

СУЧАСНІ СУБД

Лекція №9

Тема:

ORACLE PL/SQL

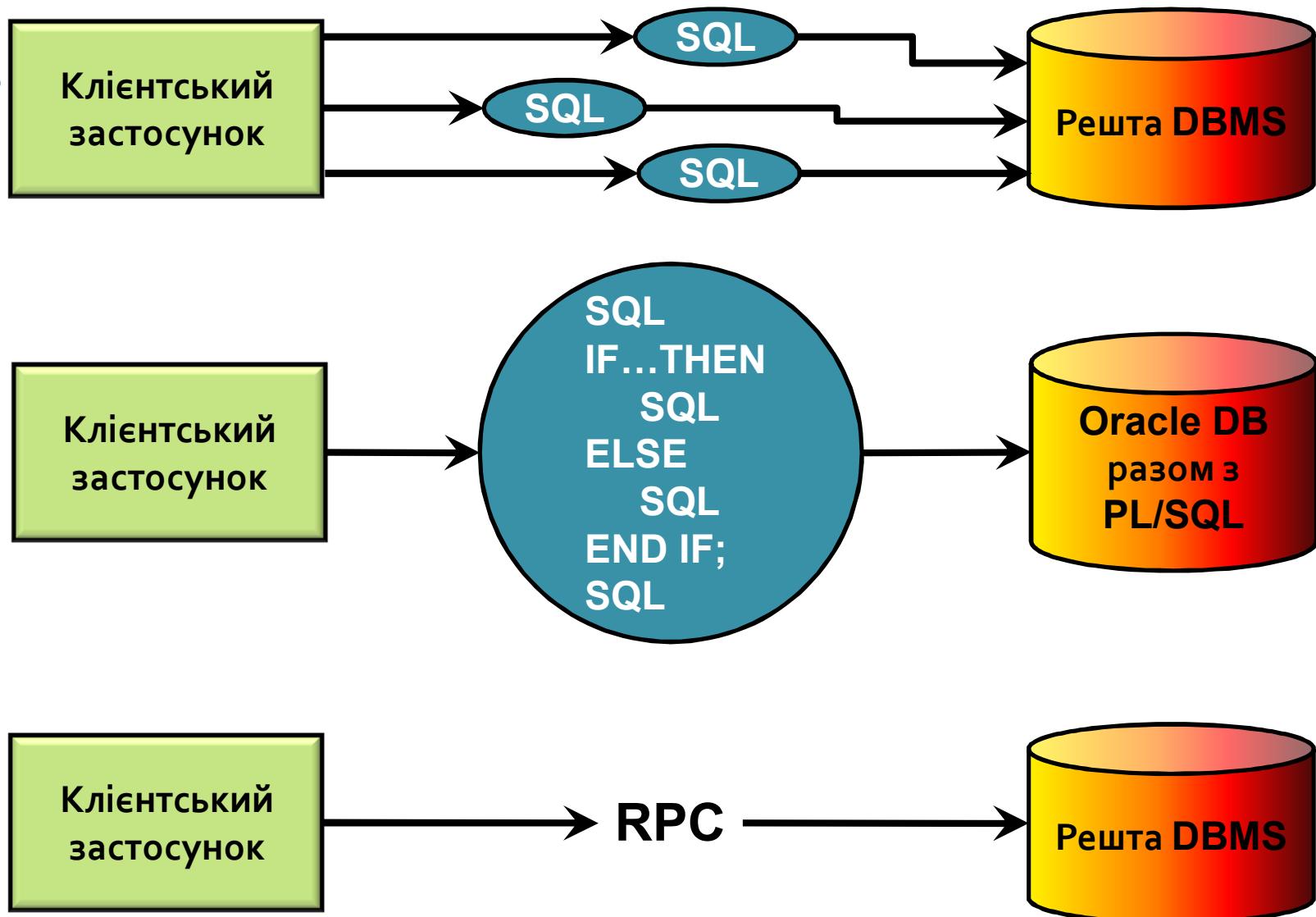
Процедурне розширення.

Огляд.

Переваги PL/SQL

№	Властивість
1.	Щільна інтеграція з SQL
2.	Висока продуктивність
3.	Висока швидкодія
4.	Повна переносимість
5.	Висока захищеність
6.	Використання попередньо визначених пакетів
7.	Підтримка об'єктно-орієнтованого програмування
8.	Підтримка розробки веб-додатків та серверів

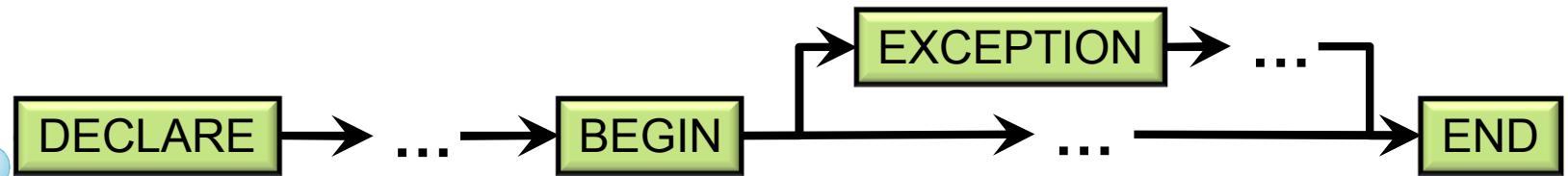
Висока продуктивність PL/SQL



Основні особливості PL/SQL.

№	Предмет
01.	Блоки PL/SQL
02.	Обробка помилок PL/SQL
03.	PL/SQL введення/виведення
04.	Змінні та константи PL/SQL
05.	Абстракція даних PL/SQL
06.	PL/SQL Структури управління
07.	Підпрограми PL/SQL
08.	Пакети PL/SQL (API, записані в PL/SQL)
09.	Умовна компіляція
10.	Вбудовані SQL- вирази

Блок PL/SQL.



DECLARE -- Декларативна частина (необов'язково)

-- Декларації локальних типів, змінних та підпрограм

BEGIN -- Виконавча частина (обов'язково)

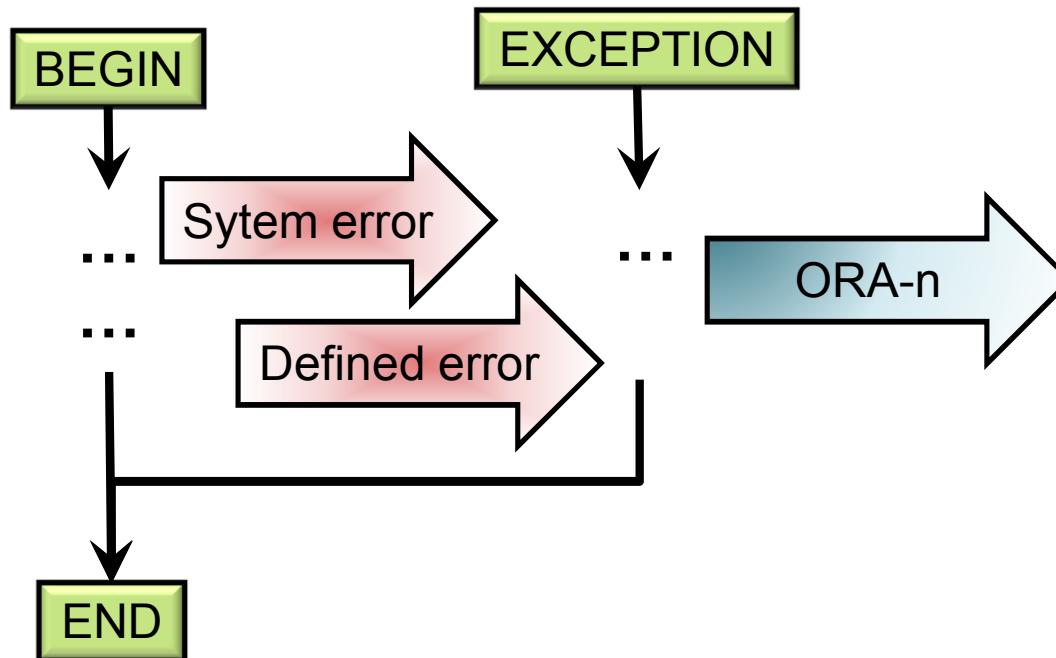
-- Вирази (в яких загалом можуть використовуватися об'єкти, описані в декларативній частині)

[**EXCEPTION** -- Exception-handling part (optional)]

-- Exception handlers for exceptions raised in executable part]

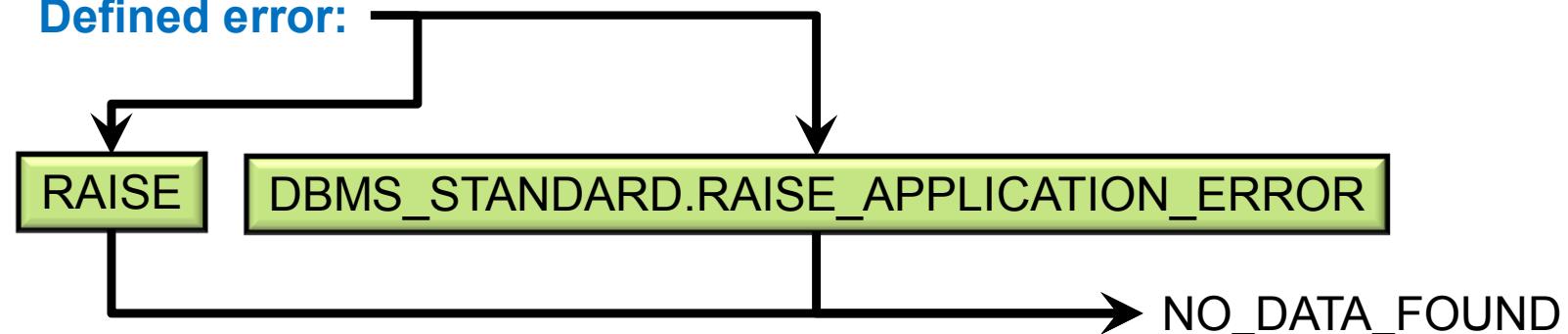
END;

Обробка помилок PL/SQL.



Sytem error: ZERO_DIVIDE

Defined error:



Обробка помилок PL/SQL. Приклади.

```
CREATE OR REPLACE PROCEDURE award_bonus ( emp_id NUMBER, bonus
NUMBER)
AS
    commission  REAL;
    comm_missing EXCEPTION;
BEGIN
    SELECT commission_pct/100 INTO commission
    FROM employees
    WHERE employee_id = emp_id;
    IF commission IS NULL
    THEN
        RAISE comm_missing;
    ELSE
        UPDATE employees
        SET salary = salary + bonus*commission
        WHERE employee_id = emp_id;
    END IF;
EXCEPTION
    WHEN comm_missing THEN
        DBMS_OUTPUT.PUT_LINE ('This employee does not receive a commission.');
        commission := 0;
    WHEN OTHERS THEN
        NULL;
END award_bonus;
/
```

PL/SQL введення/виведення.

№	Пакет(и)	PL/SQL використовує пакет ...
1.	HTF та HTP	для відображення результатів на веб-сторінці
2.	DBMS_OUTPUT	для виводу інформації в інші програми
3.	DBMS_PIPE	для передачі інформації між PL/SQL та командами операційної системи
4.	UTL_FILE	для читання та запису файлів операційної системи
5.	UTL_HTTP	для спілкування з веб-серверами
6.	UTL_SMTP	для спілкування з поштовими серверами

Змінні та константи PL/SQL.

№	Дія
1.	Декларування змінних PL/SQL
2.	Приєднання значень змінним
3.	Декларування PL/SQL констант
4.	Прив'язка змінних

Декларування змінних PL/SQL. Приклади.

DECLARE

```

part_number      NUMBER(6);      -- SQL data type
part_name        VARCHAR2(20);   -- SQL data type
in_stock          BOOLEAN;       -- PL/SQL-only data type
part_price        NUMBER(6,2);    -- SQL data type
part_description  VARCHAR2(50);   -- SQL data type
emp_rec1          employees%ROWTYPE;
                           TYPE commissions
IS TABLE OF NUMBER INDEX BY PLS_INTEGER;
comm_tab          commissions;

BEGIN NULL; END;
/

```

DECLARE

```

hours_worked     NUMBER := 40;
hourly_salary    NUMBER := 22.50;
part_name         VARCHAR2(20) := 'Part_name_1';
credit_limit      CONSTANT NUMBER := 5000.00;
in_stock          BOOLEAN := TRUE;

```

BEGIN NULL; **END**;

/

Абстракція даних PL/SQL

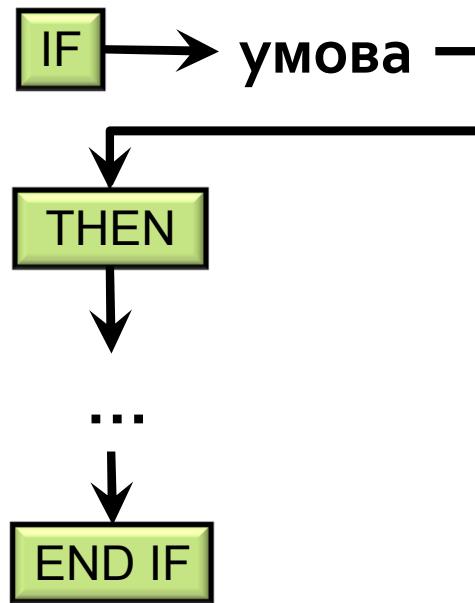
№	Тип
1.	Cursors
2.	%TYPE Attribute
3.	%ROWTYPE Attribute
4.	Collections
6.	Records
7.	Object Types

Абстракція даних PL/SQL. Приклади.

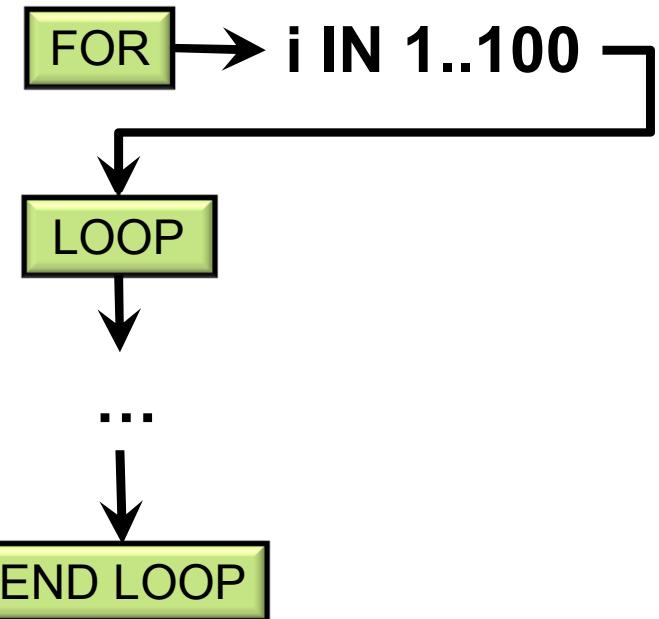
```
DECLARE
    CURSOR c1 IS
    (
        SELECT last_name, salary, hire_date, job_id
        FROM employees
        WHERE employee_id = 120
    );
    employee_rec c1%ROWTYPE;
    v_last_name employees.last_name%TYPE;
    TYPE timerec IS RECORD (hours SMALLINT, minutes SMALLINT);
    TYPE staff_list IS TABLE OF employees.employee_id%TYPE;
    staff staff_list;
BEGIN
    OPEN c1;
    FETCH c1 INTO employee_rec;
    staff := staff_list(100, 114, 115, 120, 122);
END;
/
```

PL/SQL Структури управління.

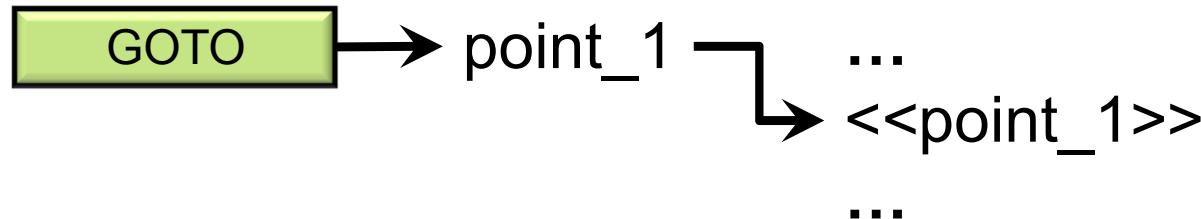
Умовні:



Ітеративні:



Послідовні:



Підпрограми PL/SQL. Приклад.

```
DECLARE
    in_string  VARCHAR2(100) := 'Test string';
    out_string VARCHAR2(200);
    PROCEDURE double
    (
        original      IN VARCHAR2,
        new_string    OUT VARCHAR2
    ) AS
    BEGIN
        new_string := original || original;  END;
    BEGIN
        DBMS_OUTPUT.PUT_LINE ('in_string: ' || in_string);
        double (in_string, out_string);
        DBMS_OUTPUT.PUT_LINE ('out_string: ' || out_string);
    END;
/
```

Пакети PL/SQL (API, записані в PL/SQL).

```
CREATE OR REPLACE PACKAGE emp_actions AS
  PROCEDURE hire_employee
  (
    employee_id      NUMBER,
    last_name        VARCHAR2,
    first_name       VARCHAR2,
    hire_date        DATE,
    job_id           VARCHAR2,
    salary            NUMBER,
    commission_pct   NUMBER,
    department_id    NUMBER
  );
  PROCEDURE fire_employee (emp_id NUMBER);
  FUNCTION num_above_salary (emp_id NUMBER) RETURN NUMBER;
END emp_actions;
/
```

Пакети PL/SQL (API, записані в PL/SQL).

```
CREATE OR REPLACE PACKAGE BODY emp_actions AS
  PROCEDURE hire_employee
  (
    employee_id      NUMBER,
    last_name        VARCHAR2,
    first_name       VARCHAR2,
    hire_date        DATE,
    job_id           VARCHAR2,
    salary            NUMBER,
    commission_pct   NUMBER,
    department_id    NUMBER
  ) IS BEGIN NULL; END hire_employee;
  PROCEDURE fire_employee (emp_id NUMBER) IS
  BEGIN NULL; END fire_employee;
  FUNCTION num_above_salary (emp_id NUMBER)
  RETURN NUMBER IS
  BEGIN NULL; RETURN 0; END num_above_salary ;
  PROCEDURE in_body_proc (emp_id NUMBER) IS
  BEGIN NULL; END in_body_proc;
END emp_actions;
/
```

СУЧАСНІ СУБД

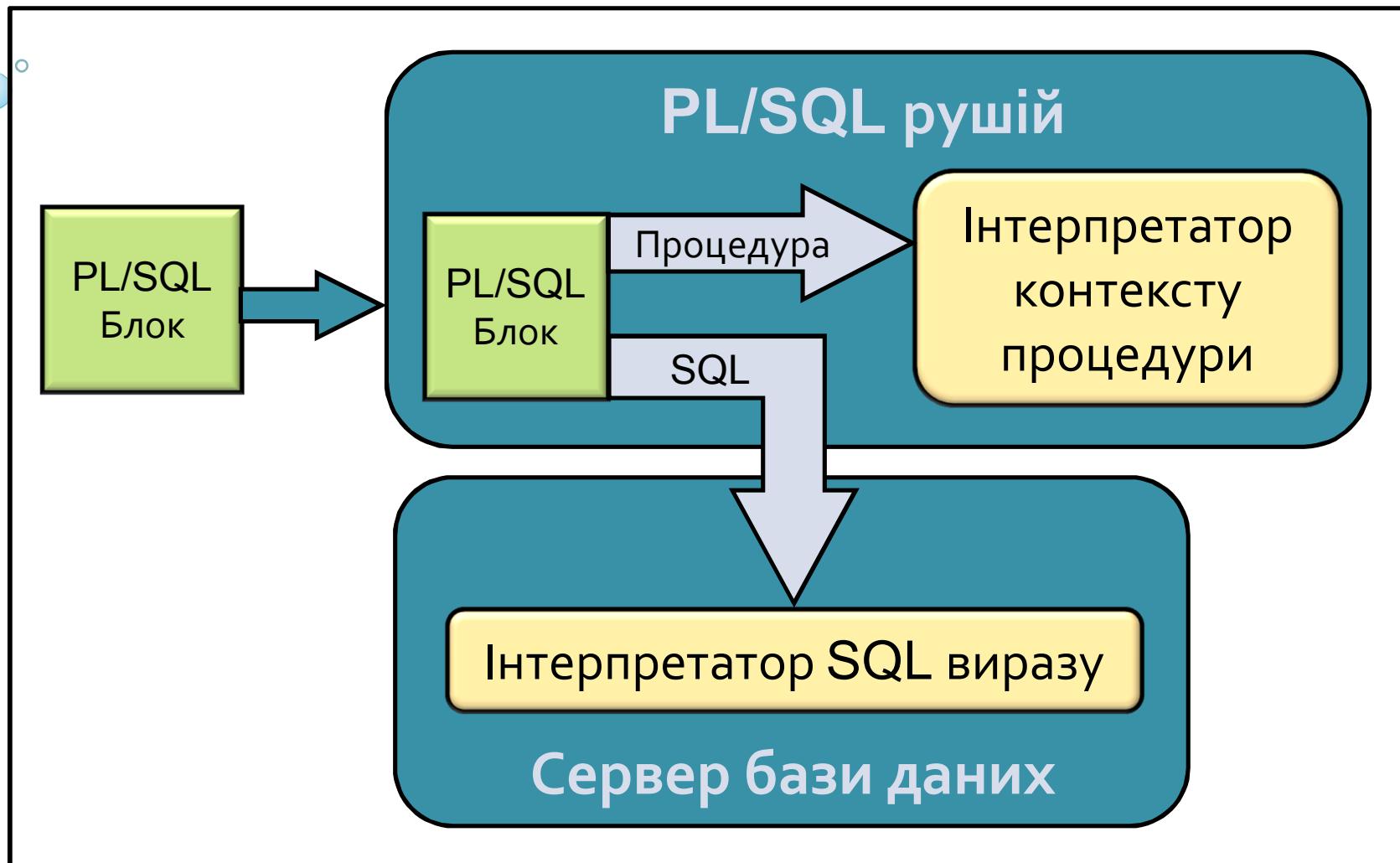
Лекція №10

Тема:

ORACLE PL/SQL

Основи мови.

Архітектура PL/SQL



PL/SQL одиниці

№	Одинаця компіляції
1.	PL/SQL block
2.	FUNCTION
3.	PACKAGE
4.	PACKAGE BODY
6.	PROCEDURE
7.	TRIGGER
8.	TYPE
9.	TYPE BODY

Набори символів та лексичні одиниці.

Назва	Приклад
Великі та малі літери	A .. Z and a .. z
Цифри	0 .. 9
Символи	() + - * / < > = ! ~ ^ ; : . ' @ % , " # \$ & _ { } ? [], \t, \n, „

Лексичні одиниці

1. Роздільники (прості і складні символи)
2. Ключові та зарезервовані слова
3. Ідентифікатори
4. Літерали
5. Коментарі

Набори символів та лексичні одиниці.

Неохідно відокремлювати сусідні ідентифікатори пробілом або пунктуацією.

```
SQL> BEGIN
```

```
2      IF x > y THEN high := x; END IF; -- correct
```

```
3      IF x > y THEN high := x; ENDIF; -- incorrect
```

```
4      END;
```

```
5      /
```

```
    END;
```

```
*
```

ERROR at line 4:

ORA-06550: line 4, column 4:

PLS-00103: Encountered the symbol ";" when expecting one of the following:

if

```
SQL>
```

Роздільники PL/SQL.

Симв.	Значення	Симв.	Значення
+	оператор додавання	:=	оператор присвоєння
%	індикатор атрибута	=>	оператор асоціації
'	роздільник символів для рядків		оператор конкатенації
.	перемикач компонентів	**	оператор експоненції
/	оператор відділу	<<	роздільник мітки (початок)
(виразник або роздільник списку	>>	роздільник мітки (кінець)
)	виразник або роздільник списку	/*	багаторядковий коментар (початок)
:	індикатор хостової змінної	*/	багаторядковий коментар (кінець)
,	роздільник елементів	..	оператор дальності
*	оператор множенн	<>	оператор відношення
"	цитований ідентифікатор-роздільник	!=	оператор відношення
=	оператор відношення	~=	оператор відношення
<	оператор відношення	^=	оператор відношення
>	оператор відношення	<=	оператор відношення
@	індикатор віддаленого доступу	>=	оператор відношення
;	термінатор оператора	--	однорядковий коментар
-	оператор віднімання / заперечення		

Роздільники PL/SQL.

В складних роздільниках пробіли недопустимі.

```
SQL> BEGIN
```

```
 2      count := count + 1; -- correct
 3      count : = count + 1; -- incorrect
 4  END;
 5 /
count : = count + 1; -- incorrect
*
```

ERROR at line 3:

ORA-06550: line 3, column 9:

PLS-00103: Encountered the symbol ":" when expecting one of the
following:

`:= . (@ % ;`

```
SQL>
```

Літерали PL/SQL.

№	Клас
1.	Числові літерали
2.	Символьні літерали
3.	Строкові літерали
4.	Булеві літерали
5.	Літерали дати та часу

Числові літерали PL/SQL.Приклади.

6.6667 0.0 -12.0 3.14159 +8300.00 .5 25.

2E5 1.0E-7 3.14159e0 -1E38 -9.5e-3

$5\text{E}3 = 5 * 10^{**3} = 5 * 1000 = 5000$ $5\text{E}-3 = 5 * 10^{**-3} = 5 * 0.001 = 0.005$

```
SQL> DECLARE
  2      n NUMBER;
  3  BEGIN
  4      n := -9.999999E-130;
  5      n := 9.999E125;
  6      n := 10.0E125;
  7  END;
  8 /
n := 10.0E125;
*
```

ERROR at line 6:

ORA-06550: line 6, column 8:

PLS-00569: numeric overflow or underflow

ORA-06550: line 6, column 3:

PL/SQL: Statement ignored

SQL>

Числові літерали PL/SQL.Приклади.

```
SQL> DECLARE
 2      x BINARY_FLOAT := sqrt(2.0f);
 3      -- single-precision floating-point number
 4      y BINARY_DOUBLE := sqrt(2.0d);
 5      -- double-precision floating-point number
 6      BEGIN
 7          NULL;
 8      END;
 9      /
```

PL/SQL procedure successfully completed.

```
SQL>
```

Символьні та строкові літерали PL/SQL

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

- 'Hello, world!'
- 'XYZ Corporation'
- '10-NOV-91'
- 'He said "Life is like licking honey from a thorn."'
- '\$1,000,000'

'I"m a string, you"re a string.'

-- q'!...!' notation allows use of single quotes inside literal
 string_var := q'!!'I'm a string, you're a string.!';

```
func_call
(
  q'[SELECT index_name FROM user_indexes WHERE status ='INVALID']
);
```

where_clause := nq'#**WHERE** col_value **LIKE** '%\00E9#';

Літерали дати та часу PL/SQL.

```
SQL> DECLARE
  2      d1 DATE      := DATE '1998-12-25';
  3      t1 TIMESTAMP := TIMESTAMP '1997-10-22 13:01:01';
  5      t2 TIMESTAMP WITH TIME ZONE :=
  6          TIMESTAMP '1997-01-31 09:26:56.66 +02:00';
  7
  8  -- Three years and two months
  9  -- For greater precision, use the day-to-second interval
 10
 11     i1 INTERVAL YEAR TO MONTH := INTERVAL '3-2' YEAR TO MONTH;
 12
 13  -- Five days, four hours, three minutes, two and 1/100 seconds
 14
 15     i2 INTERVAL DAY TO SECOND :=
 16                  INTERVAL '5 04:03:02.01' DAY TO SECOND;
 17 BEGIN
 18     NULL;
 19 END;
 20 /
```

PL/SQL procedure successfully completed.

```
SQL>
```

Приєднання значень змінним PL/SQL.

bonus := 8300.00;

bonus := salary * 0.15;

```
SQL> DECLARE
  2      counter INTEGER;
  3  BEGIN
  4      counter := counter + 1;
  5
  6      IF counter IS NULL
  7          THEN
  8              DBMS_OUTPUT.PUT_LINE('counter is NULL.');
  9      END IF;
 10  END;
 11 /
counter is NULL.
```

PL/SQL procedure successfully completed.

SQL>

Приєднання булевих значень

```
SQL> DECLARE
  2      done  BOOLEAN;          -- Initialize to NULL by default
  3      counter NUMBER := 0;
  4  BEGIN
  5      done := FALSE;          -- Assign literal value
  6      WHILE done != TRUE    -- Compare to literal value
  7          LOOP
  8          counter := counter + 1;
  9          done := (counter > 500); -- Assign value of expression
 10     END LOOP;
 11  END;
 12 /
```

PL/SQL procedure successfully completed.

```
SQL>
```

Присвоєння результату SQL запита змінним PL/SQL

SQL> **DECLARE**

```
2      emp_id    employees.employee_id%TYPE := 100;
3      emp_name   employees.last_name%TYPE;
4      wages      NUMBER(7,2);
5  BEGIN
6      SELECT last_name, salary + (salary * nvl(commission_pct,0))
7      INTO emp_name, wages
8      FROM employees
9      WHERE employee_id = emp_id;
10
11     DBMS_OUTPUT.PUT_LINE
12             ('Employee ' || emp_name || ' might make ' || wages);
13 END;
14 /
```

Employee King might make 24000

PL/SQL procedure successfully completed.

SQL>

Вирази та порівняння PL / SQL.

№	Клас операторів та виразів
1.	Оператор конкатенації
2.	Логічні оператори
3.	Булеві вирази
4.	CASE вирази

Пріоритетність операторів		
Рівень	Оператор	Значення
1.	**	Експоненція
2.	+, -	Знак
3.	*, /	Множення, ділення
4.	+, -,	Додавання, віднімання, конкатенація
5.	=, <, >, <=, >=, <>, !=, ~=, ^=, IS NULL, LIKE, BETWEEN, IN	Порівняння
6.	NOT	Логічне заперечення
7.	AND	Кон'юнкція
8.	OR	Діз'юнкція

Обробка значення NULL.

x	y	x AND y	x OR y	NOT x
TRUE	TRUE	TRUE	TRUE	FALSE
TRUE	FALSE	FALSE	TRUE	FALSE
TRUE	NULL	NULL	TRUE	FALSE
FALSE	TRUE	FALSE	TRUE	TRUE
FALSE	FALSE	FALSE	FALSE	TRUE
FALSE	NULL	FALSE	NULL	TRUE
NULL	TRUE	NULL	TRUE	NULL
NULL	FALSE	FALSE	NULL	NULL
NULL	NULL	NULL	NULL	NULL

СУЧАСНІ СУБД

Лекція №11

Тема:

ORACLE PL/SQL

Структура програми

Керування послідовністю виконання програмного коду

Команди умовного розгалуження

До команди, що керують послідовністю виконання програмного коду в залежності від заданих умов, відносяться команди **IF-THEN-ELSE** і **CASE**. Також існують так звані **CASE**-вирази, які іноді дозволяють обійтися без команд **IF** і **CASE**.

Команди безумовного розгалуження

Команда безумовного переходу: **GOTO** і команда холостого ходу **NULL** (не виконує ніяких дій).

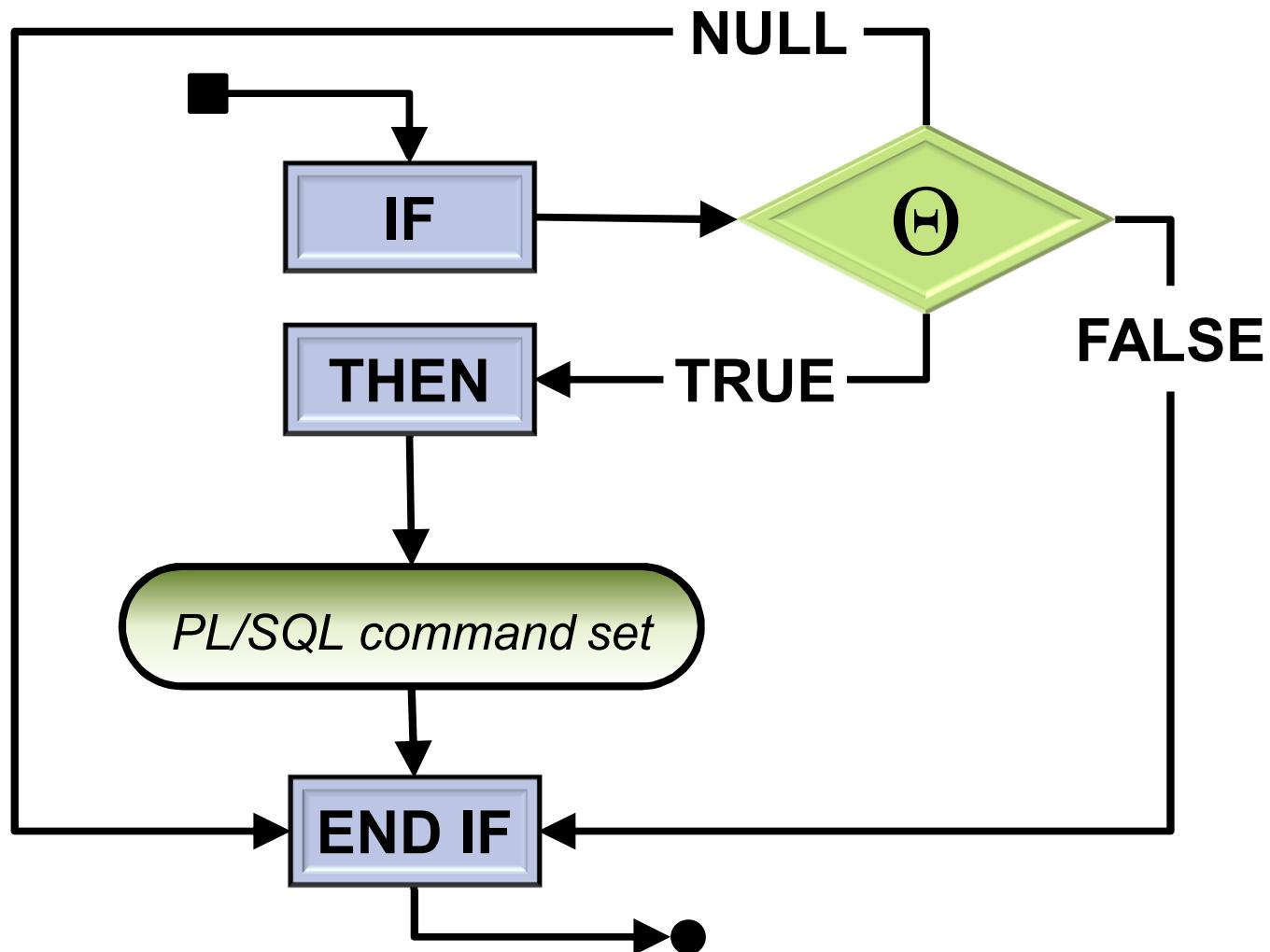
Команди ітеративного розгалуження

Структури що керують багаторазовим (ітеративним) виконання одного і того ж фрагмента програмного коду, називаються циклами. PL/SQL підтримує цикли трьох видів: прості (нескінченні), **FOR** і **WHILE**. Також в Oracle11g з'явилася команда **CONTINUE**.

Команди IF

№	Різновид IF	Характеристики
1.	IF THEN END IF;	<i>Найпростіша форма команди IF. Умова між IF та THEN визначає, чи повинна виконуватися група команд, що знаходитьться між THEN і END IF. Якщо результат перевірки умови дорівнює FALSE або NULL, то код не виконується</i>
2.	IF THEN ELSE END IF;	<i>Реалізація логіки «або-або». Залежно від умови між ключовими словами IF і THEN виконується або код, що знаходитьться між THEN і ELSE, або код між ELSE і END IF. У будь-якому випадку виконується тільки одна з двох груп виконуваних команд.</i>
3.	IF THEN ELSIF ELSE END IF;	<i>Остання, і найскладніша, форма IF вибирає дію з набору взаємовиключних умов і виконує відповідну групу виконуваних команд. У версії Oracle9i і вище її доцільніше замінити командою вибору CASE</i>

Комбінація IF-THEN



Комбінація IF-THEN. Приклад

IF salary > 40000 THEN give_bonus (employee_id,500); END IF;

IF

salary > 40000

THEN

INSERT INTO employee_bonus

 (

employee_id,

bonus_amt

)

VALUES (employee_id, 500);

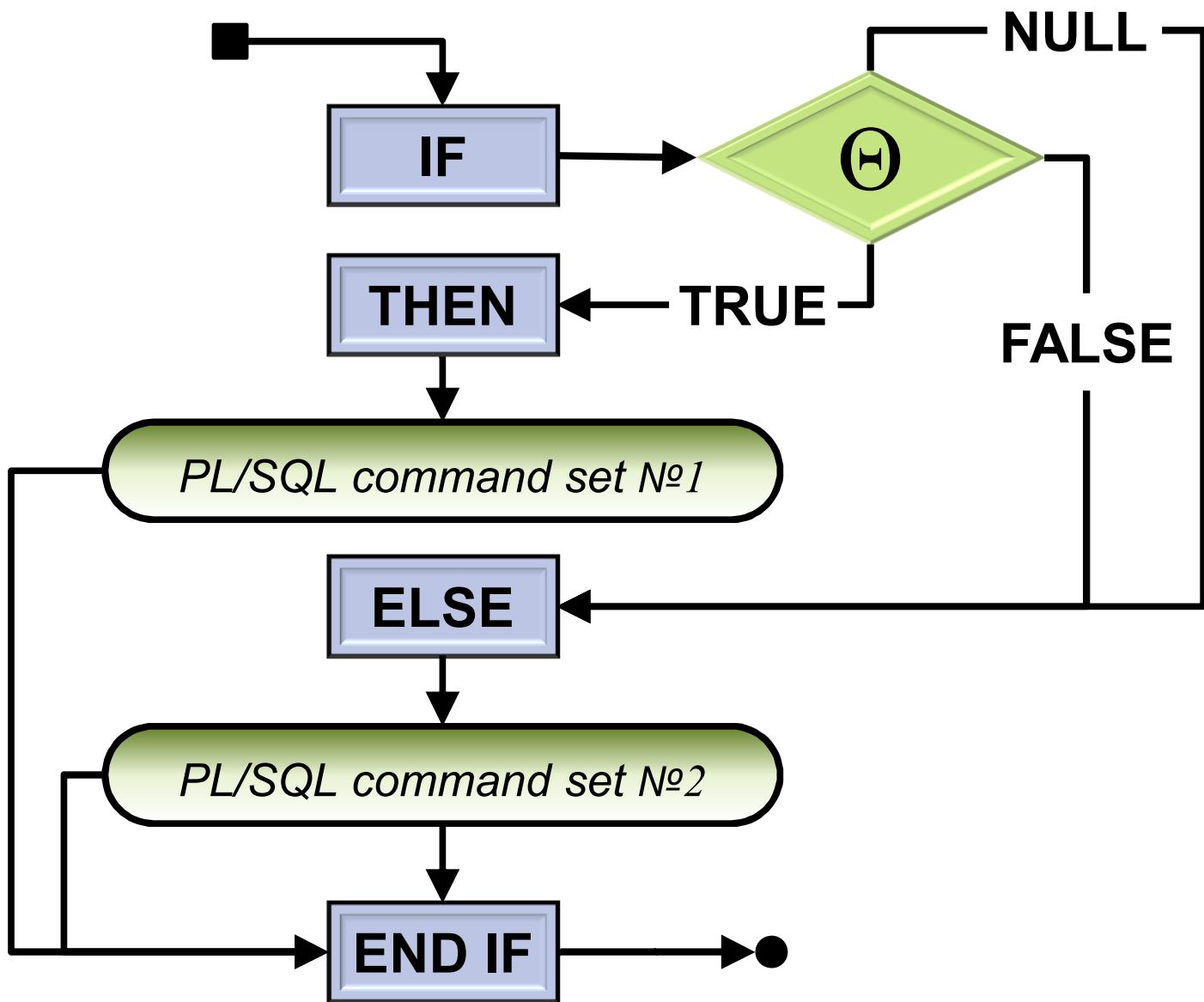
UPDATE emp_employee

SET emp_bonus_given=1

WHERE emp_employee_id=employee_id;

END IF;

Комбінація IF-THEN-ELSE



Комбінація IF-THEN-ELSE. Приклад.

```
IF  
    salary <= 40000  
THEN  
    give_bonus (employee_id, 0);  
ELSE  
    give_bonus (employee_id, 500);  
END IF;
```

```
IF  
    NVL(salary,0) <= 40000  
THEN  
    give_bonus (employee_id, 0);  
ELSE  
    give_bonus (employee_id, 500);  
END IF;
```

Логічні семафори.

IF

:customer.order_total > max_allowable_order

THEN

order_exceeds_balance := TRUE;

ELSE

order_exceeds_balance := FALSE;

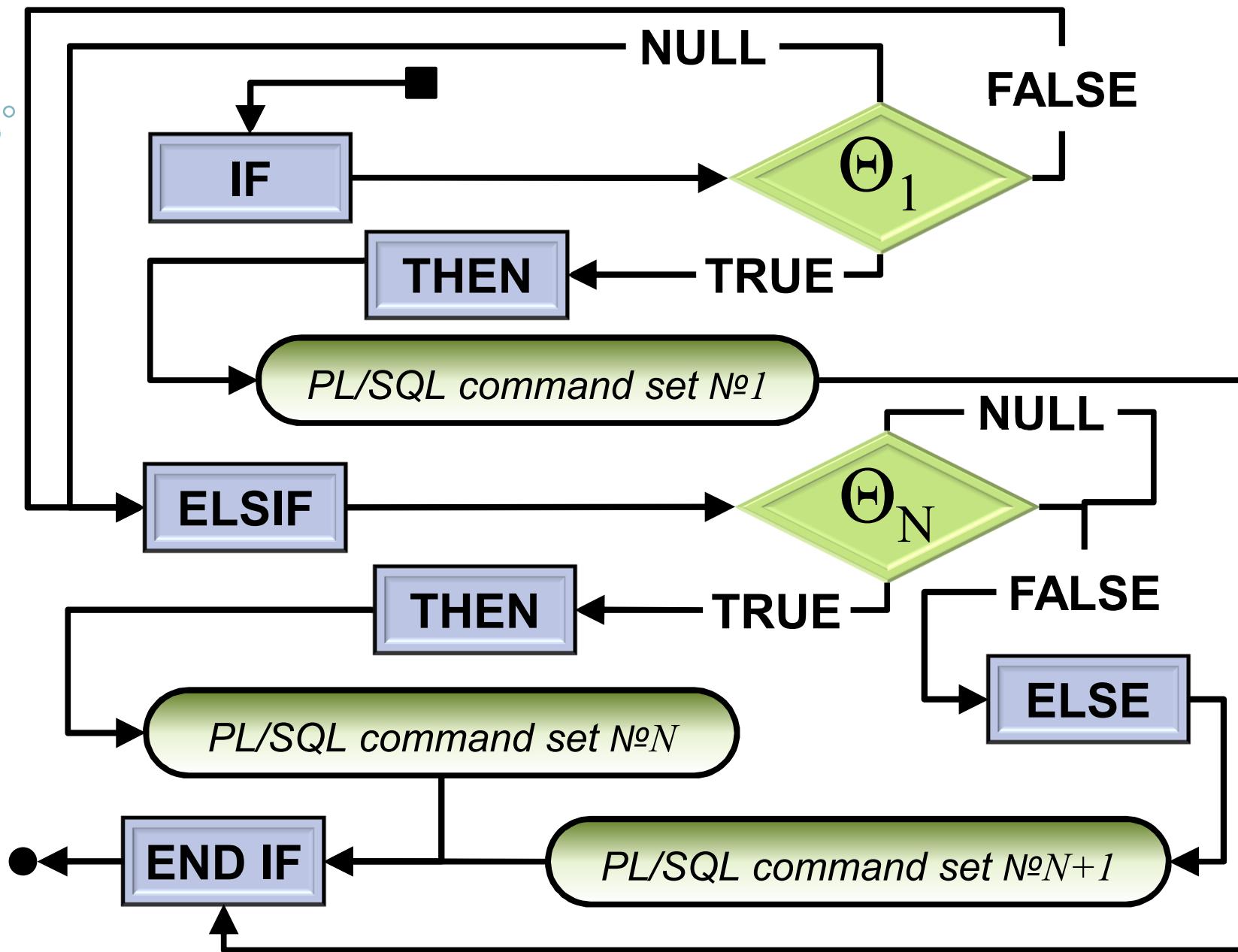
END IF;

order_exceeds_balance :=

(:customer.order_total > max_allowable_order);

IF order_exceeds_balance **THEN** ... **END IF;**

Комбінація IF-THEN-ELSIF-[ELSE]



Комбінація IF-THEN-ELSIF. Приклади

```

IF      salary BETWEEN 10000 AND 20000
THEN    give_bonus(employee_id, 1500);
ELSIF  salary BETWEEN 20000 AND 40000
THEN    give_bonus(employee_id, 1000);
ELSIF  salary > 40000
THEN    give_bonus(employee_id, 500);
ELSE    give_bonus(employee_id, 0);
END IF;

```

```

IF      salary  $\geq 10000$  AND salary  $\leq 20000$ 
THEN    give_bonus(employee_id, 1500);
ELSIF  salary > 20000 AND salary  $\leq 40000$ 
THEN    give_bonus(employee_id, 1000);
ELSIF  salary > 40000
THEN    give_bonus(employee_id, 400);
ELSE    give_bonus(employee_id, 0);
END IF;

```

Вкладені команди IF. Приклади

IF condition №1

THEN

IF condition №2

THEN

command set №2

ELSE

IF condition №3

THEN

command set №3

ELSIF condition №4

THEN

command set №

END IF;

END IF;

END IF;

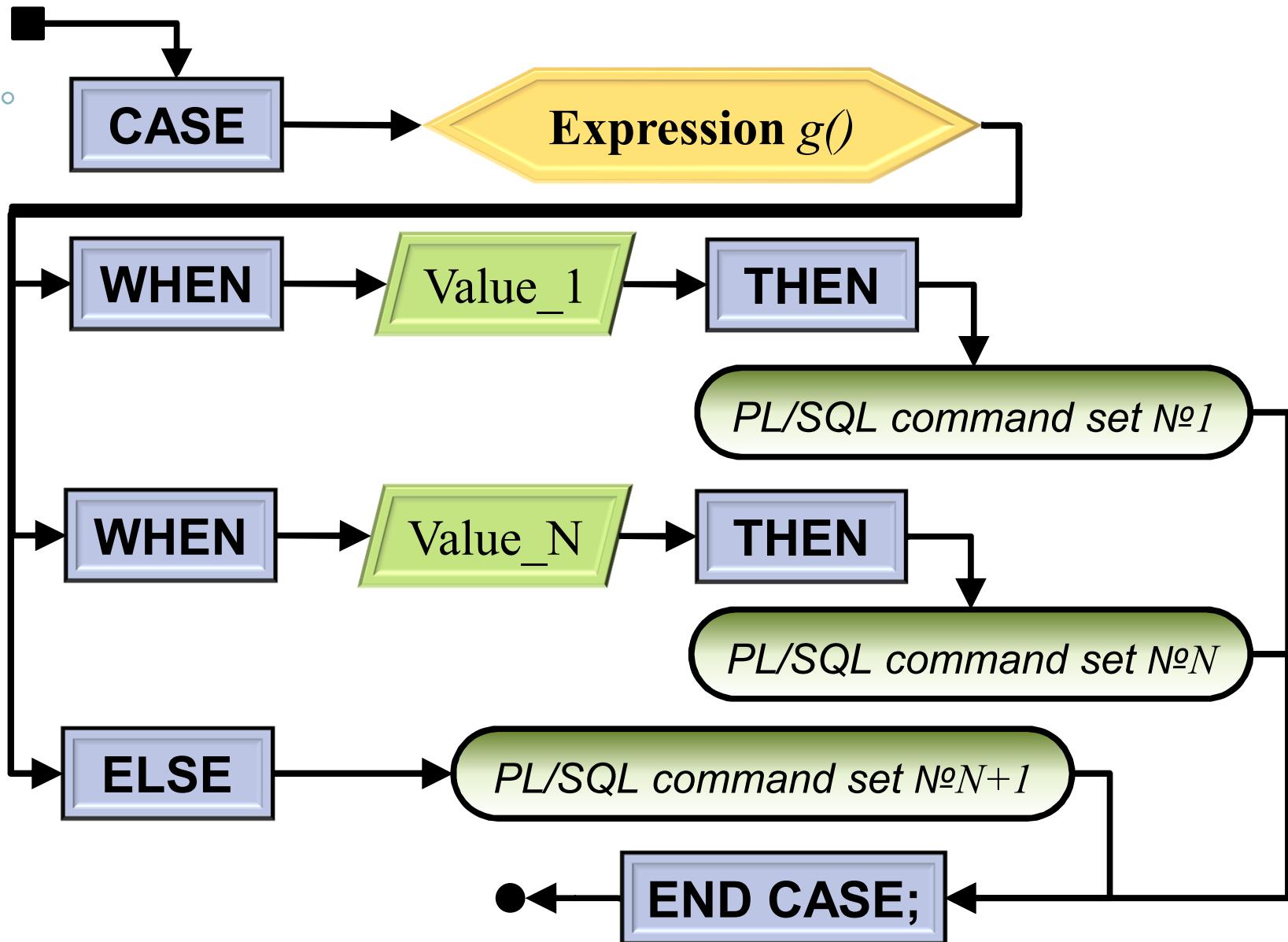
Команди IF. Пришвидшене обчислення

```
IF  
    condition №1  
    AND  
    condition №2  
    AND  
    condition №3  
    .....  
    AND  
    condition №N  
THEN  
    command set №1  
ELSE  
    command set №2  
END IF;
```

Команди і вирази CASE

№	Різновид CASE	Характеристики
1.	Простий CASE;	<i>Пов'язує одну або декілька послідовностей команд PL/SQL з відповідними значеннями (виконувана послідовність вибирається з урахуванням результату обчислення виразу, що повертає одне ззначення).</i>
2.	Пошуковий CASE;	<i>Вибирає для виконання одну або декілька послідовностей команд в залежності від результатів перевірки списку логічних значень. Виконується послідовність команд, пов'язана з першою умовою, результат перевірки якого виявився рівним TRUE</i>

Проста команда CASE. Синтаксис.



Проста команда CASE. Приклад.

```
CASE employee_type
WHEN 'S' THEN
    award_salary_bonus(employee_id);
WHEN 'H' THEN
    award_hourly_bonus(employee_id);
WHEN 'C' THEN
    award_commissioned_bonus(employee_id);
ELSE
    RAISE invalid_employee_type;
END CASE;
```

Проста команда CASE. Приклад.

CASE TRUE

WHEN salary ≥ 10000 **AND** salary ≤ 20000

THEN

give_bonus(employee_id, 1500);

WHEN salary > 20000 **AND** salary ≤ 40000

THEN

give_bonus(employee_id, 1000);

WHEN salary > 40000

THEN

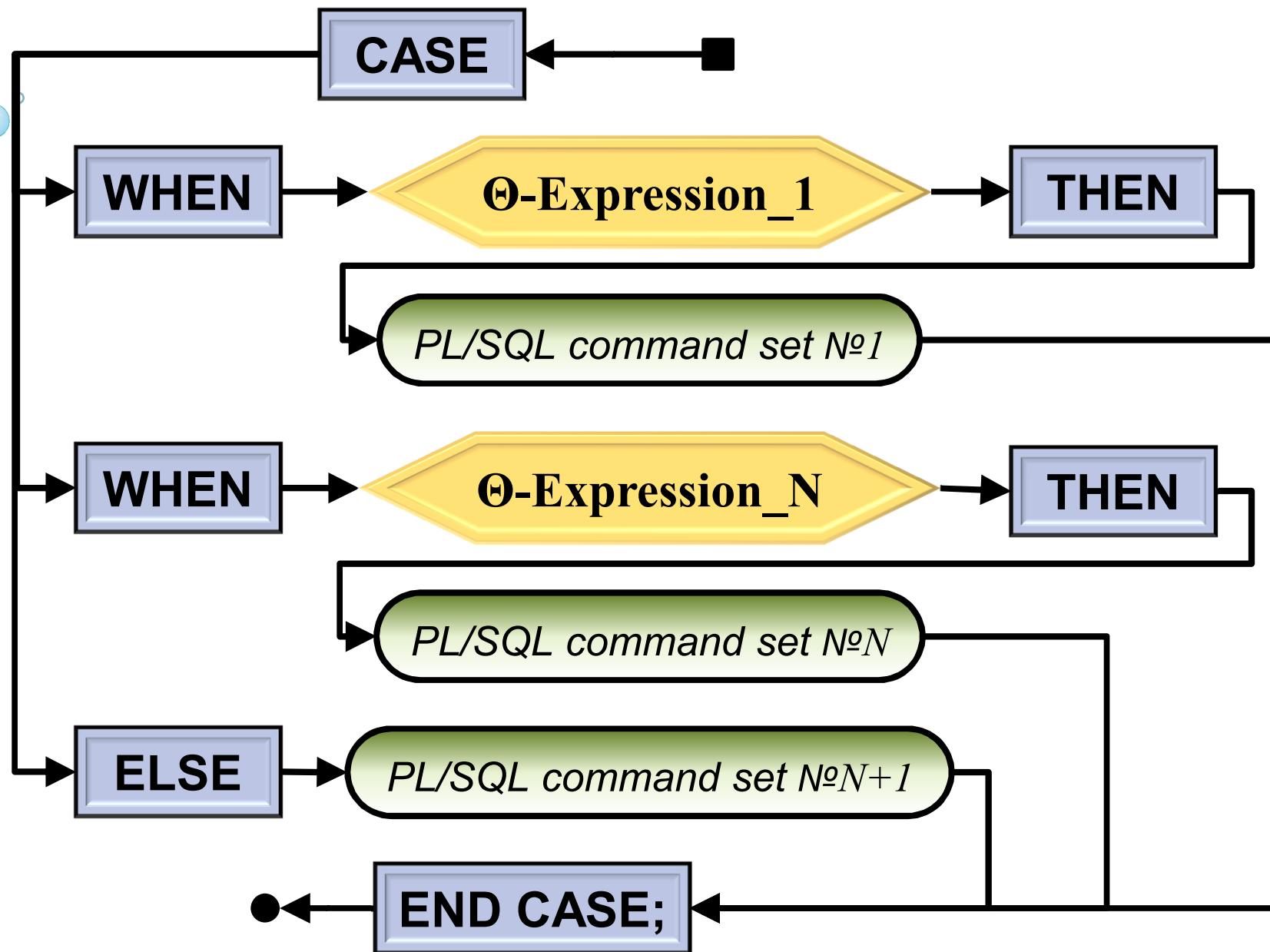
give_bonus(employee_id, 500);

ELSE

give_bonus(employee_id, 0);

END CASE;

Пошукова команда CASE. Синтаксис.



Пошукова команда CASE. Приклад.

CASE

WHEN salary ≥ 10000 **AND** salary ≤ 20000

THEN

give_bonus(employee_id, 1500);

WHEN salary > 20000 **AND** salary ≤ 40000

THEN

give_bonus(employee_id, 1000);

WHEN salary > 40000

THEN

give_bonus(employee_id, 500);

ELSE

give_bonus(employee_id, 0);

END CASE;

Пошукова команда CASE. Приклад.

CASE

WHEN salary > 40000 **THEN**

give_bonus(employee_id, 500);

WHEN salary > 20000 **THEN**

give_bonus(employee_id, 1000);

WHEN salary >= 10000 **THEN**

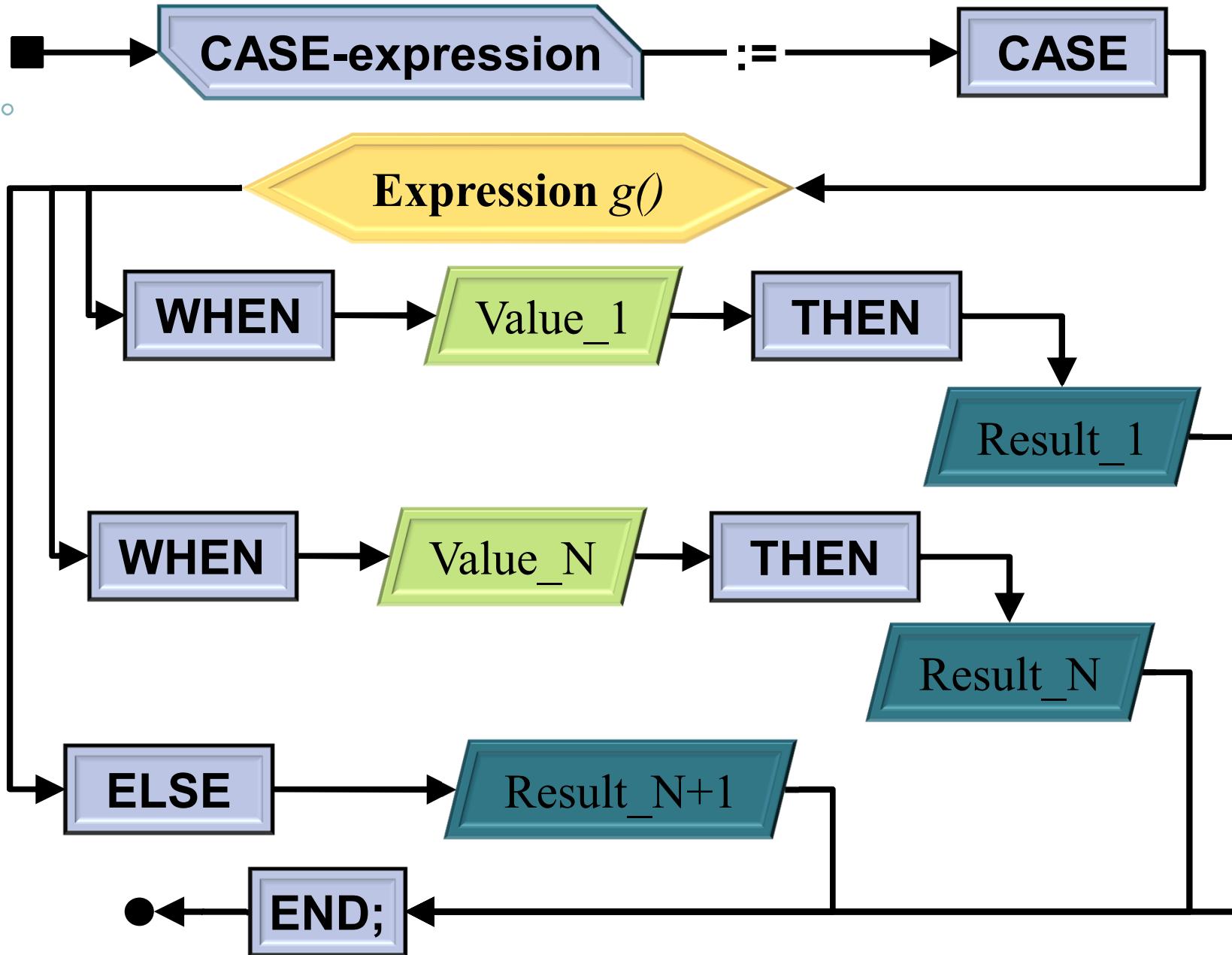
give_bonus(employee_id, 1500);

ELSE

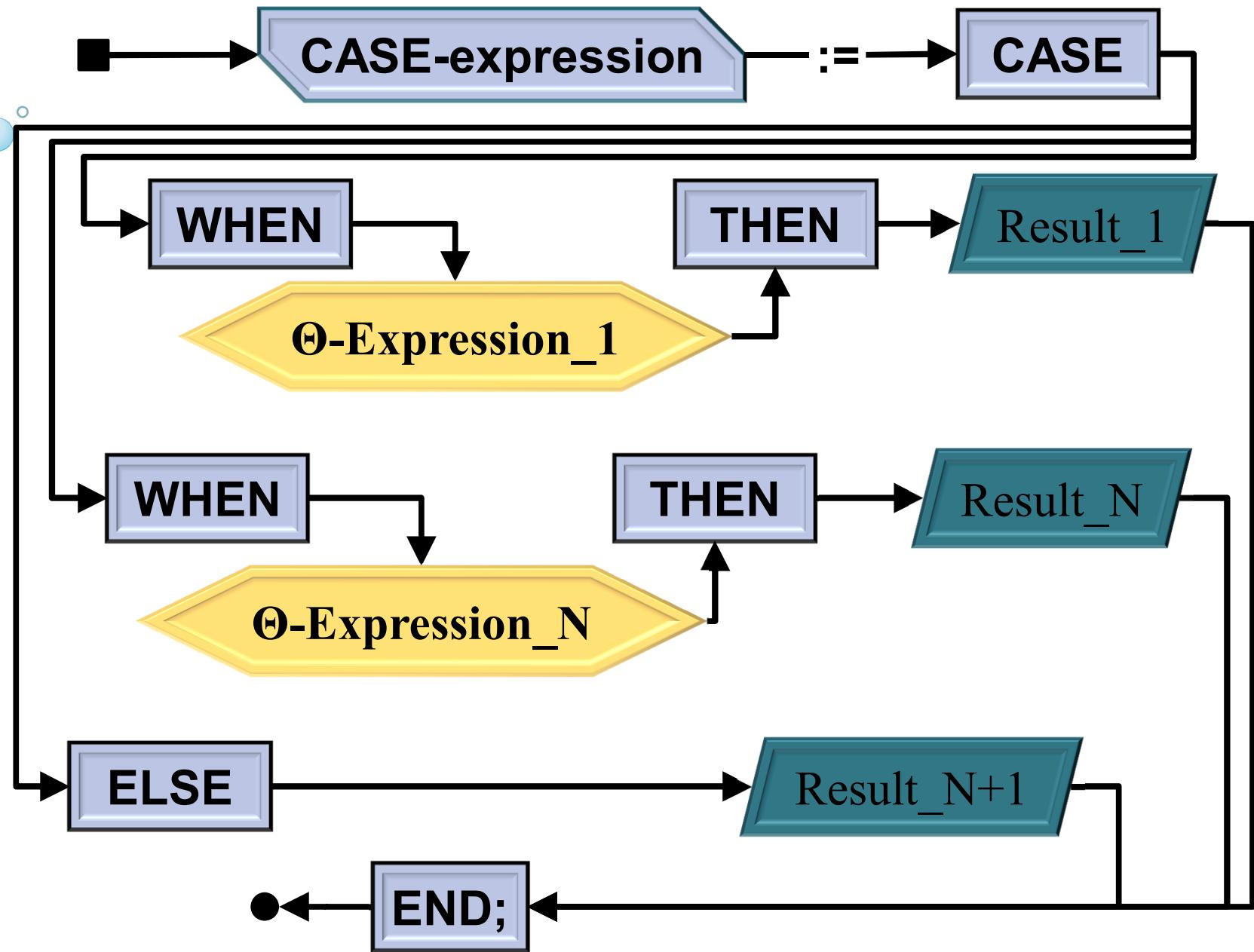
give_bonus(employee_id, 0);

END CASE;

Простий CASE -вираз. Синтаксис.



Пошуковий CASE -вираз. Синтаксис.



CASE -вираз. Приклад.

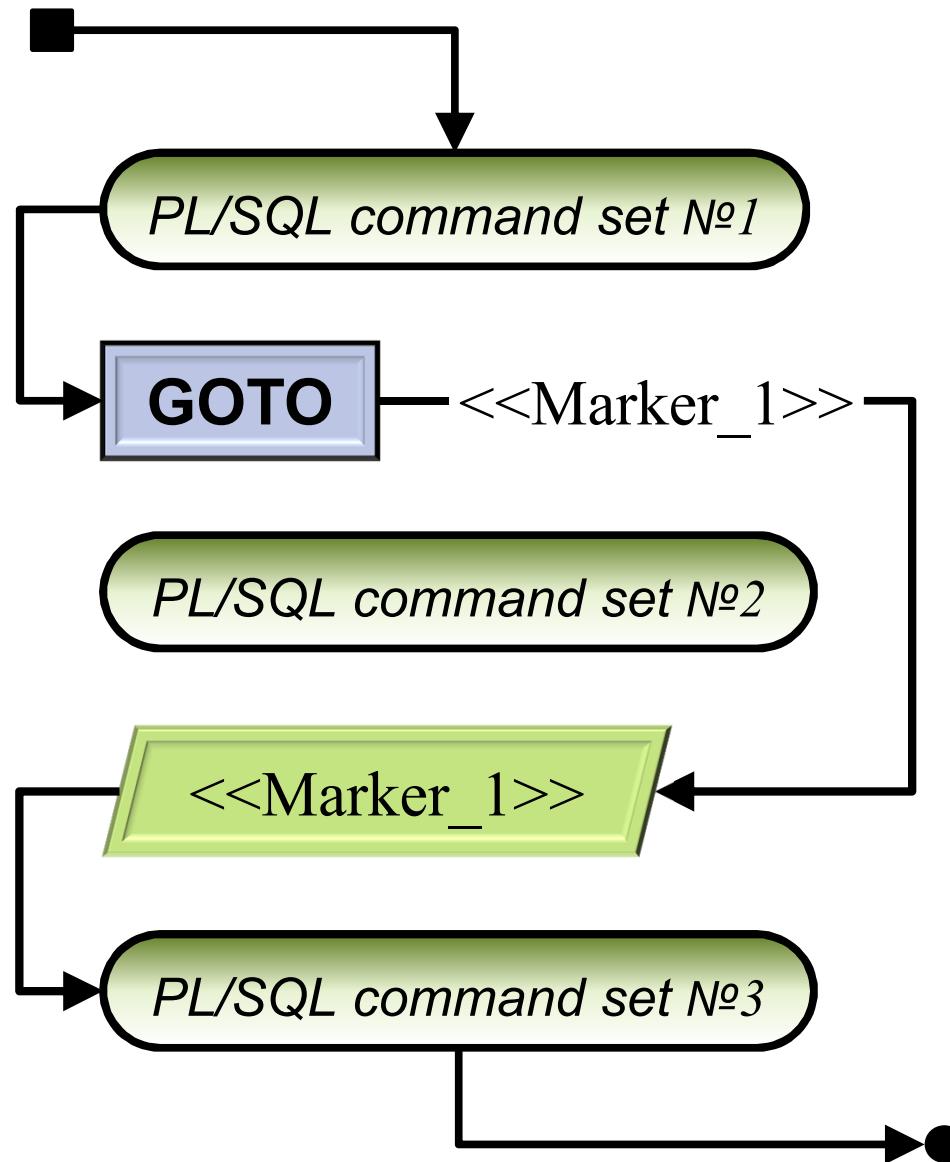
DECLARE

```
    salary NUMBER := 20000;  
    employee_id NUMBER := 36325;  
    bonus_amount NUMBER;
```

BEGIN

```
    bonus_amount := CASE  
        WHEN salary >= 10000 AND salary <= 20000  
        THEN 1500  
        WHEN salary > 20000 AND salary <= 40000  
        THEN 1000  
        WHEN salary > 40000  
        THEN 500  
        ELSE 0  
    END * 10;  
    DBMS_OUTPUT.PUT_LINE(bonus_amount);  
END;
```

Команда GOTO. Синтаксис.



Команда **GOTO**. Приклад.

BEGIN

```
GOTO second_output;  
DBMS_OUTPUT.PUT_LINE  
(‘Цей рядок ніколи не виконується.’);  
<<second_output>>  
DBMS_OUTPUT.PUT_LINE  
(‘Виконується цей рядок!’);
```

END;

- За міткою повинна слідувати хоча б одна виконувана команда *PL/SQL*.
- Цільова мітка повинна бути розташована в межах області дії оператора *GOTO*.
- Цільова мітка повинна бути розташована у тій же частині блоку *PL/SQL*, що і оператор *GOTO*.

СУЧАСНІ СУБД

Лекція №12

Тема:

ORACLE PL/SQL

Команди ітеративного розгалуження

Цикли в PL/SQL

Під командами ітеративного розгалуження необхідно розуміти такі керуючі структури PL/SQL, як цикли. Цикли призначені для багаторазового виконання певних фрагментів програмного коду. Також в Oracle-11g з'явилась команда **CONTINUE**, яка теж відноситься до команд ітеративного розгалуження. PL/SOL підтримує цикли трьох видів: прості (нескінченні), **FOR** і **WHILE**.

Три різновиди циклів потрібні для створення оптимального алгоритму розв'язанняожної конкретної задачі. У більшості випадків задача може бути вирішена за допомогою будь-якої з трьох циклічних конструкцій, але при невдалому виборі конструкції програмний код буде містити безліч зайвих рядків, що ускладнить розуміння і супровід написаних модулів.

Простий цикл починається з ключового слова **LOOP** і завершується командою **END LOOP**. Виконання циклу переривається при виконанні команди **EXIT**, **EXIT WHEN** або **RETURN** в тілі циклу (або при виникненні виключення).

Цикл FOR існує в двох формах: числовий і курсорний. Числова форма починається з ключового слова **FOR**, після якого задається початкове і кінцеве цілочисельні значення. Тіло циклу починається командою **LOOP** і завершується командою **END LOOP**. PL/SQL перебирає всі проміжні значення вказаного діапазону, після чого завершує цикл.

Цикл WHILE має багато спільного з простим циклом. Принципова відмінність полягає в тому, що умова завершення перевіряється перед виконанням чергової ітерації. Можливі ситуації, в яких тіло циклу не буде виконано жодного разу:

Цикли в PL/SQL. Приклади.

Простий цикл

```

BEGIN
  LOOP
    EXIT WHEN (l_current_year > end_year_in);
    display_total_sales(l_current_year);
    l_current_year := l_current_year + 1;
  END LOOP;
END
/

```

Цикл WHILE

```

BEGIN
  WHILE (l_current_year <= end_year_in)
  LOOP
    display_total_sales(l_current_year);
    l_current_year := l_current_year + 1;
  END LOOP;
END
/

```

Цикл FOR

```

BEGIN
  FOR l_current_year IN start_year_in .. end_year
  LOOP
    display_total_sales(l_current_year)
  END LOOP;
END
/

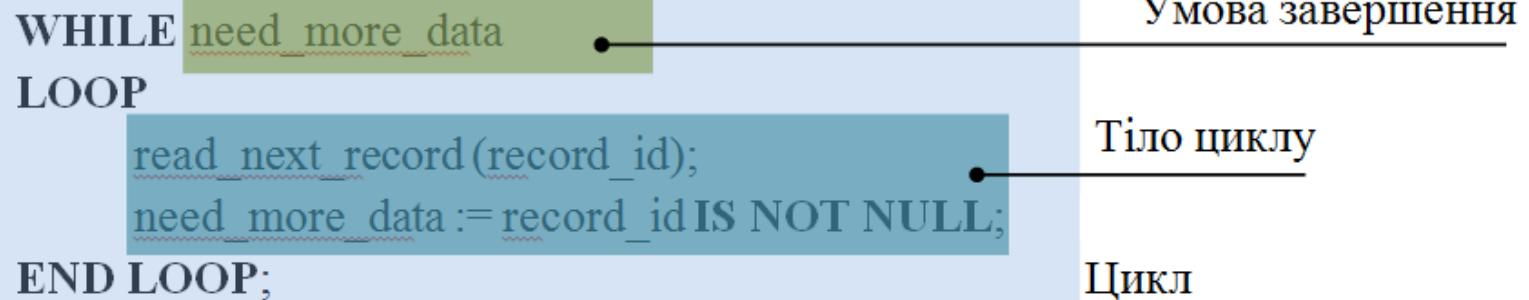
```

Структура циклів PL/SQL.

Незважаючи на відмінності між різними формами циклічних конструкцій, кожен цикл складається з двох частин: обмежувачів і тіла циклу.

Обмежувачі – ключові слова, що визначають початок циклу, умова завершення, і команда **END LOOP**, що завершує цикл.

Тіло циклу – послідовність виконуваних команд всередині границь циклу, що виконуються на кожній ітерації.



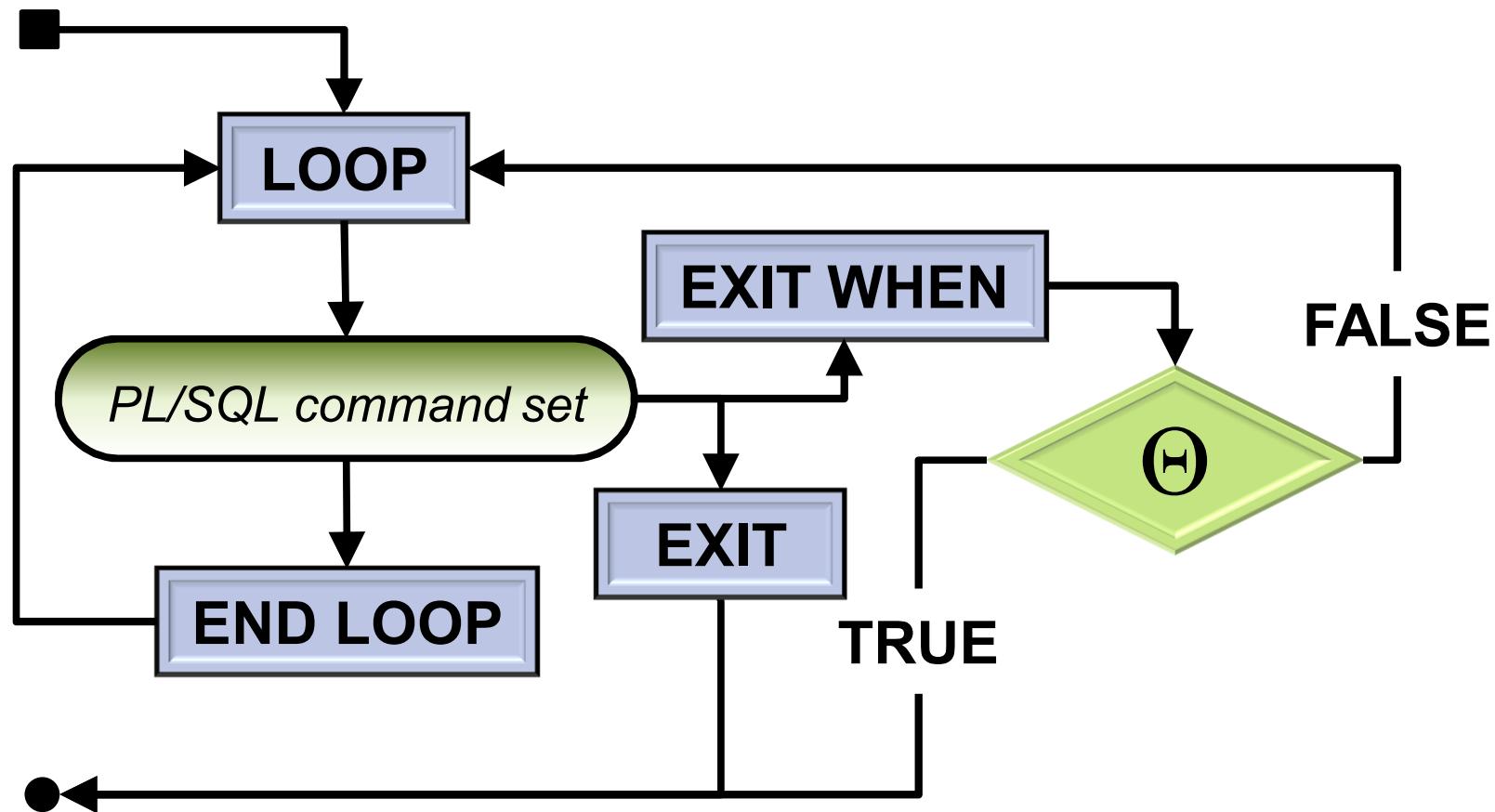
Простий цикл.

Структура простого циклу є найелементарнішій серед всіх циклічних конструкцій. Такий цикл складається з ключового слова **LOOP**, виконуваного коду (тіла циклу) і ключових слів **END LOOP**. Цикл починається командою **LOOP**, а закінчується командою **END LOOP**. Тіло циклу повинно містити як мінімум одну виконувану команду.

Властивості простого циклу

№	Властивість	Опис
1	Умова завершення циклу	Якщо в тілі циклу виконується команда EXIT . В іншому випадку цикл виконується нескінченно.
2	Позиція перевірки умови	У тілі циклу і тільки при виконанні команди EXIT або EXIT WHEN . Тіло циклу (або його частина) завжди виконується як мінімум один раз
3	Доцільність використання	<ol style="list-style-type: none"> 1) Якщо не відомо, скільки разів буде виконуватися тіло циклу; 2) Тіло циклу має бути виконане хоча б один раз

Простий цикл.



Простий цикл.Приклади.

LOOP

LOOP

```

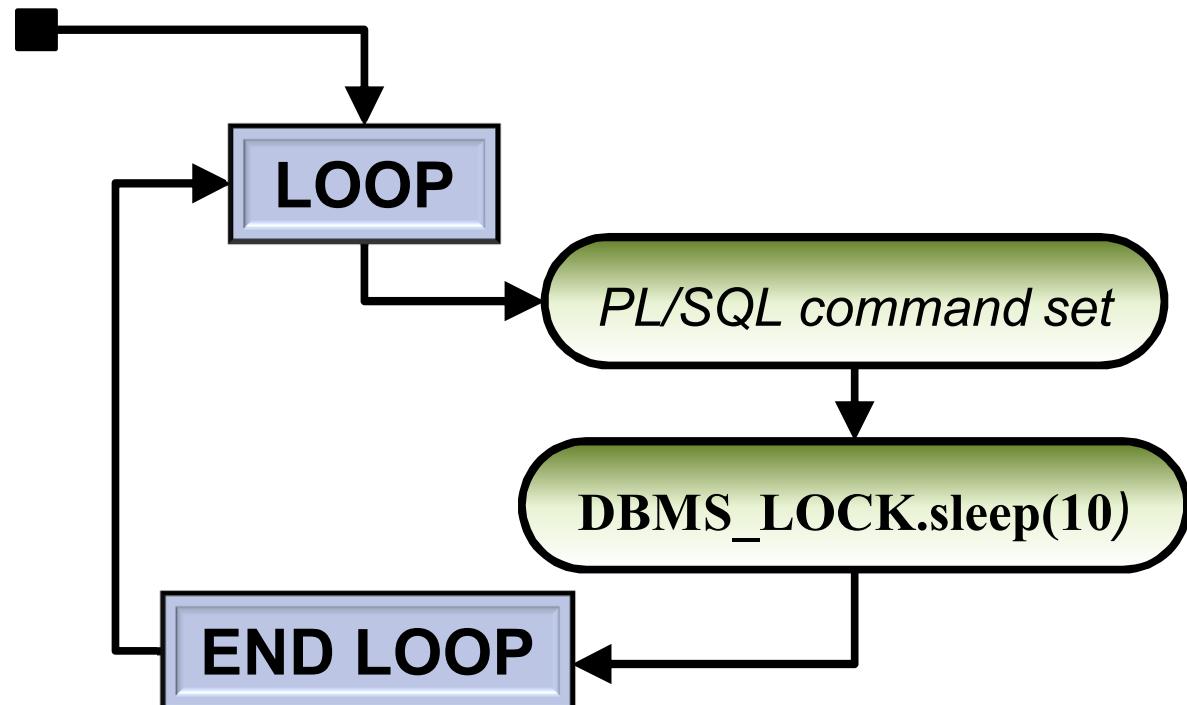
/* Обчислення балансу */
    balance_remaining := account_balance(account_id);
/* Умова вбудована в команду EXIT */
    EXIT WHEN balance_remaining < 1000;
/* Якщо цикл продовжує виконуватись з балансу списуються кошти */
    apply_balance(account_id, balance_remaining);
END LOOP;

```

- Команду **EXIT WHEN** використовують, коли умова завершення циклу визначається лише одним **Θ**-виразом.
- Команду **EXIT** використовують, коли має місце кілька умов завершення циклу або якщо при виході має бути визначено значення, що повертається. Разом з **EXIT** використовують **IF** або **CASE**.

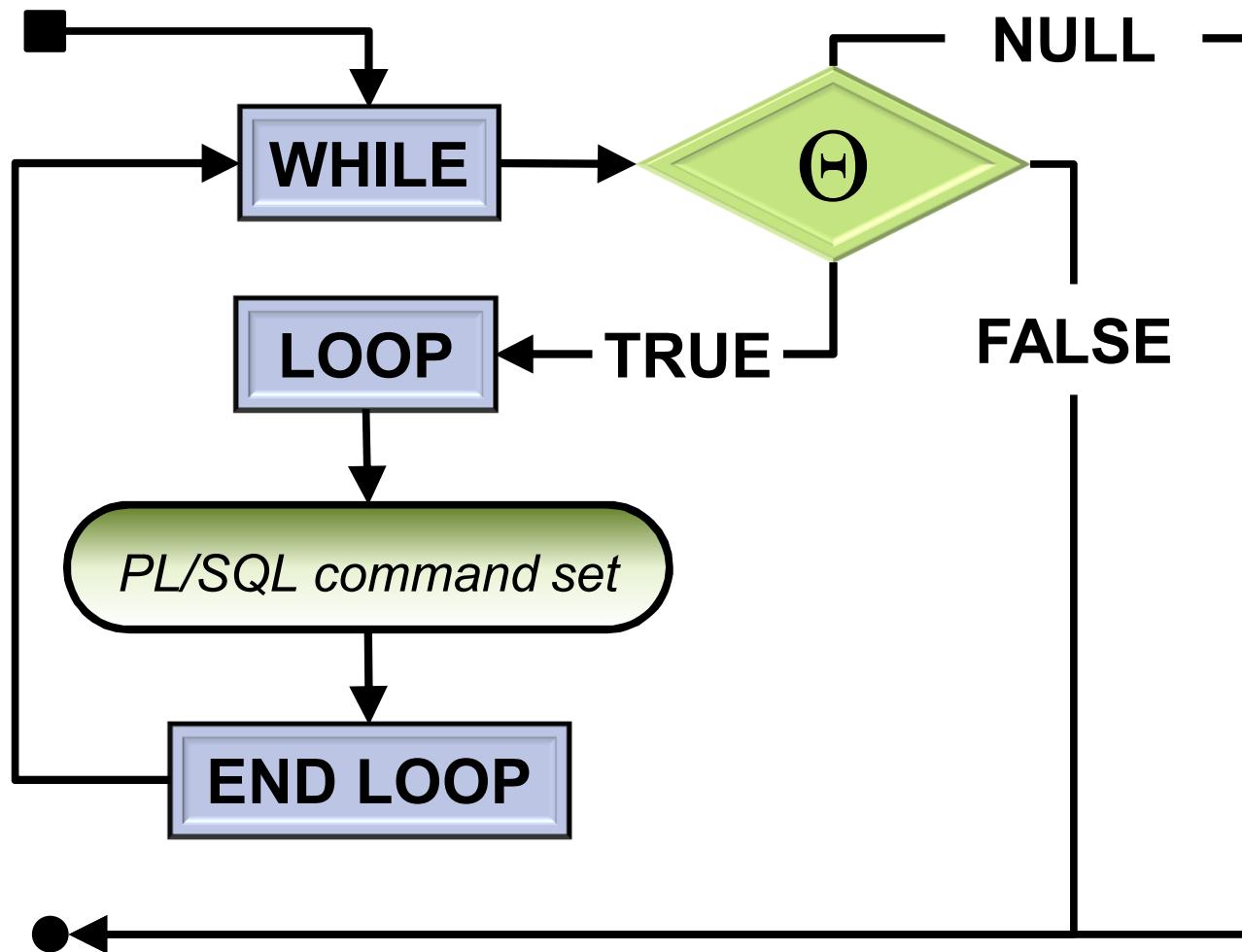
Нескінчений цикл.

Деякі програми (наприклад, засоби спостереження за станом системи) розраховані на безперервне виконання з накопиченням необхідної інформації. У таких випадках можна навмисно використовувати нескінчений цикл. Нескінчений цикл зазвичай поглинає значну частину ресурсів процесора. Проблема вирішується припиненням виконання між ітераціями командою `DBMS_LOCK.sleep(t NUMBER)`. Під час призупинення програма практично не витрачає ресурси процесора.



Цикл WHILE.

Умовний цикл WHILE виконується до тих пір, поки визначена в циклі умова залишається рівною TRUE. А оскільки можливість виконання циклу залежить від умови і не обмежується фіксованою кількістю повторень, він використовується саме в тих випадках, коли кількість повторень циклу заздалегідь невідома.



Цикл WHILE.

Умова – логічна змінна або вираз (Θ -вираз), результатом перевірки якого є логічне значення TRUE, FALSE або NULL. Умова перевіряється при кожній ітерації циклу. Якщо результат виявляється рівним TRUE, тіло циклу виконується. Якщо ж результат дорівнює FALSE або NULL, то цикл завершується, а управління передається виконуваної команді, наступної за командою END LOOP.

Властивості циклу WHILE

№	Властивість	Опис
1	Умова завершення циклу	Якщо значенням логічного виразу циклу є FALSE або NULL.
2	Позиція перевірки умови	Перед першим та кожним наступним виконанням тіла циклу. Таким чином, не гарантується навіть одноразове виконання тіла циклу WHILE
3	Доцільність використання	<ol style="list-style-type: none"> 1) Якщо не відомо, скільки разів буде виконуватися тіло циклу; 2) можливість виконання циклу повинна визначатися умовою; 3) тіло циклу може не виконуватися жодного разу.

Цикл WHILE. Приклад.

WHILE

mask_index <= mask_count

AND

NOT dateConverted

LOOP

BEGIN

/* Спроба перетворення рядка по масці в записі таблиці */

retval := TO_DATE (value_in, fmts (mask_index));

dateConverted := TRUE;

EXCEPTION WHEN OTHERS

THEN

mask_index := mask_index + 1;

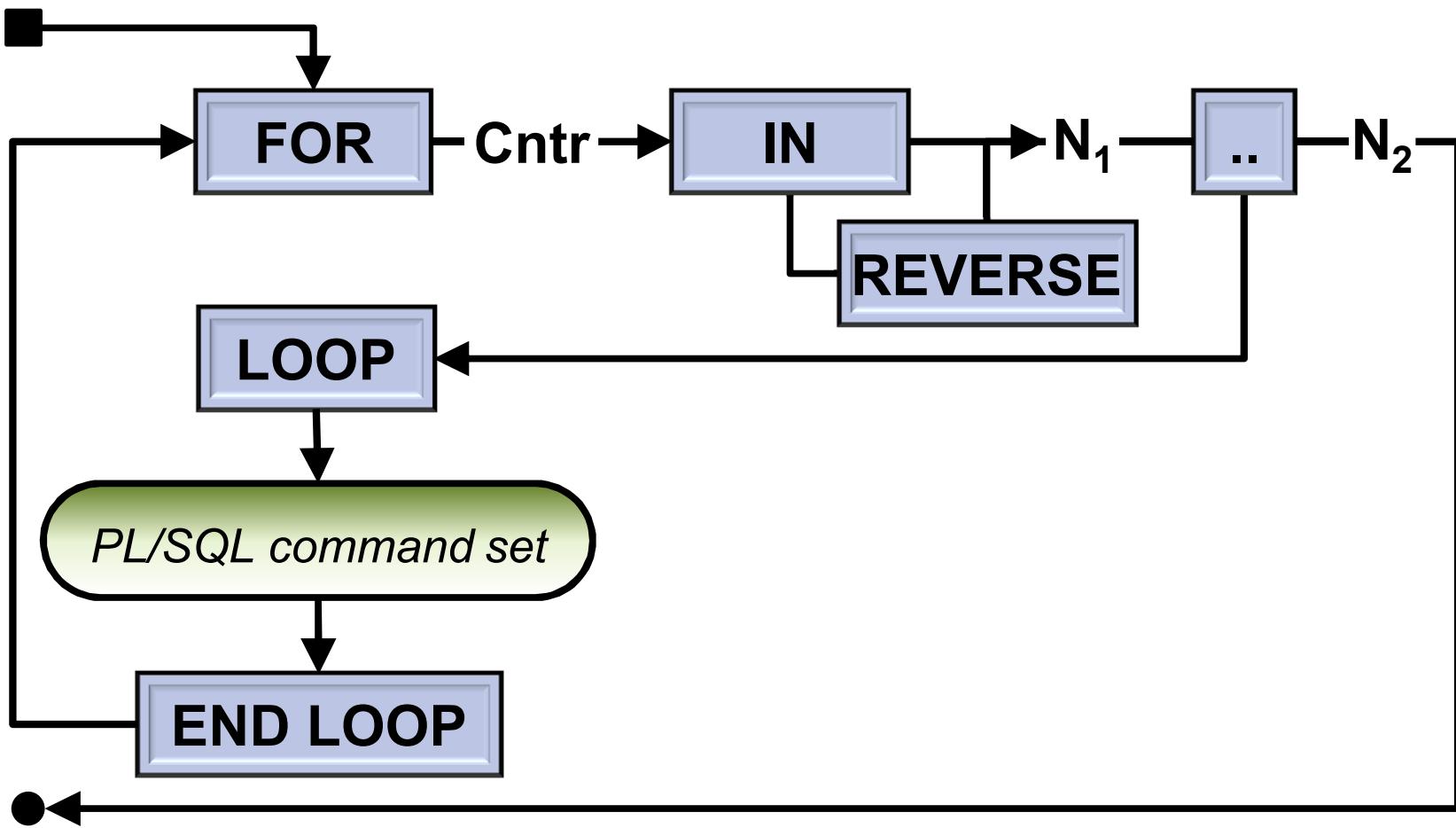
END;

END LOOP;

- Вся інформація, необхідна для обчислення умови, повинна задаватися перед первішим виконанням циклу.
- Може виявитися, що цикл WHILE не буде виконано жодного разу.

Цикл FOR з лічильником

Цикл з лічильником - це традиційний цикл FOR, підтримуваний в більшості мов програмування. Кількість ітерацій цього циклу відомо ще до його початку і задається діапазоном, певним між ключовими словами FOR і LOOP. Діапазон неявно оголошує керуючу змінну циклу (якщо вона не була явно оголошена раніше), визначає початкове і кінцеве значення діапазону, а також задає напрямок зміни лічильника (по зростанню або по спадаючий).



Цикл FOR з лічильником

Властивості циклу FOR

№	Властивість	Опис
1	Умова завершення циклу	Числовий цикл FOR безумовно завершується при виконанні кількості ітерацій, визначеного діапазоном значень лічильника. (Цикл може завершуватися і командою EXIT, але робити цього не рекомендується)
2	Позиція перевірки умови	Після кожного виконання тіла циклу PL/SQL перевіряє значення лічильника. Якщо воно виходить за межі заданого діапазону, виконання циклу припиняється. Якщо початкове значення більше кінцевого, то тіло циклу не виконується жодного разу.
3	Доцільність використання	Якщо тіло циклу повинно бути виконано певну кількість разів, а виконання не повинно перериватися передчасно

- **Не оголошуйте лічильник циклу.** PL/SQL автоматично неявно оголошує локальну змінну з типом даних INTEGER. Область дії цієї змінної збігається з границями циклу. Звертатися до лічильника за межами циклу не можна.

- **Вирази, які використовуються при визначенні діапазону, обчислюються один раз.** Вони не перераховуються в ході виконання циклу. Якщо змінити всередині циклу змінні, використовувані для визначення діапазону значень лічильника, його кордони залишаться колишніми.

- **Не міняйте значення лічильника і меж діапазону всередині циклу.** Компілятор PL/SQL або видасть повідомлення про помилку, або проігнорує зміни. В будь-якому випадку виникнуть проблеми.

- **Щоб значення лічильника зменшувалися в напрямку від кінцевого до початкового, використовуйте ключове слово REVERSE.** При цьому перше значення у визначенні діапазону (пачаткове_значення) має бути менше другого (конечное_значение). Не міняйте порядок проходження значень - просто поставте ключове слово REVERSE.

Цикли FOR з числовим лічильником. Приклади.

*Цикл виконується 10 раз;
лічильник збільшується від 1 до 10*

```
FOR loop_counter IN 1 .. 10
LOOP
    ... виконувані_команди...
END LOOP;
```

*Цикл виконується 10 раз;
лічильник зменшується від 10 до 1*

```
FOR loop_counter IN REVERSE 1 .. 10
LOOP
    ... виконувані_команди...
END LOOP;
```

Цикл не виконується жодного разу

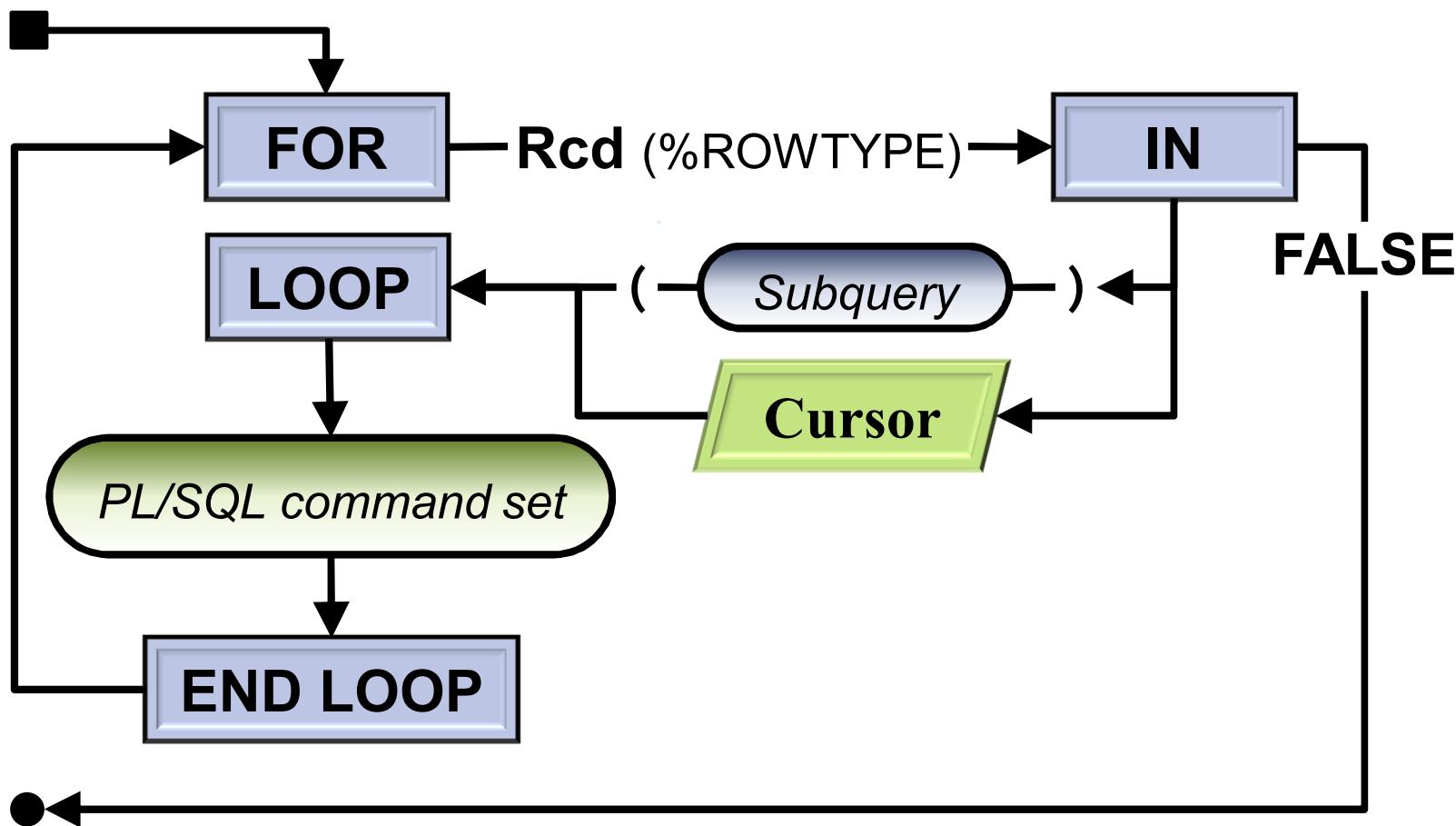
```
FOR loop_counter IN REVERSE 10 .. 1
LOOP
    ... виконувані_команди...
END LOOP;
```

Цикл виконується в діапазоні значень змінної та виразу

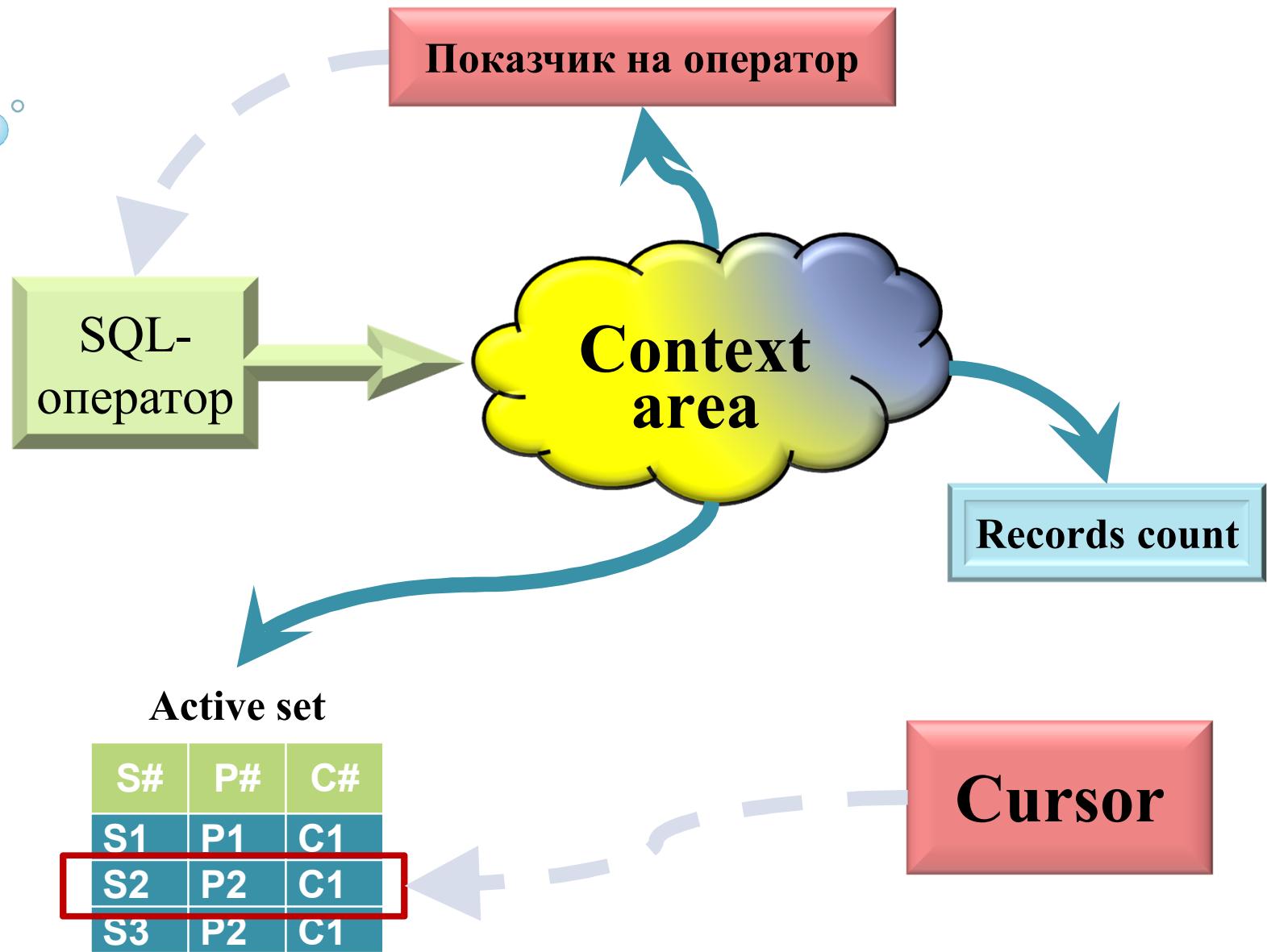
```
FOR calc_index IN
    start_period_number .. LEAST (end_period_number, current_period)
LOOP
    ... виконувані_команди...
END LOOP;
```

Цикли FOR з курсором.

Курсорна форма циклу **FOR** визначається явно заданим курсором або інструкцією **SELECT**, заданої безпосередньо в межах циклу. Цю форму використовують в тому випадку, якщо необхідно вилучити і обробити всі записи вибірка даних. Цикл **FOR** з курсором забезпечує ефективну інтеграцію процедурних конструкцій з безпосереднім доступом до баз даних. Його застосування помітно скорочує обсяг коду, та зменшує імовірність виникнення помилок при циклічній обробці даних.



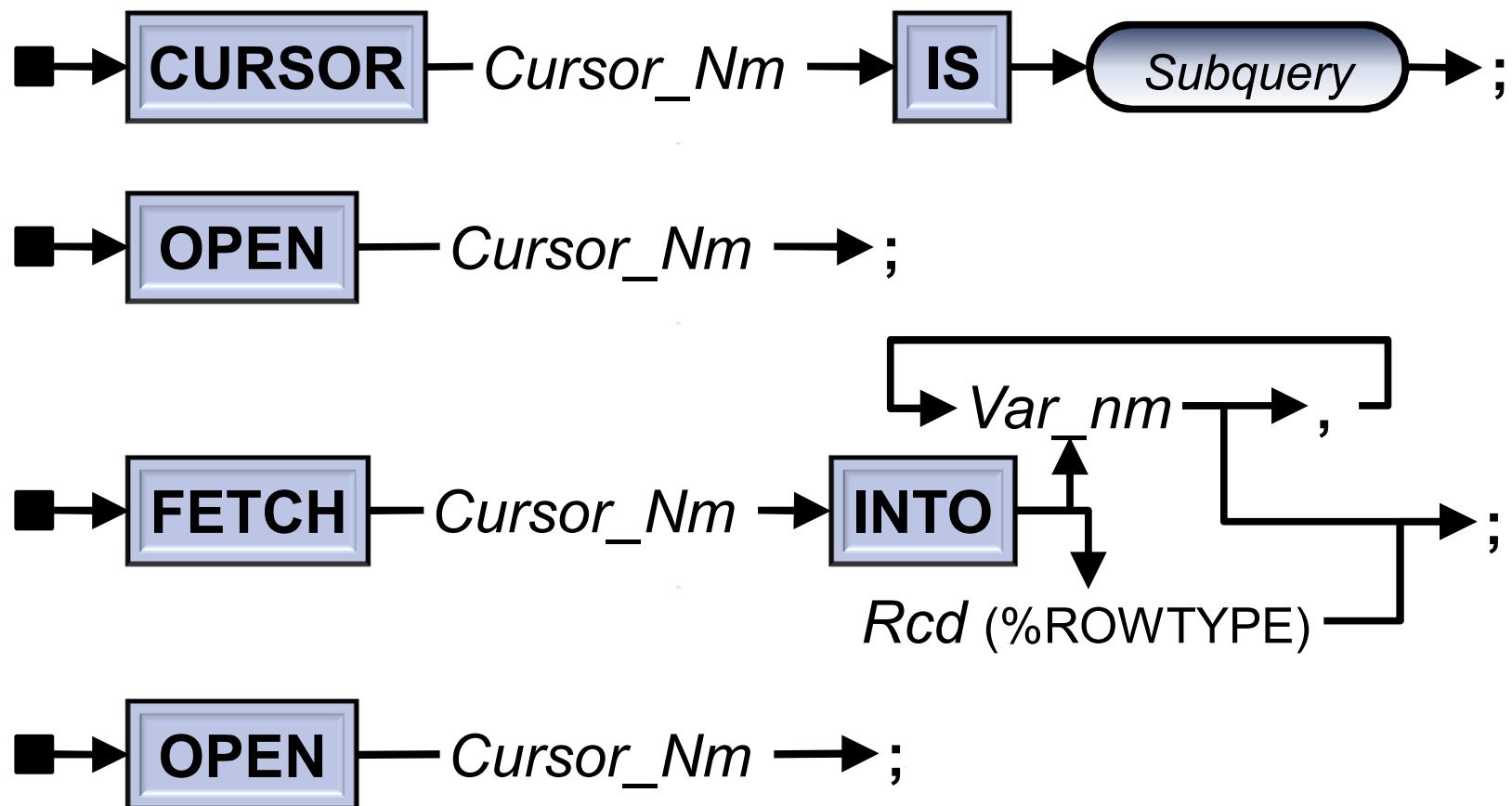
Поняття курсора



Створення і обробка курсора

Для створення і обробки явного курсору в PL/SQL необхідно:

1. Оголосити курсор.
2. Відкрити курсор для запиту.
3. Вибрати результати в змінні PL/SQL.
4. Закрити курсор.



Курсорний цикл FOR. Приклади.

DECLARE

```
CURSOR occupancy_cur IS
    SELECT pet_id, room_number
    FROM occupancy
    WHERE occupied_dt = TRUNC (SYSDATE);
    occupancy_rec  occupancy_cur%ROWTYPE;
```

BEGIN

```
OPEN occupancy_cur;
LOOP
```

```
    FETCH occupancy_cur INTO occupancy_rec;
    EXIT WHEN occupancy_cur%NOTFOUND;
    update_bill(occupancy_rec.pet_id, occupancy_rec.room_number);
```

END LOOP;

CLOSE occupancy_cur;

END;

DECLARE

```
CURSOR occupancy_cur IS
    SELECT pet_id, room_number
    FROM occupancy
    WHERE occupied_dt = TRUNC (SYSDATE);
```

BEGIN

```
FOR occupancy_rec IN occupancy_cur
LOOP
```

```
    update_bill(occupancy_rec.pet_id, occupancy_rec.room_number);
```

END LOOP;

END;

BEGIN

```
FOR occupancy_rec IN
    (

```

```
        SELECT pet_id, room_number
        FROM occupancy
        WHERE occupied_dt = TRUNC (SYSDATE)
```

)

LOOP

```
    update_bill(occupancy_rec.pet_id, occupancy_rec.room_number);
```

END LOOP;

END;

СУЧАСНІ СУБД

Лекція №13

Тема:

ORACLE PL/SQL

Збережені процедури, пакети та тригери

Модульний код

Модуляризацією називають розбиття великих блоків коду на менші блоки (модулі), які можна викликати з інших модулів. Цей процес схожий з нормалізацією даних. Модуляризація дозволяє поліпшити ряд характеристик коду.

№	Характеристика	Опис
1	Придатність для повторного використання	Розбиття великих програм на окремі взаємодіючі компоненти здійснюють так, щоб багато модулів було можливо використати більш ніж однією програмою поточного застосунка. При правильній організації такі програми можуть стати в нагоді в інших програмах!
2	Керованість	При використанні модульного підходу код можна протестувати і налагодити на рівні окремих програм (так зване модульне тестування), тобто до того, як окремі модулі будуть скомбіновані для проведення більш складних інтеграційних тестів.
3	Надійність	Код, розбитий на модулі, містить менше помилок, а виявлені помилки легше виправляти, оскільки вони локалізовані на рівні модуля. Крім того, такий код легше супроводжувати іншим програмістам.
4	Зручність читання	Імена модулів відображають їх призначення. Чим більше коду буде переміщено в програмний інтерфейс або приховано за ним, тим легше буде зрозуміти, що робить програма. Модуляризація дає можливість зосередитися на завданні в цілому, а не на окремих фрагментах коду.

Конструкції PL/SQL для розбиття коду на модулі

№	Конструкція	Опис
1	Процедура	Програма, яка здійснює одну або кілька дій і викликається як виконуваний оператор <i>PL / SQL</i> . Передавати дані процедурі і отримувати їх можна за допомогою списку параметрів.
2	Функція	Програма, яка повертає одне значення і використовується як вираз <i>PL/SQL</i> . Для передачі даних функції використовують список її параметрів. Параметри також можна використовувати для повернення інформації з функції, але зазвичай це вважається проявом поганого стилю програмування.

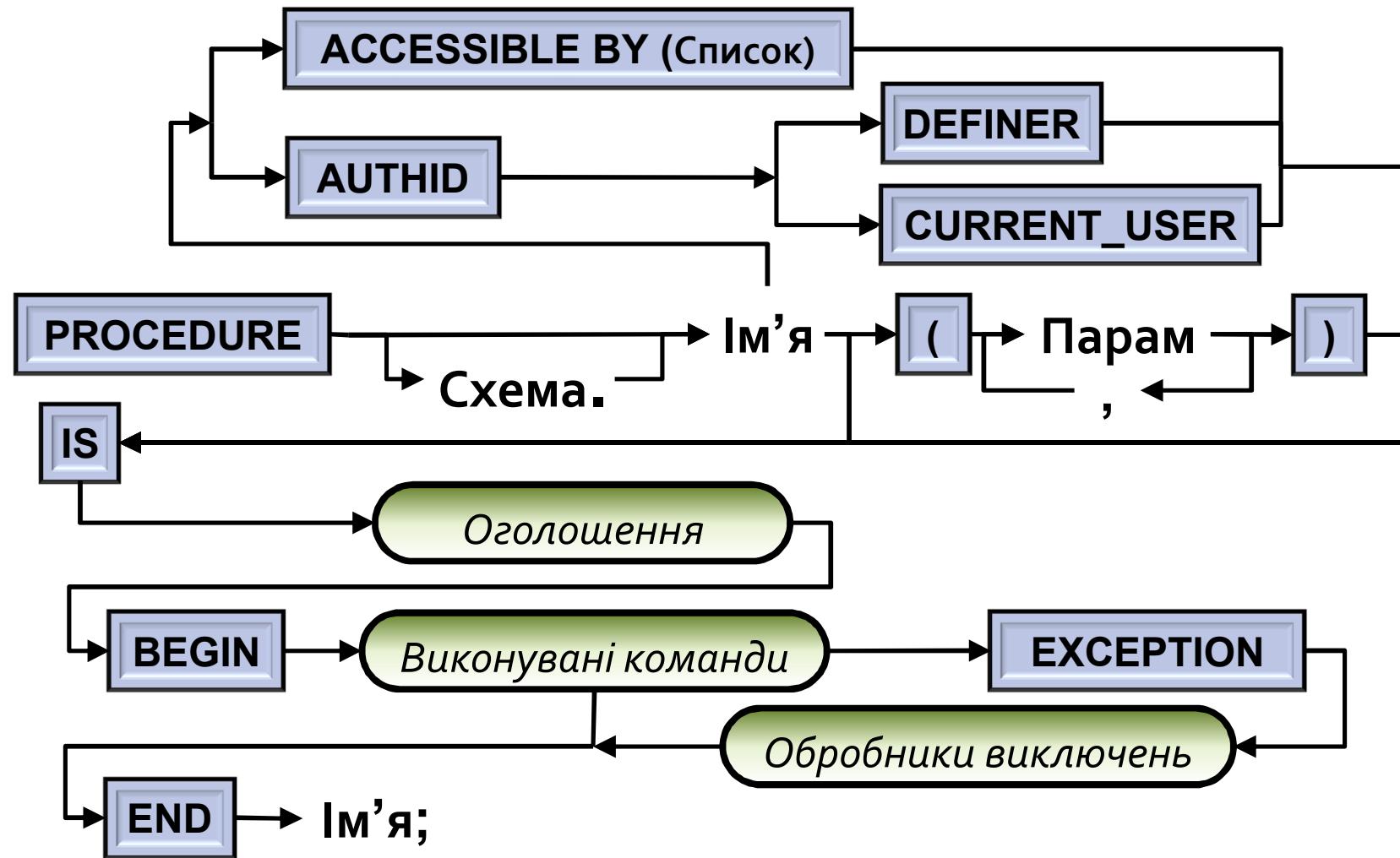
Модульний код

Конструкції PL/SQL для розбиття коду на модулі

№	Конструкція	Опис
3	Тригер бази даних	Набір команд, який викликається при виконанні деякої події (підключення до бази даних, модифікація рядка таблиці або <i>DDL</i> -операція)
4	Пакет	Іменований набір процедур, функцій, типів і змінних. Пакет не є модулем (скоріше, це мета-модуль), але він тісно пов'язаний з реалізацією модульного підходу.
5	Об'єктний тип або екземпляр об'єктного типу	Емуляція об'єктно-орієнтованого класу в Oracle. Об'єктний тип інкапсулює стан і поведінку даних, комбінуючи їх (як реляційна таблиця) з правилами (процедурами і функціями, які маніпулюють цими даними).

Процедури. Синтаксис.

Процедура представляє собою модуль, що виконує одну або кілька дій. Виклик процедури в PL / SQL є окремим виконуваним оператором. Блок PL/SQL коду може складатися тільки з виклику процедури. Процедури відносяться до числа ключових компонентів модульного коду, що забезпечують оптимізацію і повторне використання програмної логіки.



Процедури. Структура.

Схема – ім'я схеми, якій буде належати процедура (необов'язковий аргумент). За замовчуванням застосовується ім'я схеми поточного користувача. Якщо значення схеми відмінне від імені схеми поточного користувача, то цей користувач повинен мати привілей для створення процедури в іншій схемі.

Ім'я — ім'я процедури

Парам – необов'язковий список параметрів, які застосовуються для передачі даних в процедуру і повернення інформації з процедури в точку виклику.

AUTHID - визначає, за якими дозволами буде викликатися процедура: творця (власника) або поточного користувача. У першому випадку процедура виконується з правами творця, у другому - з правами того хто викликає.

Оголошення - оголошення локальних ідентифікаторів цієї процедури. Якщо оголошення відсутні, між ключовими словами **IS** і **BEGIN** не буде жодних виразів.

ACCESSIBLE BY (Oracle Database 12c) – обмежує доступ до процедури програмним модулем, перерахованим в круглих дужках.

Виконувані команди – команди, що виконуються процедурою при виклику. Між ключовими словами **BEGIN** і **END** або **EXCEPTION** повинна знаходитися принаймні одна виконувана команда.

Обробники виключень – необов'язкові обробники виключень для процедури. Якщо процедура не обробляє ніяких виключень, слово **EXCEPTION** можна опустити і завершити виконуваний розділ ключовим словом **END**.

Процедури. Приклад.

```

PROCEDURE apply_discount
(
    company_id_in IN company.company_id%TYPE,
    discount_in    IN NUMBER
)
IS
    min_discount CONSTANT NUMBER:=.05;
    max_discount CONSTANT NUMBER:=25;
    invalid_discount EXCEPTION;
BEGIN
    IF discount_in BETWEEN min_discount AND max_discount
    THEN
        UPDATE item SET item_amount:=item_amount*(1-discount_in)
        WHERE EXISTS
        (
            SELECT 'x' FROM order
            WHERE order.order_id=item.order_id
            AND
            order.company_id=company_id_in
        );
        IF SQL%ROWCOUNT = 0 THEN RAISE NO_DATA_FOUND; END IF;
        ELSE RAISE invalid_discount;
    END IF;
EXCEPTION
    WHEN invalid_discount
    THEN DBMS_OUTPUT.PUT_LINE('The specified discount is invalid.');
    WHEN NO_DATA_FOUND
    THEN DBMS_OUTPUT.PUT_LINE('No orders in the system for company');
END apply_discount;

```

Заголовок

Оголошення

Виконуваний розділ

Розділ виключень

Процедури. Виконання.

Виклик процедури

Процедура викликається як виконавська команда **PL/SQL**. Її виклик повинен закінчуватися крапкою з комою ; і може передувати іншим командам **SOL** або **PL/SQL** (якщо такі є) в виконуваному розділі блоку **PL/SOL** або слідувати за ними:

BEGIN

```
    apply_discount (new_company_id., 0.15);
END;
```

Якщо процедура не має параметрів, вона може бути викликана з порожніми круглими дужками або без них:

```
display_store_summary();
display_store_summary;
```

Команда RETURN

Ключове слово **RETURN** зазвичай асоціюється з функціями, оскільки вони повинні повернати значення. Однак **PL/SQL** дозволяє використовувати команду **RETURN** в процедурах. Версія цієї команди для процедур не приймає виразів і не може повернати значення в програмний модуль з точкою виклику процедури. Вона просто припиняє виконання процедури і повертає управління в точку виклику коду.

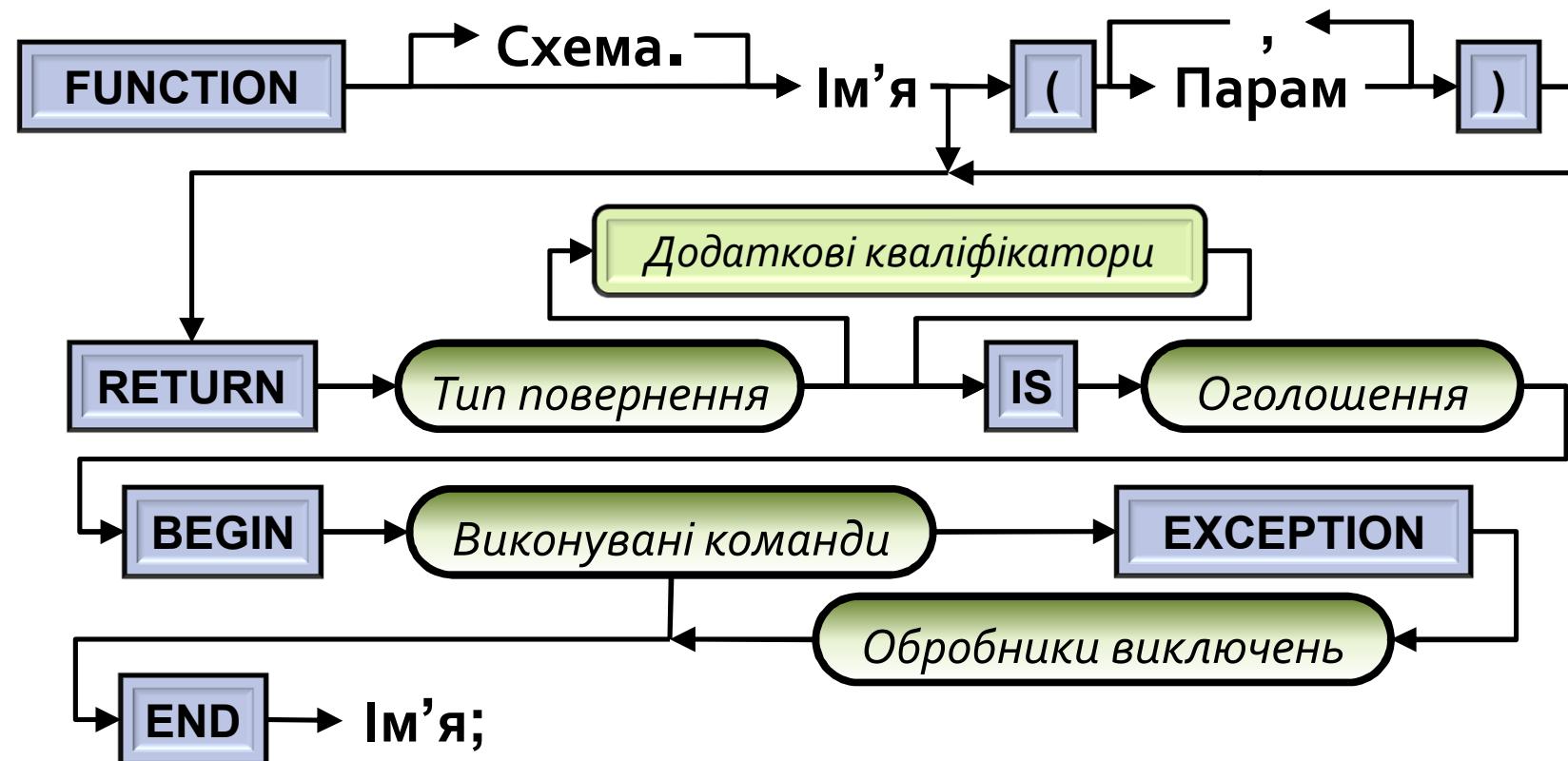


*Використовувати цей різновид **RETURN** не рекомендується, оскільки в цьому випадку в процедурі з'являються дві і більше точки виходу, а це ускладнює логіку виконання. Взагалі бажано уникати використання **RETURN** і **GOTO** для обходу нормальній керуючої структури в програмних елементах.*

Функції. Синтаксис.

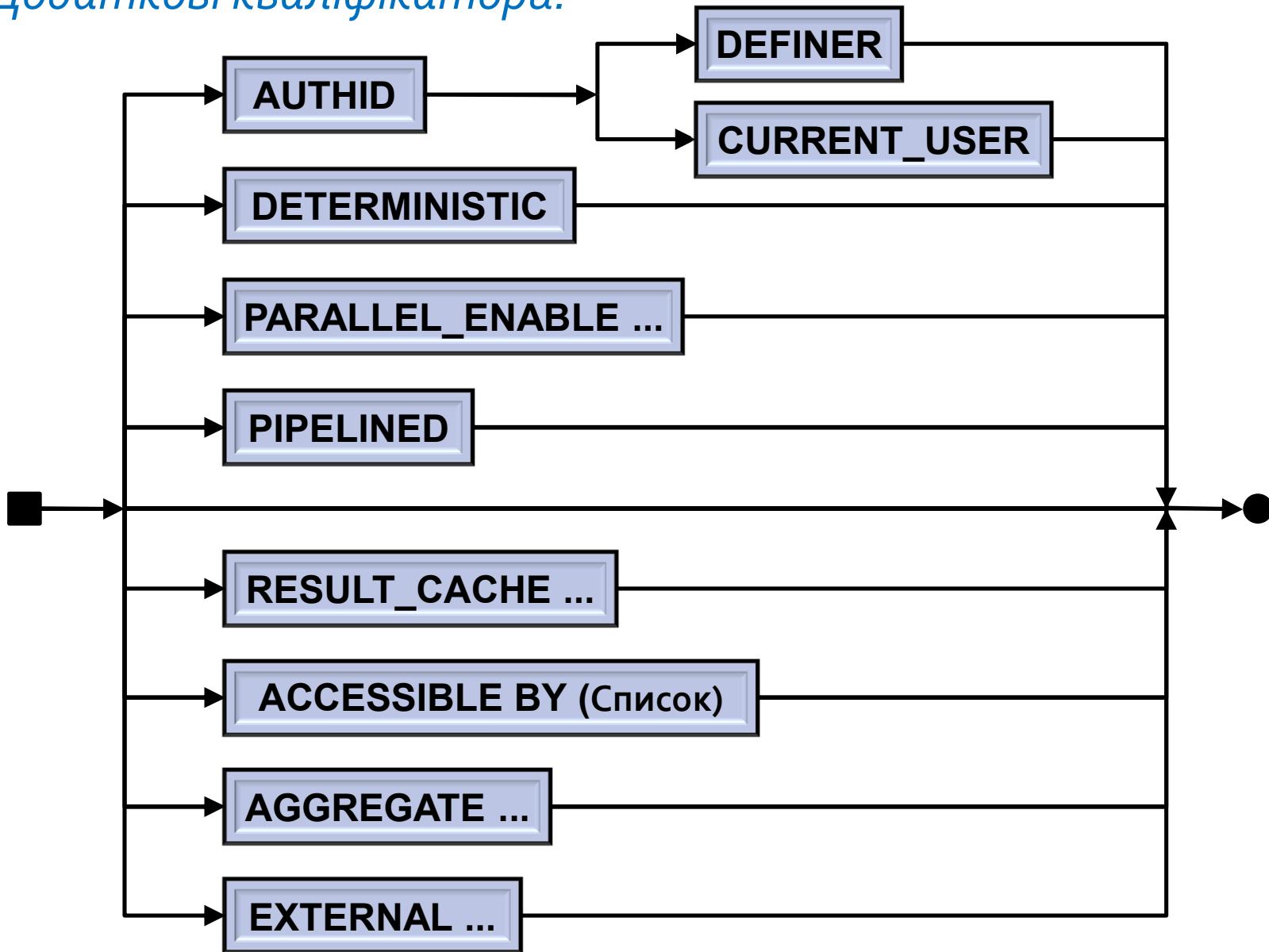
Функція являє собою модуль, який повертає значення командою **RETURN** (замість аргументів **OUT** або **IN OUT**). На відміну від виклику процедури, який являє собою окремий оператор, виклик функції завжди є частиною виконуваного оператора, тобто включається в вираз або править значенням за замовчуванням, що присвоюється змінної при оголошенні.

Значення, що повертається функцією належить до визначеного типу даних. Функції можна використовувати замість виразів, які мають той же тип даних, що і значення, які повертаються.



Функції. Синтаксис.

Додаткові кваліфікатори:



Функції. Структура.

Схема – ім'я схеми, якій буде належати функція (необов'язковий аргумент). За замовчуванням застосовується ім'я схеми поточного користувача. Якщо значення схеми відмінне від імені схеми поточного користувача, то цей користувач повинен мати привілеї для створення функцій в іншій схемі.

Ім'я – ім'я функції

Параметр – необов'язковий список параметрів, які застосовуються для передачі даних в функцію і повернення інформації з неї в точку виклику.

Тип повернення – задає тип значення, що повертається функцією. Тип повернення повинен бути визначений в заголовку функції.

AUTHID – визначає, за якими дозволами буде викликатися функція: власника або поточного користувача. У першому випадку (використовується за замовчанням) застосовується модель прав власника, у другому – модель прав користувача, хто викликає функцію.

DETERMINISTIC – визначає функцію як детерміновану, тобто значення повернення повністю визначається значеннями її аргументів. Якщо включити цю секцію, ядро SQL зможе оптимізувати виконання функції при її виклику в запитах.

PARALLEL_ENABLE – використовується для оптимізації і дозволяє функції виконуватися паралельно в разі, коли вона викликається з команди **SELECT**.

PIPELINED – вказує, що результат табличній функції повинен повертатися в ітеративному режимі за допомогою команди **PIPE ROW**.

RESULT_CACHE – вказує, що вхідні дані і результат виклику функції повинен бути збережений в кеші результатів. (Можливо лише в Oracle 11g і вище).

ACCESSIBLE BY (*Oracle Database 12c*) – обмежує доступ до функції програмними модулями, перерахованими в круглих дужках.

EXTERNAL – визначає функцію з «зовнішньою реалізацією» - тобто написану на мові С.

AGGREGATE – використовується при визначенні агрегатних функцій.

Оголошення – оголошення локальних ідентифікаторів цієї функції. Якщо оголошення відсутні, між ключовими словами **IS** і **BEGIN** не буде жодних виразів.

Виконувані команди – команди, що виконуються функцією при виклику. Між ключевими словами **BEGIN** і **END** або **EXCEPTION** повинна знаходитися принаймні одна виконувана команда.

Обробники виключень – необов'язкові обробники виключень для функції. Якщо функція не виконує жодних виключень, слово **EXCEPTION** можна опустити і завершити виконуваний розділ ключовим словом **END**.

Функції. Приклад.

```

FUNCTION tot_sales
(
    company_id_in IN company.company_id%TYPE,
    statusjn IN order.status_code%TYPE:=NULL
)
RETURN NUMBER
IS
    status intorder.status_code%TYPE:=UPPER(status_in);
    CURSOR sales_cur (IN status_code%TYPE) IS
        SELECT SUM (amount*discount) FROM item
        WHERE EXISTS
        (
            SELECT 'X' FROM order
            WHERE order.order_id=item.order_id AND
                company_id=company_id_in AND
                status_code LIKE statusjn
        );
    returnn_value NUMBER;
BEGIN
    OPEN sales_cur (statusjnt);
    FETCH sales_cur INTO return_value;
    IF sales_cur%NOTFOUND
        THEN CLOSE sales_cur; RETURN NULL;
    ELSE CLOSE sales_cur; RETURN return_value;
END tot_sales;

```

Заголовок функції

Оголошення

Виконуваний розділ

Функції. Тип повернення.

Тип повернення

Функція PL/SQL може повертати дані практично будь-якого типу, підтримуваного PL/SQL, – від скалярів (одиничних значень на зразок дати або рядка) до складних структур: колекцій, об'єктних типів, курсорних змінних, тощо.

Повернення рядку:

```
FUNCTION favorite_nickname
(
    name_in IN VARCHAR2
)
RETURN VARCHAR2
IS
BEGIN
...
END favorite_nickname;
```

Повернення курсорної змінної:

```
TYPE overdue_rt IS RECORD
(
    isbn books.isbn% TYPE,
    days_overdue PLS_INTEGER
);
TYPE overdue_rct IS REF CURSOR RETURN overdue_rt;
FUNCTION overdue_info
(
    username_in IN lib_users.username% TYPE
)
RETURN overdue_rct
IS
BEGIN
...
END overdue_info;
```

```
FUNCTION onerow
(
    isbn_in IN books.isbn% TYPE
)
RETURN books% ROWTYPE
IS
BEGIN
...
END onerow;
```

Повернення запису таблиці:

Виклик функції.

Функція може бути викликана з будь-якої частини виконуваної команди PL/SQL, де допускається використання виразу.

```
DECLARE
```

```
    v_nickname VARCHAR2 (100) := favorite_nickname ('Steven');
```

```
...
```

```
DECLARE
```

```
    TYPE pet_t IS OBJECT
```

```
(
```

```
        tag_no INTEGER,  
        name  VARCHAR2 (60),  
        breed VARCHAR2(100),  
        dob   DATE,
```

```
    MEMBER FUNCTION age RETURN NUMBER
```

```
);
```

```
my_parrot pet_t: =pet_t
```

```
(
```

```
    1001,
```

```
'Mercury',
```

```
'African Grey',
```

```
    TO_DATE ('09 / 23/1996 ',' MM / DD / YYYY ')
```

```
);
```

```
BEGIN
```

```
    IF my_parrot.age () < INTERVAL '50' YEAR
```

```
    THEN DBMS_OUTPUT.PUT_LINE ( 'Still a youngster!');
```

```
END IF;
```

```
...
```

```
END;
```

```
DECLARE
```

```
    my_first_book books% ROWTYPE;
```

```
BEGIN
```

```
    my_first_book: = book_info.onerow ('1-56592-335-9');
```

```
...
```

```
END;
```

Виклик функції.

DECLARE

```
    1_name employees.last_name% TYPE;
BEGIN
    SELECT last_name INTO 1_name FROM employees
    WHERE employee_id = hr_info_pkg.employee_of_the_month ('FEBRUARY');
    ...
END;
```

```
CREATE VIEW young_managers AS
    SELECT managers.employee_id AS manager_employee_id
    FROM employees managers
    WHERE most_reports_before_manager
    (
        CURSOR
        (
            SELECT reports.hire_date FROM employees reports
            WHERE reports.manager_id = managers.employee_id
        ),
        managers.hire_date
    ) = 1;
```



В PL/SQL, неможливо просто проігнорувати значення повернутиня функції, навіть якщо воно не буде використане. Наприклад, для виклику функції: **BEGIN favorite_nickname ('Steven');** **END;** буде видана помилка **PLS-00221: 'FAVORITE_NICKNAME' is not a procedure or is undefined.** Функцію неможна використовувати так, як процедуру.

Параметри процедур і функцій.

Параметри використовуються для передачі інформації між модулем і блоком виклику. Параметри модуля є частиною його заголовка (або сигнатури). Вони не менш важливі компоненти модуля, ніж виконувані команди, що складають його тіло.



Кількість параметрів. Якщо процедура або функція має занадто мало параметрів, це знижує її універсальність. Занадто велика кількість параметрів ускладнює її повторне використання. Зазвичай, кількість параметрів визначається вимогами програми, але існують різні варіанти їх визначення (наприклад, кілька параметрів можна об'єднати в одну запис).



Типи параметрів. При виборі типу необхідно враховувати, для яких цілей будуть використовуватися параметри: тільки для читання, тільки для запису, для читання і запису.



Імена параметрів. Параметрам слід привласнювати прості імена, що відображають їх призначення в модулі.



Значення за замовчуванням. Значення за замовчуванням варто присвоювати завжди. Це підвищує стійкість програми. Коли параметру слід задати значення за замовчуванням, ставлять найчастіше вживане значення, а коли потрібно змусити програміста ввести певне значення, ставлять нетпове, або абсурдне щодо контексту задачі значення з обов'язковою його посттробокою.

Режими передачі параметрів.

Режим	Призначення	Використання параметрів
IN	Тільки для читання	Значення параметра може застосовуватися, але не може бути змінено в модулі. Якщо режим параметра не заданий, використовується режим IN
OUT	Тільки для запису	У модулі можна привласнити значення параметру, але не можна використовувати його. Втім, це «офіційне» визначення - насправді Oracle дозволяє читати значення параметра OUT в підпрограмі
IN OUT	Для читання і запису	У модулі можна використовувати і змінювати значення параметра

```

PROCEDURE predict_activity
(
    last_date_in IN DATE,
    task_desc_inout IN OUT VARCHAR2,
    next_date_out OUT DATE
)

```

Режими передачі параметрів.

Режим IN

Параметр **IN** дозволяє передавати значення модулю, але не може використовуватися для передачі інформації з модуля викликає блоку **PL/SQL**. В контексті викликаної програми параметри **IN** працюють як константи. Значення формального параметра **IN**, як і значення константи, не може бути змінено всередині програми. Параметру **IN** не можна присвоїти нове значення або змінити його іншим чином – компілятор видасть повідомлення про помилку. Режим **IN** використовується за умовчанням; якщо режим параметра не заданий, параметр автоматично вважається визначеним в режимі **IN**. Проте варто завжди вказувати режим параметра, щоб очікуване використання було явно зазначено в коді. Фактичним значенням параметра **IN** може бути змінна, іменована константа, літерал або складний вираз.

Режим OUT

Параметр **OUT** за змістом протилежний параметру **IN**. Параметри **OUT** можуть використовуватися для повернення значень з програми викликає блоку **PL/SQL**. Параметр **OUT** схожий з значенням повернення функції, але він включається в список параметрів, і кількість таких параметрів не обмежена (64 000). У програмі параметр **OUT** працює як неініціалізованих змінна. Він не містить ніякого значення до успішного завершення викликаної програми. Під час виконання програми всі операції присвоювання параметру **OUT** насправді виконуються з внутрішньою копією параметра. Коли програма успішно завершується і повертає управління в точку виклику, значення локальної копії переміщається в параметр **OUT** і стає доступним в блоці з точкою виклику.

Режими передачі параметрів.



*Параметрами **OUT** неможна задавати значення за замовчуванням. Значення параметра **OUT** може здаватися тільки в тілі модуля.*



*Всі операції присвоювання параметрам **OUT** скасовуються при ініціюванні виключення в програмі. Так як значення параметра **OUT** присвоюється тільки в разі успішного завершення програми, все проміжні присвоювання ігноруються. Якщо обробник НЕ перехопить виключення і не присвоїть значення параметру **OUT**, параметр залишиться незмінним. Змінна збереже значення, яке вона мала до виклику програми.*



*Фактичний параметр, відповідний формальному параметру **OUT**, не може бути константою, літералом або виразом. Інакше кажучи, він повинен підтримувати присвоювання.*

Режим **IN OUT**

У параметрі **IN OUT** можна передавати значення програмі і повернати їх на сторону виклику (або вихідне, незмінне значення, або нове значення, задане в програмі). На параметри **IN OUT** поширюються два обмеження параметрів **OUT**:

- *Параметр **IN OUT** не може бути константою, літералом або виразом.*
- *Фактичний параметр **IN OUT** повинен бути змінною.*

В цьому режимі параметр не може бути константою, літералом або виразом, тому що ці формати не можуть використовуватися **PL/SQL** як приймачі для розміщення вихідних значень. Параметри **IN OUT** можуть використовуватися в обох сторонах привласнення, тому що вони працюють як ніціалізовані змінні. **PL/SQL** не втрачає значення параметра **IN OUT** на початку виконання програми. Це значення може використовуватися в програмі будь-де.

Зв'язування формальних та фактичних параметрів.

В PL/SQL представлені два методи встановлення відповідності між формальними і фактичними параметрами:

- за позицією (неявне зв'язування);
- за іменем (явне зв'язування із зазначенням імені формального параметра і позначення $=>$).

Заголовок процедури

```
PROCEDURE calc_all (id_in IN INTEGER, total_out OUT NUMBER)
```

Виклик процедури

```
calc_all (1007, tot_sales);
```

Щоб встановити відповідність параметрів по імені, слід при виклику підпрограми явно зв'язати формальний параметр з фактичним. Для цього використовується комбінація символів $=>$:

ім'я_формального_параметра $=>$ значення_параметра

```
new_sales := total_sales (company_id_in => order_pkg.company_id, status_in =>'N');
new_sales := total_sales (status_in =>'N', company_id_in => order_pkg.company_id);
new_sales := total_sales (order_pkg.company_id, status_in =>'N');
```

```
l_new_sales := total_sales (company_id_in => order_pkg.company_id, 'N');
l_new_sales := total_sales ('N', company_id_in => order_pkg.company_id);
```

СУЧАСНІ СУБД

Лекція №14

Тема:

ORACLE PL/SQL

Збережені процедури,
пакети та тригери

(Продовження)

Пакет являє собою згрупований за певними правилами іменований набір елементів коду **PL/SOL**. Він забезпечує логічну структуру для організації програм і інших елементів **PL/SQL**: курсорів, типів даних і змінних. Пакети втілюють низку специфічних функціональних можливостей, зокрема можливість приховування логіки і даних та визначення глобальних даних, що існують протягом сеансу.

№	Перевага	Опис
1	Спрошення супроводу і розширення застосунків	Якість застосунків PL/SQL визначається не тільки їх продуктивністю, але і простою супроводу. Пакети забезпечують інкапсуляцію коду (зокрема, приховати команди SQL за інтерфейсом процедур), дають можливість визначати константи для літералів та «чарівних» чисел, і групувати логічно пов'язані функції. Пакетний підхід до проектування і реалізації скорочує кількість потенційних збоїв в застосунках.
2	Підвищення продуктивності застосунків.	У багатьох ситуаціях використання пакетів підвищує продуктивність і ефективність роботи застосунків. Визначення постійних структур даних рівня пакета дозволяє переводити статичні значення з бази даних в байт-код. Це дає можливість уникнути повторних запитів, а отже, значно прискорити отримання результату. Крім того, підсистема управління пам'яттю Oracle оптимізована для доступу до відкомпільоване коду пакетів.

Переваги використання пакетів

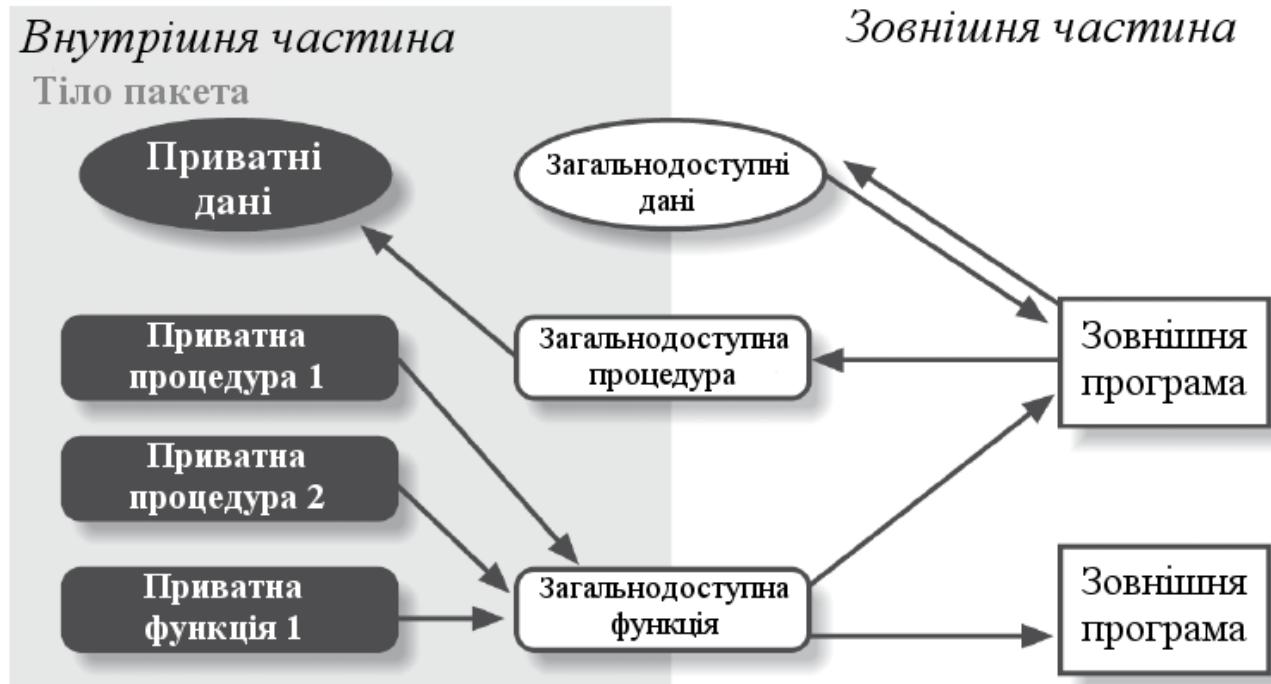
№	Перевага	Опис
3	Виправлення недоліків застосунків або вбудованих елементів	Деякі з вбудованих програмних компонентів Oracle мають недоліки. Зокрема, не кращим чином реалізовані найважливіші функції вбудованих пакетів UTL_FILE і DBMS_OUTPUT . Миритися з ними не обов'язково: можна розробити власний пакет на базі існуючого, виправивши якомога більше проблем. Наприклад, можна замінити вбудовану функцію DBMS_OUTPUT.PUT_LINE додаванням перевантаження для типу XMLType . Подібного результату можна досягти і за допомогою окремих функцій і процедур, але рішення з пакетами ефективніше..
4	Зниження необхідності в перекомпіляції коду	Пакет зазвичай складається з двох елементів: специфікації і тіла. Зовнішні програми (які не визначені в пакеті) можуть викликати тільки програми, перераховані в специфікації. Зміна і перекомпіляція тіла пакета не відбувається на роботі цих зовнішніх програм. Зниження необхідності в перекомпіляції коду є найважливішим фактором адміністрування великих обсягів програмного коду застосунка.

Приховування інформації. Приховування інформації про систему або додатку зазвичай переслідує дві мети. По-перше, можливості людини по роботі зі складними системами обмежені, отже розробник звільняється від необхідності вникати в непотрібні подробиці і може зосередитися на дійсно важливих аспектах. По-друге, приховування інформації перешкоджає доступу до закритих даних. Наприклад, розробник може викликати в своєму додатку готову функцію для обчислення деякого значення, але при цьому формула обчислень може бути таємною. Крім того, в разі зміни формулі всі модифікації будуть вноситися тільки в одному місці.

Спільні і приватні елементи. Концепція спільних і приватних елементів тісно пов'язана з концепцією приховування інформації. Загальнодоступний код визначається в специфікації пакета і доступний будь-якій схемі, яка має для цього пакета привілей **EXECUTE**. Приватний код доступний тільки в межах пакету. Зовнішні програми, що працюють з пакетом, не можуть використовувати приватний код.

Специфікація пакету. Вона містить визначення всіх загальнодоступних елементів пакету, на які можна посилатися ззовні. Специфікація нагадує великий розділ оголошень; вона не містить блоків **PL/SQL** або виконуваного коду. З добре спроектованої специфікації розробник може отримати всю необхідну для використання пакета інформацію.

Пакети. Приватність.



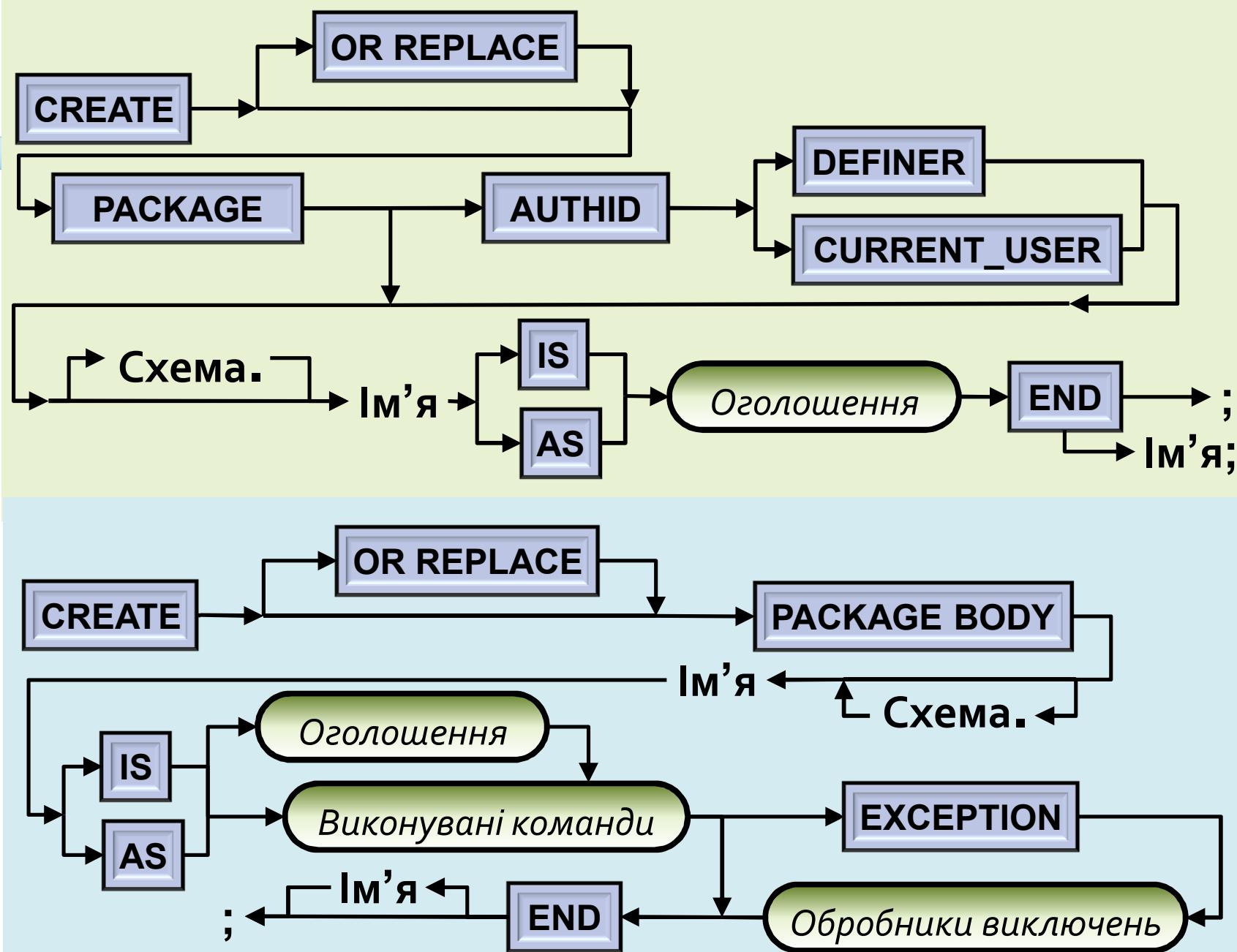
- Зовнішні програми не можуть перетинати кордон внутрішньої реалізації; інакше кажучи, зовнішня програма не може звертатися або викликати елементи, визначені в тілі пакету. Це приватні елементи, невидимі за межами пакета.
- Елементи, визначені в специфікації пакета, розташовуються по обидва боки від кордону між внутрішньою і зовнішньою частиною. Такі програми можуть викликатися зовнішньою програмою (з зовнішньої частини), вони доступні для приватних програм і в свою чергу можуть викликати або звертатися до всіх інших елементів пакету.
- Якщо виявиться, що об'єкт, який раніше був приватним (наприклад, модуль або курсор), треба зробити загальнодоступним, необхідно додати його в специфікацію і перекомпілювати пакет. Після цього об'єкт стане доступним поза межами пакета.
- Загальнодоступні елементи пакета забезпечують єдиний шлях до внутрішньої частини. В цьому відношенні специфікація пакета діє як керувальний механізм для пакета в цілому.

Тіло пакета. Тут знаходиться весь код, який необхідний для реалізації елементів, визначених у специфікації пакета. Тіло може містити відсутні в специфікації особисті елементи, на які не можна посилатися ззовні пакета, зокрема оголошення змінних і визначення пакетних модулів. Крім того, в тілі пакета може перебувати виконуваний (ініціалізаційний) розділ, який виконується тільки один раз для ініціалізації пакету.

Ініціалізація. Концепція ініціалізації полягає в тому, що ініціалізується не окрема змінна, а весь пакет шляхом виконання коду довільної складності. При цьому Oracle стежить за тим, щоб пакет ініціалізувати тільки один раз за сесію.

Сталість протягом сесії. Концепція сталості (або зберігання) полягає в тому, що при підключені до Oracle і виконанні програми, яка присвоює значення змінній рівня пакета (тобто змінній, оголошений в пакеті поза програмами, що містяться в ньому), ця змінна зберігає значення протягом усього сесії, навіть якщо виконання програми, що привласнила це значення, завершується.

Пакети. Синтаксис.



Пакети. Приклад.

```

CREATE OR REPLACE PACKAGE q_test
AS
    TYPE vrbn_info_rct IS REF CURSOR
    RETURN vyrobnyky%ROWTYPE;
    TYPE pstvk_info IS RECORD (g postavky%ROWTYPE);
    p_pstvk pstvk_info;
    q_pstvk postavky%ROWTYPE;
    q_n VARCHAR2(40);
    PROCEDURE q_proc(n IN NUMBER, q OUT VARCHAR2);
    FUNCTION q_func(k IN NUMBER, c IN VARCHAR2)
    RETURN NUMBER;
    FUNCTION q_func_tbl(k IN NUMBER, c IN VARCHAR2)
    RETURN vrbn_info_rct;
END q_test;
/

```

```

CREATE OR REPLACE PACKAGE BODY q_test
AS
    q_pstvk_prv pstvk_info;
    q_prv VARCHAR2(200);

    PROCEDURE q_proc(n IN NUMBER, q OUT VARCHAR2)
    IS BEGIN NULL; END q_proc;

    PROCEDURE q_proc_prv(n IN NUMBER, q OUT VARCHAR2)
    IS BEGIN NULL; END q_proc_prv;

    FUNCTION q_func(k IN NUMBER, c IN VARCHAR2)
    RETURN NUMBER IS BEGIN RETURN NULL; END q_func;

    FUNCTION q_func_tbl(k IN NUMBER, c IN VARCHAR2)
    RETURN vrbn_info_rct
    IS
        q_ret vrbn_info_rct;
    BEGIN
        OPEN q_ret FOR SELECT * FROM vyrobnyky;
        RETURN q_ret;
    END q_func_tbl;

    FUNCTION q_func_prv(k IN NUMBER) RETURN NUMBER
    IS BEGIN RETURN NULL; END q_func_prv;
END q_test;
/

```

Пакети. Правила побудови. Специфікація

Елементи практично будь-якого типу - числа, виключення, типи, колекції, тощо – можуть оголошуватися на рівні пакета (тобто такі елементи не належать конкретним процедурам або функціям цього пакету). Такі дані називаються даними рівня пакетів. У загальному випадку оголошувати змінні в специфікаціях пакетів не рекомендується, хоча оголошення констант на рівні пакета цілком прийнятні. У пакеті (як в специфікації, так і в тілі) не можна оголошувати курсорні змінні (Типу **REF CURSOR**), оскільки вони не можуть зберігати своє значення протягом сеансу.

У специфікації допускається оголошення типів для будь-яких структур даних: колекцій, записів або курсорних змінних.

У специфікації можна оголошувати процедури і функції, але необхідно вказувати тільки їх заголовки тобто визначення процедури або функції до ключово-го слова **IS** або **AS**. Заголовок повинен завершуватися символом крапки з комою «;» .

У специфікацію пакета можна включати явні курсори. Вони можуть бути представлені в одній з двох форм: **SQL - запит** або *є частиною оголошення курсору*, або ховається в тілі пакета (тоді в оголошенні присутній тільки вираз **RETURN**).

Якщо в специфікації пакета оголошуються процедури або функції або пакетний курсор без запиту, то тіло пакета повинно обов'язково включати реалізацію цих елементів.

Специфікація пакету може містити умову **AUTHID**, що визначає, як будуть вирішуватися посилання на об'єкти даних: відповідно до привілейв власника пакета (**AUTHID DEFINER**) або того, хто його викликає (**AUTHID CURRENT_USER**).

Після команди **END** в кінці специфікації пакета можна розмістити необов'язкову мітку, що ідентифікує пакет: **END my_package;**

Тіло пакета необхідно в тому випадку, якщо істинні хоча б деякі з наступних умов:

- *Специфікація пакету містить оголошення курсору з секцією RETURN.* В цьому випадку команда **SELECT** повинна бути вказана в тілі пакету.
- *Специфікація пакету містить оголошення процедури або функції.* У цьому випадку реалізація модуля повинна бути завершена в тілі пакету.
- *При ініціалізації пакету повинен виконуватися код, вказаний в ініціалізацій розділі.* Специфікація пакета не підтримує виконуваний розділ (виконувані команди в блоці BEGIN-END); ці команди можуть перебувати тільки в тілі пакету.

Пакети. Правила побудови. Тіло пакета.

Тіло пакета може містити розділ оголошень, виконуваний розділ і розділ виключення. Розділ оголошень містить повну реалізацію всіх курсорів і програм, які визначаються в специфікації, а також визначення всіх приватних елементів (не вказані в специфікації). Розділ оголошень може бути порожнім – за умови, що в тілі пакету присутній розділ ініціалізації.

Виконуваний розділ пакету також називається розділом ініціалізації; він містить додатковий код, що виконується при ініціалізації пакету в сеансі.

У розділі виключень обробляються всі виключення, ініційовані в розділі ініціалізації. Розділ виключень розташовується в кінці тіла пакета тільки в тому випадку, якщо визначено розділ ініціалізації.

Тіло пакета може мати наступну структуру: тільки розділ оголошень; тільки виконуваний розділ; виконуваний розділ і розділ виключень; розділ оголошень, виконуваний розділ і розділ виключень.

Секція **AUTHID** не може входити в тіло пакету; вона повинна розміщуватися в специфікації пакета. Все, що оголошено в специфікації, може використовуватися в тілі пакету.

Для тіла і специфікації пакета діють одні правила і обмеження оголошень структур даних – наприклад, неможливість оголошення курсорних змінних.

За командою **END** тіла пакета може слідувати необов'язкова мітка з ім'ям пакета: **END my_package;**

Пакети. Ініціалізація пакета.

Пакет може містити структури даних, що зберігаються протягом всього сеансу. Коли в ході сеансу вперше відбувається звернення до пакету (викликається оголошена в ньому програма, читається або записується значення змінної або використовується оголошений в пакеті тип), Оракл ініциалізує його, виконуючи такі дії:

- *Створення екземплярів даних рівня пакетів (значення змінних і констант).*
- *Присвоєння змінним і константам значень за замовчуванням, зазначених в оголошеннях.*
- *Виконання блоку коду, що міститься в ініціалізацій розділі.*

Оракл виконує всі ці дії тільки один раз за сеанс і тільки тоді, коли виникне безпосередня необхідність в цій інформації.



Пакет може бути повторно ініціалізований в ході сеансу, якщо він був перекомпільований з моменту останнього використання або було виконане скидання стану всього сеансу, на що вказує помилка ORA-04068.

Тригери

Тригери - це іменовані програмні блоки, які виконуються у відповідь на події, що відбуваються в базі даних. Вони відносяться до числа важливих елементів застосунків *Oracle* і використовуються для виконання дій, спрямованих на підтримку цілісності і структурної узгодженості баз даних.

№	Дія	Опис
1	Перевірка внесених в таблиці змін	Оскільки логіка перевірки даних безпосередньо пов'язана з конкретним об'єктом бази даних, тригери гарантують її суворе виконання і дотримання.
2	Автоматизація супроводу бази даних	Починаючи з Oracle8i можна було використовувати тригери, які автоматично виконуються при завантаженні і вивантаженні бази даних, для виконання операцій ініціалізації і очищення. Це значно зручніше, ніж створювати для цих операцій зовнішні по відношенню до бази даних сценарії
3	Узгодження обмежень з операціями адміністрування	За допомогою тригерів можна перевірити, чи допускається виконання певної операції над конкретним об'єктом бази даних (наприклад, видалення або модифікація таблиці). Коли правила перевірки реалізовані у вигляді тригерів, обійти їх дуже важко, якщо взагалі можливо.

Тригери

Події, з якими можна зв'язувати тригери

№	Подія	Опис
1	Команди DML	Тригери DML запускаються у відповідь на внесення, оновлення та видалення запису таблиці бази даних. Їх можна використовувати з метою перевірки значень, встановлених за замовчуванням, виконання аудиту змін і навіть заборони певних команд DML .
2	Команди DDL	Тригери DDL запускаються у відповідь на виконання команд DDL - наприклад, при створенні таблиці. З їх допомогою можна виконувати аудит і забороняти певні операції.
3	Події бази даних	Тригери подій бази даних використовуються при запуску і зупинці бази даних, при підключення і відключення сервера, а також при виникненні помилок Oracle. Починаючи з Oracle8i вони також дозволяють отримувати інформацію про операції з базою даних.
4	Тригери INSTEAD OF	Тригери INSTEAD OF є альтернативою триггерам DML . Вони запускаються безпосередньо перед операціями внесення, оновлення, видалення, і їх код визначає, які дії слід виконати замість відповідної операції DML . Тригери INSTEAD OF керують операціями над представленнями, але не над таблицями. З їх допомогою можна перетворювати неоновлювані представлення в оновлювані, змінюючи при необхідності їх поведінку.

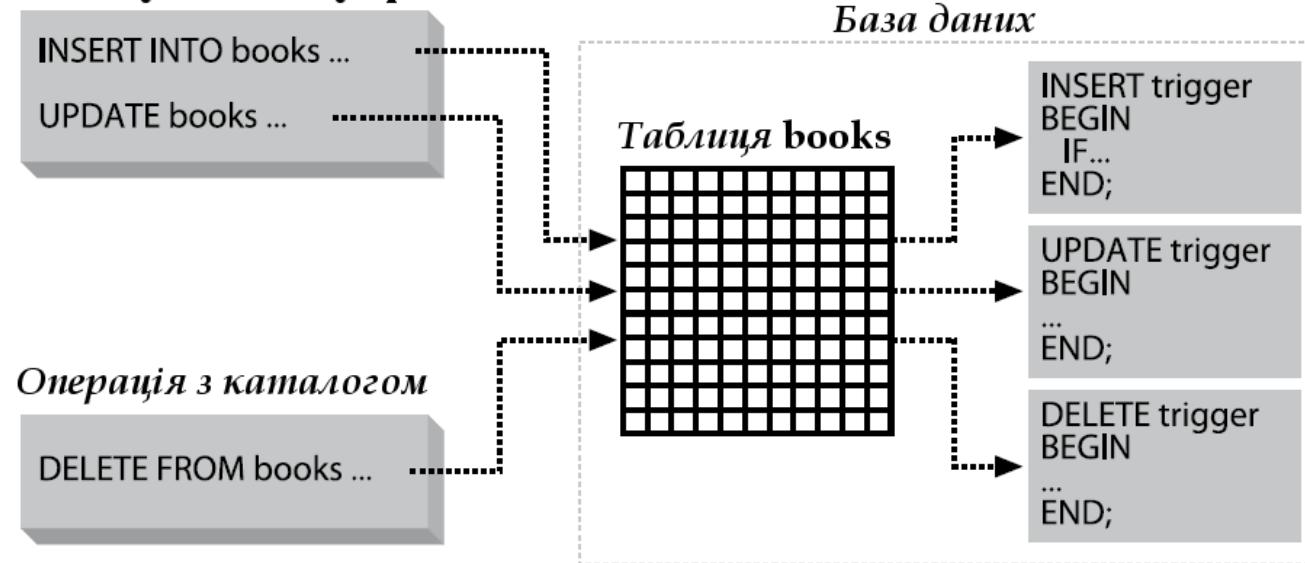
Події, з якими можна зв'язувати тригери (продовження)

№	Подія	Опис
5	Призупинені команди	у Oracle9i введена концепція призупинених команд. Якщо в ході виконання команди виникла проблема доступності простору (недостатньо табличного простору або вичерпана квота), Oracle може перевести її в режим призупинення до тих пір, поки проблема не буде вирішена. З цією подією можна пов'язати тригер, який автоматично повідомляє про проблему або навіть самостійно усуває її.

Тригери рівня команд DML

Тригери рівня команд DML активізуються після внесення, оновлення або видалення записів конкретної таблиці. Це найпоширеніший тип тригерів. Решта тригерів використовуються переважно адміністраторами бази даних. У **Oracle11i** з'явилася можливість об'єднання декількох тригерів DML в один **складений тригер**.

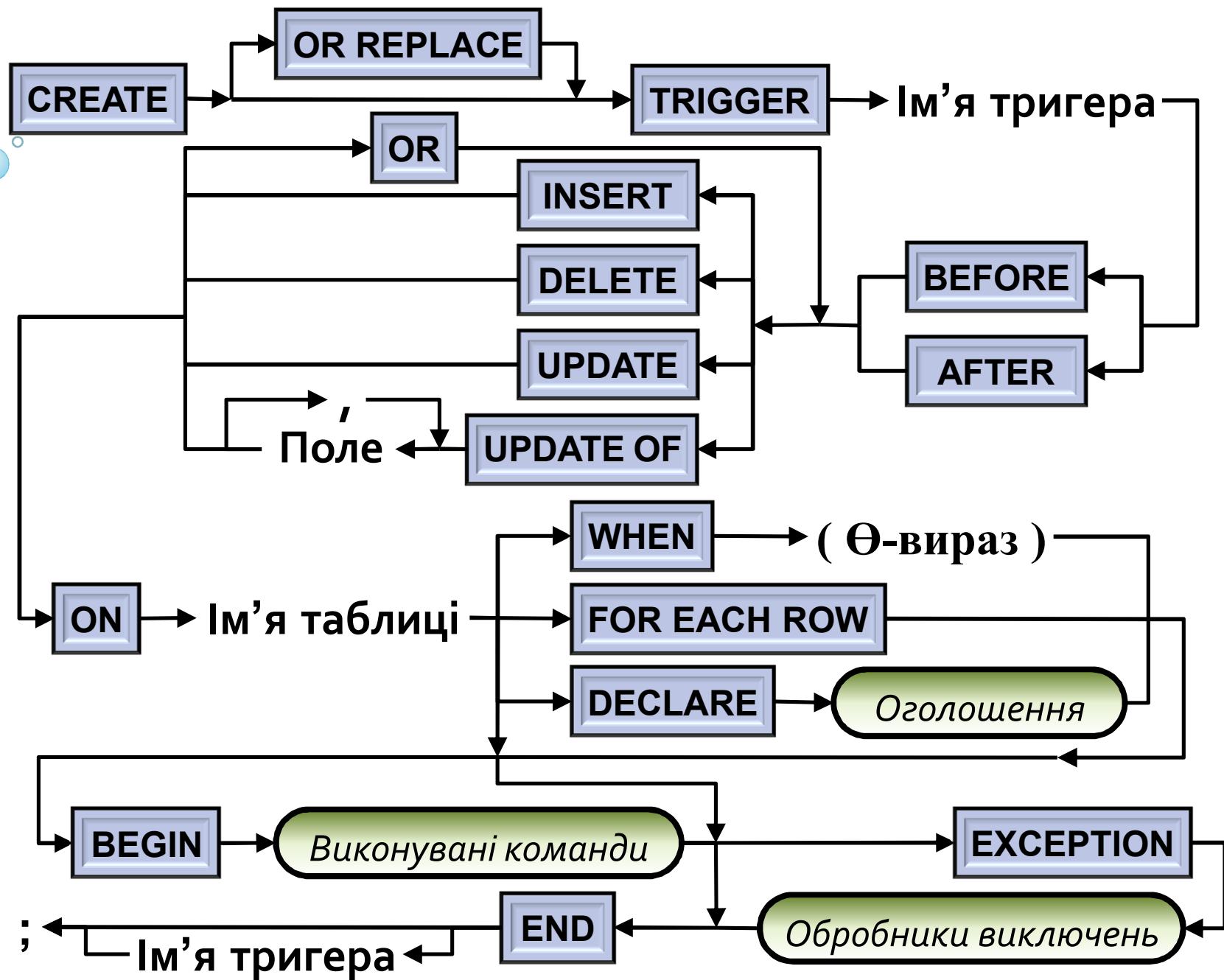
Застосунок для управління бібліотекою



При проектуванні тригера необхідно визначити:

- Як тригер буде запускатися – по одному разу дляожної команди SQL або для кожного модифікованого запису.
- Коли саме повинен викликатися тригер – до або після виконання операції над записами.
- Для яких операцій повинен спрацьовувати тригер – внесення, оновлення, видалення або їх певної комбінації.

Тригери рівня команд DML. Синтаксис.



Тригери рівня команд DML.Приклади.

```
CREATE OR REPLACE
TRIGGER bef_ins_ceo_comp
BEFORE INSERT
ON ceo_compensation
FOR EACH ROW
BEGIN
    PRAGMA AUTONOMOUS_TRANSACTION;
    INSERT INTO ceo_comp_history
    VALUES
    (
        :NEW.name,
        :OLD.compensation,
        :NEW.compensation,
        'AFTER INSERT', SYSDATE
    );
    COMMIT;
END;
```

```
CREATE OR REPLACE
TRIGGER validate_employee_changes
AFTER INSERT OR UPDATE
ON employees
FOR EACH ROW
BEGIN
    check_date(:NEW.hire_date);
    check_email(:NEW.email);
END;
```

Основні концепції тригерів.

Тригер BEFORE. Викликається до внесення будь-яких змін (наприклад, BEFORE INSERT).

Тригер AFTER. Виконується після окремої команди SQL, яка може обробляти одну або більше записів бази даних (наприклад, AFTER UPDATE).

Тригер рівня команди. Виконується для команди SQL в цілому (яка може обробляти одну або кілька рядків бази даних).

Тригер рівня запису. Виконується для окремого запису оброблюваної командою SQL. Якщо, припустимо, таблиця містить 1000 рядків, то наступна команда UPDATE модифікує всі ці рядки. І якщо для таблиці визначено тригер рівня запису, він буде виконаний 1000 разів.

Псевдозапис NEW. Структура даних з ім'ям NEW має такий же самий вигляд і (майже) такі ж властивості, як запис PL/SQL. Цей псевдозапис доступний тільки всередині тригерів поновлення і вставки; він містить значення модифікованого запису після внесення змін.

Псевдозапис OLD. Структура даних з ім'ям OLD має такий же самий вигляд і (майже) такі ж властивості, як запис PL/SQL. Цей псевдозапис доступний тільки всередині тригерів поновлення і вставки; він містить значення модифікованого запису до внесення змін.

Секція WHEN. Частина тригера DML, що визначає умови виконання коду тригера (і дозволяє уникнути зайвих операцій).