указывается количество заказчиков этого товара, количество заказанных единиц, сумма заказа. Результаты будут храниться в таблице major_stats.

Создание замещающих триггеров баз данных

Понятие триггера

ТРИГГЕР БАЗЫ ДАННЫХ - это хранимая программная единица PL/SQL, ассоциированная с конкретной таблицей базы данных.

В отличие от подпрограмм, которые должны вызываться явно, триггер базы данных вызывается неявно и не имеет атрибутов.

Акт выполнения триггера называется его *активизацией* (firing). Событием, запускающим триггер, является операция DML (INSERT, UPDATE или DELETE),выполняемая над таблицей или представлением базы данных. В Oracle8i эти функции расширены: триггер может срабатывать на системное событие, например на запуск или останов базы данных, а также на определенные виды операций DDL.

Типы триггеров

Триггер DML активизируется оператором DML, и тип триггера определяется типом этого оператора. Триггеры DML задаются для операций ввода, обновления и удаления информации (INSERT, UPDATE, DELETE).Они активизируются до или после операции, на уровне строки или оператора.

В Oracle8i предлагается еще один вид триггеров. *Триггеры замещения* (instead of) можно создавать только для представлений (либо объектных, либо реляционных). В отличие от триггеров DML, которые выполняются в дополнение к операторам DML, триггеры замещения выполняются вместо операторов DML, вызывающих их срабатывание. Триггеры замещения должны быть строковыми триггерами.

В Orac1e8i и выше существует третий тип триггеров. *Системный триггер* активизируется не на операцию DML, выполняемую над таблицей, а на системное событие, например, на запуск или останов базы данных. Системные триггеры срабатывают и на операции DDL, такие как создание таблицы.

Привилегии для создания триггера

Для создания триггера базы данных необходимо иметь привилегии CREATE TRIGGER, а также либо владеть ассоциированной таблицей, либо иметь привилегии ALTER для ассоциированной таблицы, либо иметь привилегии ALTER ANY TABLE.

Триггер базы данных состоит из трех частей: события триггера, необязательного ограничения триггера и действия триггера. Когда происходит событие триггера, триггер базы данных возбуждается, и анонимный блок PL/SQL выполняет предписанное действие. Триггеры базы данных возбуждаются с привилегиями владельца, а не текущего пользователя. Поэтому владелец должен иметь должный доступ ко всем объектам, вовлекаемым в действие триггера.

Создание замещающих триггеров

Триггеры замещения (insteadof) можно создавать только для представлений. В отличие от триггеров DML, которые выполняются в дополнение к операторам DML (или до, или после них), триггеры замещения выполняются вместо операторов DML, вызывающих их срабатывание. Триггеры замещения должны быть строковыми триггерами.

Замещающие триггеры используются в двух случаях:

- Для того чтобы сделать представление модифицируемым, если иначе это сделать нельзя.
- Для модификации столбцов в столбце вложенной таблицы представления.

Модифицируемые и немодифицируемые представления

Модифицируемым (modifiable) называется такое представление, по отношению к которому можно выполнить оператор DML. Как правило, представление является модифицируемым, если оно не содержит:

- Операций над множествами (UNION, UNION ALL, MINUS)
- Функций агрегирования (SUM, AVG и т.д.)
- Конструкций GROUP BY, CONNECT BY и START WITH
- Операции DISTINCT
- Соединений

Модифицируемые и немодифицируемые представления

Существуют представления, которые содержат соединения и при этом являются модифицируемыми. Обычно представление с соединением может быть модифицировано, если операция DML, выполняемая над ним, одновременно модифицирует только одну базовую таблицу и если оператор DML отвечает слудующим условиям:

INSERT - Оператор не ссылается, явно или неявно, на столбцы таблицы, не сохраняющей ключи.

UPDATE - Обновляемые столбцы отображаются на столбцы таблицы, сохраняющей ключи.

DELETE – В соединении только одна таблица, сохраняющая ключи

Модифицируемые и немодифицируемые представления

Если представление является немодифицируемым, то для него можно создать замещающий триггер, выполняющий нужные действия и тем самым разрешающий его модификацию.

Замещающий триггер можно создать и для модифицируемого представления, если требуется дополнительная обработка информации.

Пример замещающего триггера

Создадим представление classes_rooms :

```
CREATE OR REPLACE VIEW classes_rooms AS
SELECT department, course, building, room_number
FROM rooms, classes
WHERE rooms. room_id = classes.room_id;
```

Ввести информацию в это представление нельзя. Над ним можно выполнить операции обновления или удаления данных, но, скорее всего, эти действия будут некорректны.

Например, в результате выполнения над classes_rooms операции DELETE будут удалены соответствующие строки из classes.

Пример замещающего триггера

Создадим триггер замещения и с его помощью выполним обновление базовых таблиц:

```
CREATE TRIGGER ClassesRoomsInsert
    INSTEAD OF INSERT ON classes_rooms
DECLARE
    v_roomID rooms.room_id%TYPE;
BEGIN
        -- Сначала определим идентификатор аудитории.
SELECT room_id
   INTO v_roomID
   FROM rooms
   WHERE building = :new.building
   AND room_number = :new.room_number;
```

Пример замещающего триггера

```
-- А теперь обновим группу.
   UPDATE CLASSES
       SET room id = v roomlD
       WHERE department = :new.department
       AND course = :new.course;

END ClassesRoomsInsert;
```

С помощью триггера ClassesRoomsInsert оператор INSERT выполняется успешно и делает именно то, что нужно.

Задание

Создайте представление, содержащее имя отдела, номер региона, Фамилию, имя, должность и номер сотрудника.

Операции DML для данного представления определим по следующим правилам:

INSERT – Назначить отдел, введенному сотруднику. В результате s_dept (при необходимости s_region) обновляются.

UPDATE – Изменить отдел, назначенный сотруднику. Это может привести к обновлению или s_emp или s_dept, в зависимости от того, какой столбец представления обновляется.

DELETE – Очистить идентификатор отдела для сотрудника (сотрудник не зачислен ни в один отдел). В результате s_emp обновляется: ID устанавливается в значении NULL.

Создайте триггер, реализующий сформулированные выше правила и позволяющий правильно выполнять операции DML над созданным представлением.

Входные данные процедуры – ФИО, должность сотрудника, название отдела. Заработную плату сотруднику определить как среднюю з/п всех сотрудников фирмы.