Database management systems

Lecture 1

Prof. Univ. Dr. Diaconita Vlad

PL/SQL Language

- Is the Procedural Language extension to SQL
- Includes SQL statements for data managing and transaction support
- Includes PL/SQL specific statements
- Is a 3GL language

```
Select first_name into …  -SQL
IF...THEN -PLSQL
 x:=y+1;      -PLSQL
 update employees … - SQL
ELSE
   update employees … - SQL
END IF;
```

PL/SQL Language

- Is a block-structured language. The PL/SQL code block helps modularize code by using:
  - Anonymous blocks
  - Procedures and functions
  - Packages
  - Database triggers
- The benefits of using modular program constructs are:
  - Easy maintenance
  - Improved data security and integrity
  - Improved performance
  - Improved code clarity

PL/SQL Language

- Supports defining variables and constants of all the SQL data types
- User defined Types can also be constructed
- Introduces PL/SQL specific data types (e.g. PLS_INTEGER, BOOLEAN)
- Supports the fundamental control structures
- Supports the processing of a data set, row by row

PL/SQL Operators

Operator	Characteristics
+, -, *, /, ** (power operator)	Arithmetic
AND, OR, NOT	Logical
<, >, =, >=, <=, <>, !=	Comparison
BETWEEN ... AND ...	Comparison
IN	Comparison
LIKE	Comparison
IS NULL	Check if NULL
||	Concatenation
@	Connecting at distance (database links)
& **or&&**	Addressing substitution variables

PL/SQL Block

- Anonymous blocks:
  - Form the basic PL/SQL block structure
  - Initiate PL/SQL processing tasks from applications
  - Can be nested within the executable section of any PL/SQL block

```
[DECLARE      -- Declaration Section (Optional)
  variable declarations; ... ]
BEGIN         -- Executable Section (Mandatory)
  SQL or PL/SQL statements;
[EXCEPTION    -- Exception Section (Optional)
  WHEN exception THEN statements; ]
END;          -- End of Block (Mandatory)
```

Procedures

- Procedures are named PL/SQL blocks that perform a sequence of actions.

```
CREATE PROCEDURE getemp IS  -- header
  emp_id  employees.employee_id%type;
  lname   employees.last_name%type;
BEGIN
  emp_id := 100;
  SELECT last_name INTO lname
  FROM EMPLOYEES
  WHERE employee_id = emp_id;
  DBMS_OUTPUT.PUT_LINE('Last name: '||lname);
END;
/
```

Functions

- Functions are named PL/SQL blocks that perform a sequence of actions and return a value. A function can be invoked from:
  - Any PL/SQL block
  - A SQL statement (subject to some restrictions)

```
CREATE FUNCTION avg_salary RETURN NUMBER IS
  avg_sal employees.salary%type;
BEGIN
  SELECT AVG(salary) INTO avg_sal
  FROM EMPLOYEES;
  RETURN avg_sal;
END;
/
```

Packages

- PL/SQL packages have a specification and an optional body. Packages group related subprograms together.

```
CREATE PACKAGE emp_pkg IS
  PROCEDURE getemp;
  FUNCTION avg_salary RETURN NUMBER;
END emp_pkg;
/
CREATE PACKAGE BODY emp_pkg IS
  PROCEDURE getemp IS ...
  BEGIN ... END;

  FUNCTION avg_salary RETURN NUMBER IS ...
  BEGIN ... RETURN avg_sal; END;
END emp_pkg;
/
```

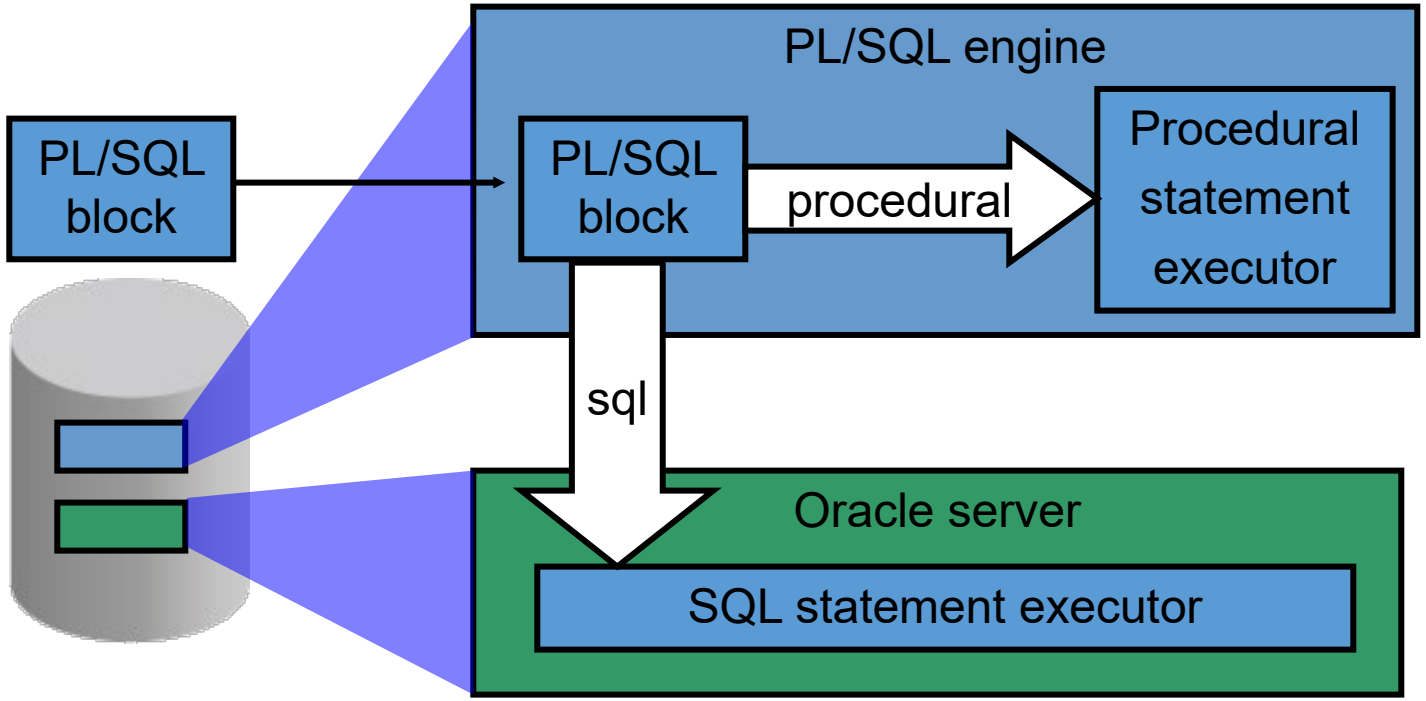
Triggers

- PL/SQL triggers are code blocks that execute when a specified application, database, or table event occurs.
  - Oracle database triggers have a specific structure.
  - Apex dynamic actions (PL/SQL) are similar to triggers

```
CREATE TRIGGER check_salary
BEFORE INSERT OR UPDATE ON employees
FOR EACH ROW
DECLARE
  c_min constant number(8,2) := 1000.0;
  c_max constant number(8,2) := 500000.0;
BEGIN
  IF :new.salary > c_max OR
     :new.salary < c_min THEN
    RAISE_APPLICATION_ERROR(-20000,
      'New salary is too small or large');
  END IF;
END;
/
```

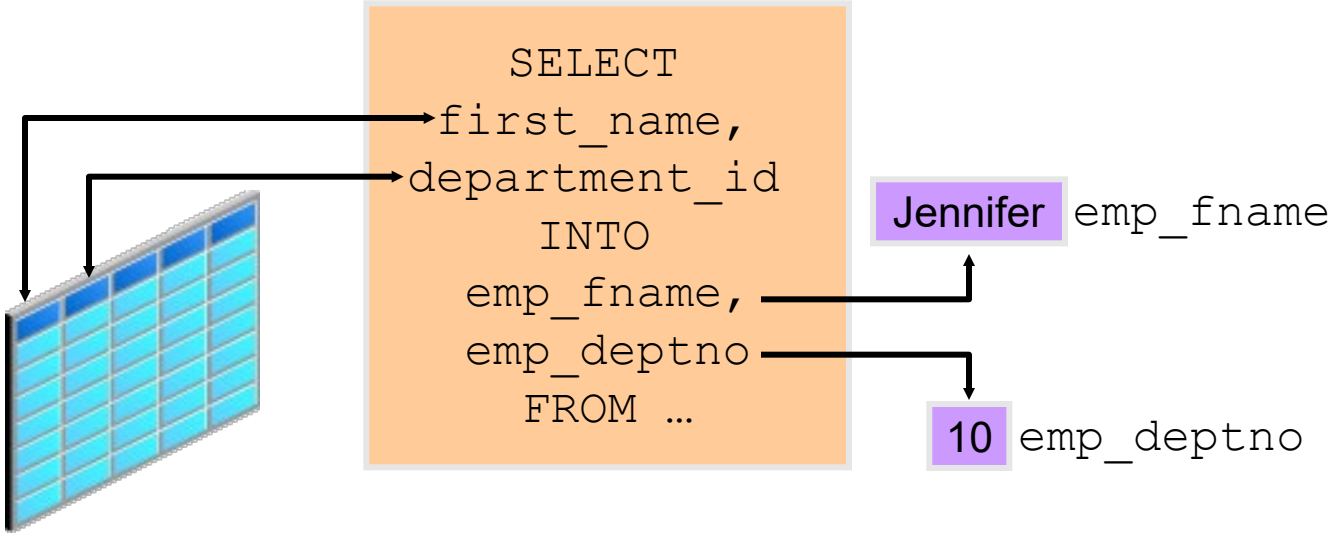
PL/SQL Execution Environment

- The PL/SQL run-time architecture:



Variables

- Variables can be used for:
  - Temporary storage of data
  - Manipulation of stored values
  - Reusability



Identifiers

- Identifiers are used for:
  - Naming a variable
  - Providing conventions for variable names
    - Must start with a letter
    - Can include letters or numbers
    - Can include special characters (such as dollar sign, underscore, and pound sign)
    - Must limit the length to 30 characters
    - Must not be reserved words

Handling Variables in PL/SQL

- Variables are:
  - Declared and initialized in the declarative section
  - Used and assigned new values in the executable section
  - Passed as parameters to PL/SQL subprograms
  - Used to hold the output of a PL/SQL subprogram

Declaring and Initializing PL/SQL Variables

Syntax

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

Examples

```
DECLARE
  emp_hiredate    DATE;
  emp_deptno      NUMBER(2) NOT NULL := 10;
  location        VARCHAR2(13) := 'Atlanta';
  c_comm          CONSTANT NUMBER := 1400;
```

Delimiters in String Literals

```
SET SERVEROUTPUT ON
DECLARE
   event VARCHAR2(15);
BEGIN
  event := q'!Father's day!';
  DBMS_OUTPUT.PUT_LINE('3rd Sunday in June is :
   '||event);
  event := q'[Mother's day]';
  DBMS_OUTPUT.PUT_LINE('2nd Sunday in May is :
   '||event);
END;
/
```

```
3rd Sunday in June is : Father's day
2nd Sunday in May is : Mother's day
PL/SQL procedure successfully completed.
```

Types of Variables

- PL/SQL variables:
  - Scalar
  - Composite
  - Reference
  - Large object (LOB)
- Non-PL/SQL variables: Bind variables, Substitution

Guidelines for Declaring and Initializing PL/SQL Variables

- Follow naming conventions.
- Use meaningful names for variables.
- Initialize variables designated as `NOT NULL` and `CONSTANT`.
- Initialize variables with the assignment operator (`:=`) or the `DEFAULT` keyword:

```
Myname VARCHAR2(20):='John';
```

```
Myname VARCHAR2(20) DEFAULT 'John';
```

- Declare one identifier per line for better readability and code maintenance.

Guidelines for Declaring PL/SQL Variables

- Avoid using column names as identifiers.

```
DECLARE
  employee_id  NUMBER(6);
BEGIN
  SELECT    employee_id
  INTO      employee_id
  FROM      employees
  WHERE     last_name = 'Kochhar';
END;
/
```

- **Use the `NOT NULL` constraint when the variable must hold a value.**

Base Scalar Data Types

- CHAR [(maximum_length)]
- VARCHAR2 (maximum_length)
- LONG
- LONG RAW
- NUMBER [(precision, scale)]
- BINARY_INTEGER
- PLS_INTEGER
- BOOLEAN
- BINARY_FLOAT
- BINARY_DOUBLE
- DATE
- TIMESTAMP
- TIMESTAMP WITH TIME ZONE
- TIMESTAMP WITH LOCAL TIME ZONE
- INTERVAL YEAR TO MONTH
- INTERVAL DAY TO SECOND

PL/SQL Blocks

```
set serveroutput on
declare
 a number(10,2);
 b number(2) default 7;
 c number(2) default 5;
begin
a:=max(b,c);
dbms_output.put_line('a='||a);
end;
/
```

- ?

```
set serveroutput on
declare
 a number(10,2);
 b number(2) default 7;
 c number(2) default 5;
begin
a:=greatest(b,c);
dbms_output.put_line('a='||a);
end;
/
```

- ?

PL/SQL Blocks

```
set serveroutput on
declare
 a varchar2(3);
 b number(2) default 7;
begin
a:=decode(b,7, 'YES','NO');
dbms_output.put_line('a='|
   |a);
end;
/
```

- ?

```
set serveroutput on
declare
 a varchar2(3);
 b number(2) default 7;
begin
a:=case when b=7 then 'YES'
   else 'NO' end;
dbms_output.put_line('a='||a)
   ;
end;
/
```

- ?

PL/SQL Blocks

```
set serveroutput on
declare
 a number(7);
 b number(7);
begin
select sum(salary),max(salary) into a,b from
   employees;
dbms_output.put_line('a='||a);
dbms_output.put_line('b='||b);
end;
/
```

?

```
set serveroutput on
declare
 a number(7);
 b number(7);
begin
select sum(salary),max(salary) from
   employees;
end;
/
```

?

PL/SQL Blocks

```
set serveroutput on
begin
select * from employees;
end;
/
```

?

```
set serveroutput on
declare
 p_first_name varchar2(20);
 p_last_name varchar2(20);
begin
select first_name,last_name into p_first_name,
   p_last_name from employees where
   employee_id=101;
dbms_output.put_line(p_first_name ||' '||
   p_last_name);
end;
/
```

?

BINARY_FLOAT and BINARY_DOUBLE

- Represent floating point numbers in IEEE 754 format
- Offer better interoperability and operational speed
- Store values beyond the values that the data type NUMBER can store
- Provide the benefits of closed arithmetic operations and transparent rounding

```
NUMBER          : 68
BINARY_FLOAT    : 35
BINARY_DOUBLE   : 33
```

%TYPE Attribute

- The %TYPE attribute
  - Is used to declare a variable according to:
    - A database column definition
    - Another declared variable
  - Is prefixed with:
    - The database table and column
    - The name of the declared variable
  - Examples:

```
emp_lname       employees.last_name%TYPE;
balance         NUMBER(7,2);
min_balance     balance%TYPE := 1000;
```

Declaring Boolean Variables

- Only the values `TRUE`, `FALSE`, and `NULL` can be assigned to a Boolean variable.
- Conditional expressions use the logical operators `AND` and `OR` and the unary operator `NOT` to check the variable values.
- The variables always yield `TRUE`, `FALSE`, or `NULL`.
- Arithmetic, character, and date expressions can be used to return a Boolean value.

Bind Variables

- Bind variables are:
  - Created in the environment
  - Also called *host* variables
  - Created with the `VARIABLE` keyword
  - Used in SQL statements and PL/SQL blocks
  - Accessed even after the PL/SQL block is executed
  - Referenced with a preceding colon

```
VARIABLE emp_salary NUMBER
BEGIN
   SELECT salary  INTO :emp_salary
   FROM  employees WHERE employee_id = 178;
END;
/
PRINT emp_salary
SELECT first_name, last_name FROM employees
WHERE salary=:emp_salary
```

Substitution Variables

- Are used to get user input at run time
- Are referenced within a PL/SQL block with a preceding ampersand
- Are used to avoid hard-coding values that can be obtained at run time

```
VARIABLE emp_salary NUMBER
SET AUTOPRINT ON
DECLARE
  empno NUMBER(6):=&empno;
BEGIN
  SELECT salary  INTO :emp_salary
  FROM  employees WHERE employee_id = empno;
END;
/
```

Bibliography

- De Elvis Foster, Shripad Godbole, Database Systems: A Pragmatic Approach, Apress, 2012

SGBD - Oracle

Tipuri de date, Structuri fundamentale de control

Data types

- Scalar types
  - SQL: CHAR, VARCHAR2, NCHAR, NVARCHAR2, RAW, NUMBER, DATE, TIMESPTAMP
  - PL/SQL: PLS_INTEGER, BINARY_INTEGER, BOOLEAN
- Composite types
  - Record: user defined, %ROWTYPE
  - **Collections:**
    - **Index-by Tables (associative arrays)**
    - **Nested Tables**
    - **Variable-size arrays (VARRAY)**
- Reference Types: REF_CURSOR
- LOB Types: BFILE, BLOB, CLOB, NLOB

PL/SQL – Data types

Data Type	Maximum Size in PL/SQL	Maximum Size in SQL
CHAR[Foot 1]	32,767 bytes	2,000 bytes
NCHAR[Footref 1]	32,767 bytes	2,000 bytes
RAW[Footref 1]	32,767 bytes	2,000 bytes
VARCHAR2[Footref 1]	32,767 bytes	4,000 bytes
NVARCHAR2[Footref 1]	32,767 bytes	4,000 bytes
LONG[Foot 2]	32,760 bytes	2 gigabytes (GB) - 1
LONG RAW[Footref 2]	32,760 bytes	2 GB
BLOB	128 terabytes (TB)	(4 GB - 1) * *database_block_size*
CLOB	128 TB	(4 GB - 1) * *database_block_size*
NCLOB	128 TB	(4 GB - 1) * *database_block_size*

Composite data types

Record:

TRUE	23-DEC-98	ATLANTA	

Collection:

PL/SQL table structure

1	SMITH
2	JONES
3	NANCY
4	TIM

VARCHAR2

PLS_INTEGER

PL/SQL table structure

1	5000
2	2345
3	12
4	3456

NUMBER

PLS_INTEGER

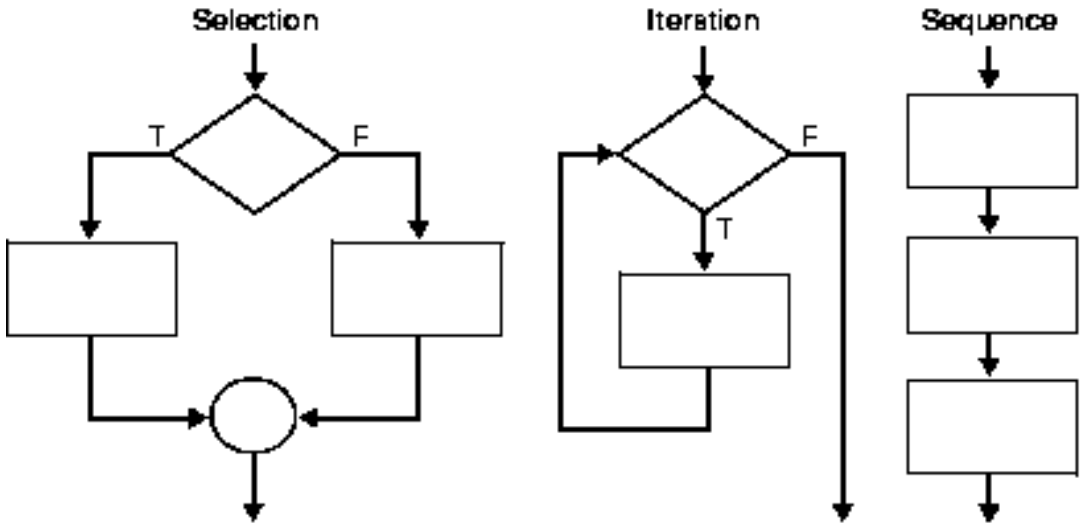
%ROWTYPE

```
SET SERVEROUTPUT ON
DECLARE
  x angajati%rowtype;
BEGIN
 select * into x from angajati where id_angajat=100;
 dbms_output.put_line(x.id_angajat);
 dbms_output.put_line(x.prenume);
 dbms_output.put_line(x.nume);
 dbms_output.put_line(x.email);

END;
/
```

PL/SQL Control Structures

- Sequence
- Decision (conditional, selection): IF, CASE
- Iterative: FOR, WHILE, LOOP



IF-THEN Statement

- IF condition THEN
- sequence_of_statements1
- ELSE
- sequence_of_statements2
- END IF

- IF condition1 THEN
- sequence_of_statements1
- ELSIF condition2 THEN
- sequence_of_statements2
- ….
- ELSIF conditionN THEN
- sequence_of_statementsN
- ELSE
- sequence_of_statements3
- END IF;

CASE Statement

- [<<label_name>>]
- CASE selector
- WHEN expression1 THEN sequence_of_statements1;
- WHEN expression2 THEN sequence_of_statements2;
- ...
- WHEN expressionN THEN sequence_of_statementsN;
- [ELSE sequence_of_statementsN+1;]
- END CASE [label_name];

LOOP and WHILE

- LOOP
- FETCH c1 INTO ...
- EXIT WHEN ....; -- exit loop if condition is true
- ...
- END LOOP;

- WHILE condition LOOP
- sequence_of_statements
- END LOOP;

FOR

- FOR counter IN [REVERSE] lower_bound..higher_bound LOOP
- sequence_of_statements
- END LOOP;

Sequential Control: GOTO and NULL Statements

- **Using GOTO statements in PL/SQL is generally discouraged as it can lead to less readable and maintainable code**. GOTO statement cannot branch into an IF statement, CASE statement, LOOP statement, sub-block or from an exception handler into the current block. The NULL statement passes control to the next statement. In a conditional construct, the NULL statement tells readers that a possibility has been considered, but no action is necessary.

```
DECLARE
  v_counter NUMBER := 0;
BEGIN
  LOOP
    v_counter := v_counter + 1;
    IF v_counter <= 5 THEN
      GOTO display_messageA;
     ELSE
     GOTO display_messageB;
    END IF;
    <<display_messageA>>
    DBMS_OUTPUT.PUT_LINE('Counter A: ' || v_counter);
    <<display_messageB>>
    DBMS_OUTPUT.PUT_LINE('Counter B: ' || v_counter);
    EXIT WHEN v_counter >= 10;
  END LOOP;
EXCEPTION
  WHEN OTHERS THEN
    NULL; -- Se inhiba propagarea exceptiei catre mediu
END;
/
```

Collections

- **Index-by Tables (associative arrays):** Lets you look up elements using arbitrary numbers and strings for subscript values. These are similar to hash tables in other programming languages;
- **Nested Tables:** Use (initially) sequential numbers as subscripts;
- **Variable-size arrays (VARRAY):** Hold a fixed Numar de elemente (although you can change the Numar de elemente at runtime). They use sequential numbers as subscripts.

Comparison

Collection Type	Number of Elements	Subscript Type	Dense or Sparse	Where Created
Associative array (or index-by table)	Unbounded	String or integer	Either	Only in PL/SQL block
Nested table	Unbounded	Integer	Starts dense, can become sparse	Either in PL/SQL block or at schema level
Variable-size array (varray)	Bounded	Integer	Always dense	Either in PL/SQL block or at schema level

Methods (selection)

- EXISTS(n)
- COUNT
- LIMIT(only for NESTED TABLES and VARRAYS)
- FIRST and LAST
- PRIOR and NEXT/PRIOR(N) and NEXT(n)
- EXTEND/EXTEND(n)/EXTEND(n,i) (only for NESTED TABLES)
- TRIM/TRIM(n) (only for NESTED TABLES andVARRAYS)
- DELETE/DELETE(n)/DELETE(n,m) (when n is specified, doesn’t work for VARRAYS)

Index-by-tables

```
SET SERVEROUTPUT ON
DECLARE
TYPE EmpTabTyp IS TABLE OF angajati%ROWTYPE INDEX BY PLS_INTEGER;
emp_tab EmpTabTyp;
BEGIN
SELECT * INTO emp_tab(200) FROM angajati WHERE id_angajat = 200;
     dbms_output.put_line('Nume='||emp_tab(200).prenume);
     dbms_output.put_line(case when emp_tab.exists(200) then 'exista' else 'nu exista' end);
     dbms_output.put_line(case when emp_tab.exists(1) then 'exista' else 'nu exista' end);
SELECT * INTO emp_tab(205) FROM angajati WHERE id_angajat = 205;
     dbms_output.put_line('Nume='||emp_tab(205).prenume);
     dbms_output.put_line('Numar de elemente='||emp_tab.count);
     emp_tab(122):=emp_tab(205);
     dbms_output.put_line('Nume='||emp_tab(122).prenume);
     dbms_output.put_line('Numar de elemente='||emp_tab.count);
     emp_tab(2005).prenume:='John';
     dbms_output.put_line('Numar de elemente='||emp_tab.count);
END;
/
```

Index-by-tables -> indexed by PLS_INTEGER

```
set serveroutput on
DECLARE
TYPE tab_ind IS TABLE OF angajati%ROWTYPE INDEX BY PLS_INTEGER;
t tab_ind;
BEGIN
SELECT * BULK COLLECT INTO t FROM angajati WHERE ROWNUM<= 50 order by salariul desc;
DBMS_OUTPUT.PUT_LINE('Numar de elemente '||t.COUNT);
t.delete(t.first);
DBMS_OUTPUT.PUT_LINE('Numar de elemente '||t.COUNT);
FOR i IN t.FIRST.. t.LAST LOOP
    DBMS_OUTPUT.PUT_LINE(t(i).prenume ||' '||t(i).nume ||'->'||i);
END LOOP;
END;
/
```

Index-by-tables -> indexed by varchar2

```
SET SERVEROUTPUT ON
DECLARE
TYPE tab_ind IS TABLE OF NUMBER INDEX BY VARCHAR2(1);
t tab_ind;
i varchar2(1);
BEGIN
t('a') := ASCII('a'); t('A') := ASCII('A');
t('b') := ASCII('b'); t('B') := ASCII('B');
t('x') := ASCII('x'); t('X') := ASCII('X');
i := t.FIRST;
WHILE i IS NOT NULL LOOP
  DBMS_OUTPUT.PUT_LINE('t('||i ||')='||t(i));
  i := t.NEXT(i);
END LOOP;
END;
/
```

Nested tables

- Within the **database (SQL)**, nested tables are column types that hold sets of values. Oracle stores the rows of a nested table in no particular order.
- When you retrieve a nested table from the database into a PL/SQL variable, the rows are given consecutive subscripts starting at 1. That gives you array-like access to individual rows.
- Nested tables do not have a pre-declared Numar de elemente
- Nested tables might not have consecutive subscripts, while arrays are always dense (have consecutive subscripts).
- **Initially**, nested tables are dense, but they can become **sparse** (have nonconsecutive subscripts). You can delete elements from a nested table using the built-in procedure DELETE. The built-in function NEXT lets you iterate over all the subscripts of a nested table, even if the sequence has gaps.

Nested tables

```
set serveroutput on
DECLARE
TYPE tab_imb IS TABLE OF NUMBER;
t tab_imb := tab_imb(1,20,3,40,5);
t_null tab_imb;
BEGIN
t.EXTEND(5);
FOR i IN 6..10 LOOP
t(i):=t(i-1)+5;
END LOOP;
DBMS_OUTPUT.PUT('Colectia are ' || t.COUNT ||' elemente: ');
FOR i IN t.FIRST..t.LAST LOOP
      DBMS_OUTPUT.PUT(t(i) || ' ');
END LOOP;
DBMS_OUTPUT.NEW_LINE;
t.delete;
IF t IS NULL THEN
    DBMS_OUTPUT.PUT_LINE('Colectia este nula');
   else
    DBMS_OUTPUT.PUT_line('Colectia are ' || t.COUNT ||'
elemente');
END IF;
t.extend(2);
t(1):=500;t(2):=700;
DBMS_OUTPUT.PUT_line('Colectia are ' || t.COUNT ||'
elemente');
t := t_null;
IF t IS NULL THEN
    DBMS_OUTPUT.PUT_LINE('Colectia este nula');
   else
    DBMS_OUTPUT.PUT_line('Colectia are ' || t.COUNT ||'
elemente');
END IF;
END;
/
```

Nested tables – bulk collect

```
set serveroutput on
DECLARE
TYPE tab_ind IS TABLE OF angajati%ROWTYPE;
  -- daca index by nu este specificat, este nested table
t tab_ind;
BEGIN
SELECT * BULK COLLECT INTO t FROM angajati WHERE ROWNUM<= 50 order by salariul desc;
DBMS_OUTPUT.PUT_LINE('Numar de elemente '||t.COUNT); -- e dens la inceput
FOR i IN t.FIRST.. t.LAST LOOP
    DBMS_OUTPUT.PUT_LINE(i||'->'||t(i).nume||' '||t(i).prenume);
    if ROUND(DBMS_RANDOM.VALUE(1,5000)) mod 2 = 0 then
      t.delete(i);
    end if;
END LOOP;
DBMS_OUTPUT.PUT_LINE('-------------------------------');
DBMS_OUTPUT.PUT_LINE('Numar de elemente '||t.COUNT); -- acum este o colectie rara;

FOR i IN t.FIRST.. t.LAST LOOP
    if t.exists(i) then
      DBMS_OUTPUT.PUT_LINE(i||'->'||t(i). nume ||' '||t(i).prenume);
    else
      dbms_output.put_line('Elementul '||i||' a fost sters');
    end if;
END LOOP;
END;
/
```

Nested tables

```
create type t_grade_celsius is table of number(5,2);
/
create table localitati(
id_loc number primary key,
denumire varchar2(100),
grade t_grade_celsius)
  nested table grade store as t_grade ;
insert into localitati values(1,'Constanta', t_grade_celsius(10,15,17,20));
insert into localitati values(2,'Bucuresti', t_grade_celsius(7,11,15,17,21));
commit;
```

Nested tables

- select column_value grade from table(select grade from localitati where id_loc=1);

- select id_loc,denumire,b.* from localitati a,table(a.grade) b;

- update localitati set grade=t_grade_celsius(10,15,17,20,27,21,21) where id_loc=1;

Varrays

- A varray has a maximum size, which you specify in its type definition. Its index has a fixed lower bound of 1 and an upper bound.
- A varray can contain a varying Numar de elemente, from zero (when empty) to the maximum specified in its type definition.
- Are dense, every elements has an index that shows it’s position
- The DELETE method only works for deleting **all** the elements in the array

Varray

```
DECLARE
TYPE tab_vec IS VARRAY(10) OF NUMBER;
t tab_vec := tab_vec();
BEGIN
FOR i IN 1..10 LOOP
  t.EXTEND;
  t(i):=i;
END LOOP;
DBMS_OUTPUT.PUT('Colectia are ' || t.COUNT ||'
elemente: ');
FOR i IN t.FIRST..t.LAST LOOP
    DBMS_OUTPUT.PUT(t(i) || ' ');
END LOOP;
DBMS_OUTPUT.NEW_LINE;
FOR i IN 1..10 LOOP
  IF i mod 2 = 1 THEN t(i):=null;
  END IF;
END LOOP;
DBMS_OUTPUT.PUT('Colectia are ' || t.COUNT ||'
elemente: ');
FOR i IN t.FIRST..t.LAST LOOP
    DBMS_OUTPUT.PUT(nvl(t(i), 0) || ' ');
END LOOP;
DBMS_OUTPUT.NEW_LINE;
--t.delete(1); -- wrong number or types of arguments
in call to 'DELETE'
t.DELETE;
DBMS_OUTPUT.PUT_LINE('Colectia are '||t.COUNT||'
elemente');
END;
/
```

VARRAYS – Example SQL

- CREATE OR REPLACE TYPE t_vect_proiecte AS VARRAY(10) OF varchar2(32);
- /
- CREATE TABLE echipe

(id_echipa NUMBER(4) PRIMARY KEY,

denumire VARCHAR2(40),

proiecte t_vect_proiecte);

- insert into echipe values (1,'Inspire',t_vect_proiecte('HR Application','Trading Application'));
- insert into echipe values (2,'Excel',t_vect_proiecte('Client Support','HR Application','Trading Application'));
- insert into echipe values (3,'Global',t_vect_proiecte('Advanced Support','Big Data Analitics','Cloud Migration'));

VARRAYS – Example SQL

- update echipe set proiecte=t_vect_proiecte('Big Data Analitics','Cloud Migration') where id_echipa=3;
- select id_echipa,denumire,b.* from echipe a, table(a.proiecte) b;

```
select id_echipa,denumire,b.*  from echipe a, table(a.proiecte) b;
```

Script Output | Query Result

SQL | All Rows Fetched: 7 in 0,004 seconds

	ID_ECHIPA	DENUMIRE	COLUMN_VALUE
1	1	Inspire	HR Application
2	1	Inspire	Trading Application
3	2	Excel	Client Support
4	2	Excel	HR Application
5	2	Excel	Trading Application
6	3	Global	Big Data Analitics
7	3	Global	Cloud Migration

```
DESC ANGAJATI
SET SERVEROUTPUT ON
DECLARE
  V_NUME  VARCHAR2(25);
  V_PRENUME ANGAJATI.PRENUME%TYPE;
  V_SALARIUL ANGAJATI.NUME%TYPE;
BEGIN
  SELECT NUME,PRENUME,SALARIUL INTO V_NUME,V_PRENUME,V_SALARIUL FROM ANGAJATI
WHERE ID_ANGAJAT=200;
  DBMS_OUTPUT.PUT_LINE(V_NUME||' '||V_PRENUME||' '||V_SALARIUL);
END;
/

DECLARE
  TYPE R_ANG IS RECORD(
  V_NUME  VARCHAR2(25),
  V_PRENUME ANGAJATI.PRENUME%TYPE,
  V_SALARIUL ANGAJATI.NUME%TYPE);
  V_ANG R_ANG;

BEGIN
  SELECT NUME,PRENUME,SALARIUL INTO V_ANG FROM ANGAJATI WHERE ID_ANGAJAT=200;
  DBMS_OUTPUT.PUT_LINE(V_ANG.V_NUME||' '||V_ANG.V_PRENUME||' '||V_ANG.V_SALARIUL);
END;
/



DECLARE
  V_ANG ANGAJATI%ROWTYPE;
BEGIN
  SELECT * INTO V_ANG FROM ANGAJATI WHERE ID_ANGAJAT=200;
  DBMS_OUTPUT.PUT_LINE(V_ANG.NUME||' '||V_ANG.PRENUME||' '||V_ANG.SALARIUL);
END;
/

DECLARE
  TYPE T_VEC IS TABLE OF ANGAJATI%ROWTYPE INDEX BY PLS_INTEGER;
  C T_VEC;
BEGIN
  SELECT * BULK COLLECT INTO C FROM ANGAJATI ORDER BY 1;
  DBMS_OUTPUT.PUT_LINE(C(1).NUME||' '||C(1).PRENUME||' '||C(1).SALARIUL);
  DBMS_OUTPUT.PUT_LINE(C(10).NUME||' '||C(10).PRENUME||' '||C(10).SALARIUL);
  DBMS_OUTPUT.PUT_LINE(C(107).NUME||' '||C(107).PRENUME||' '||C(107).SALARIUL);

END;
/

SELECT * FROM ANGAJATI ORDER BY 1;

-- SA SE PARCURGA UN SIR DE CIFRE SI SA SE AFISEZE SUMA CIFRELOR PARE SI SUMA CIFRELOR
IMPARE
```

```
DECLARE
   S VARCHAR2(50):='423908TY4902A3849#2798$6328';
   S_PARE PLS_INTEGER:=0;
   S_IMPARE PLS_INTEGER DEFAULT 0;
   C CHAR;
BEGIN
  FOR I IN 1..LENGTH(S) LOOP
   BEGIN
    C:=SUBSTR(S,I,1);
    IF C MOD 2 = 0 THEN
      S_PARE:=S_PARE+C;
    ELSE
      S_IMPARE:=S_IMPARE+C;
    END IF;
      EXCEPTION
        WHEN OTHERS THEN NULL;
   END;
  END LOOP;
  DBMS_OUTPUT.PUT_LINE('SUMA PARE='||S_PARE);
  DBMS_OUTPUT.PUT_LINE('SUMA IMPARE='||S_IMPARE);

END;
/

SELECT NULL+7+5+3 FROM DUAL;



DECLARE
   S VARCHAR2(50):='423908TY4902A3849#2798$6328';
   S_PARE PLS_INTEGER:=0;
   S_IMPARE PLS_INTEGER DEFAULT 0;
   C CHAR;
   I PLS_INTEGER:=100;
BEGIN
  DBMS_OUTPUT.PUT_LINE(I);
  FOR I IN 1..LENGTH(S) LOOP
   -- I:=I+1;
    C:=SUBSTR(S,I,1);
    IF C NOT BETWEEN '1' AND '9' THEN
      NULL;
    ELSIF C MOD 2 = 0 THEN
      S_PARE:=S_PARE+C;
    ELSE
      S_IMPARE:=S_IMPARE+C;
    END IF;

  END LOOP;

  DBMS_OUTPUT.PUT_LINE('SUMA PARE='||S_PARE);
  DBMS_OUTPUT.PUT_LINE('SUMA IMPARE='||S_IMPARE);
  I:=I+1;
  DBMS_OUTPUT.PUT_LINE(I);
```

```
END;
/

DECLARE
   S VARCHAR2(50):='423908TY4902A3849#2798$6328';
   S_PARE PLS_INTEGER:=0;
   S_IMPARE PLS_INTEGER DEFAULT 0;
   C CHAR;
   I PLS_INTEGER:=1;
BEGIN
  WHILE (I<=LENGTH(S)) LOOP
    C:=SUBSTR(S,I,1);
    IF C NOT BETWEEN '1' AND '9' THEN
      NULL;
    ELSIF C MOD 2 = 0 THEN
      S_PARE:=S_PARE+C;
    ELSE
      S_IMPARE:=S_IMPARE+C;
    END IF;
  I:=I+1;
  END LOOP;
  DBMS_OUTPUT.PUT_LINE('SUMA PARE='||S_PARE);
  DBMS_OUTPUT.PUT_LINE('SUMA IMPARE='||S_IMPARE);

END;
/


DECLARE
   S VARCHAR2(50):='423908TY4902A3849#2798$6328';
   S_PARE PLS_INTEGER:=0;
   S_IMPARE PLS_INTEGER DEFAULT 0;
   C CHAR;
   I PLS_INTEGER:=1;
BEGIN
  LOOP
    C:=SUBSTR(S,I,1);

    CASE
    WHEN C NOT BETWEEN '1' AND '9' THEN
      NULL;
    WHEN C MOD 2 = 0 THEN
      S_PARE:=S_PARE+C;
    ELSE
      S_IMPARE:=S_IMPARE+C;
    END CASE;

  I:=I+1;
  EXIT WHEN I>LENGTH(S);
  END LOOP;
  DBMS_OUTPUT.PUT_LINE('SUMA PARE='||S_PARE);
  DBMS_OUTPUT.PUT_LINE('SUMA IMPARE='||S_IMPARE);

END;
```

/

SGBD Oracle

Cursorul

Managing cursors

- **SQL (Implicit) Cursor**
- **Explicit Cursor**
  - **OPEN-LOOP-FECTH-CLOSE / FOR LOOP**
  - **Cursors with parameters**
  - **The FOR UPDATE clause**
  - **Dynamic cursors**

Managing cursors

- **A cursor is a pointer to a pre allocated memory location in the System Global Area (SGA – a memory structure which is shared by all server and background processes).**
- **The attributes that provide information regarding the results of the cursors can be referenced using:**
  - **SQL%ATRIBUTE_NAME for implicit cursors. The values are retained until another SQL statement is run;**
  - **CURSOR_NAME%ATRIBUTE_NAME for explicit cursor.**

Cursor attributes

- **All the attributes of the implicit cursors are NULL if no SQL statement was previously run**
- **%ROWCOUNT**
  - **Has the PLS_INTEGER type**
  - **Is NULL if no DML statement was previously run**
  - **Returns the number of rows affected by the previous DML statement**
  - **Isn’t very useful, together with %FOUND and %NOTFOUND, for SELECT statements in the case of implicit cursors**
- **%FOUND**
  - **Has the BOOLEAN data type**
  - **Is NULL if no DML statement was previously run**
  - **For implicit cursors, is TRUE if the last statement affected/returned at least one row**
  - **For explicit cursors, is TRUE if the previous FETCH was successful**

Cursor attributes

- %NOTFOUND
  - Is NULL if no LMD statement was previously run
  - Has opposite meaning compared to %FOUND
- %BULK_ROWCOUNT
  - A composite attribute designed for use with the FORALL statement.
  - Its i*th* element stores the number of rows processed by the i*th* execution of an UPDATE or DELETE statement.
  - If the i*th* execution affects no rows, %BULK_ROWCOUNT(i) returns zero.
- %BULK_EXCEPTIONS
  - Stores information about any exceptions encountered by a FORALL statement that uses the SAVE EXCEPTIONS clause.
  - A loop is required to determine where the exception occurred. SQL%BULK_EXCEPTIONS(i).ERROR_INDEX specifies which iteration of the FORALL loop caused an exception.

Cursor attributes

- **%IS_OPEN**
  - **Has the BOOLEAN type**
  - **Indicates if a cursor is opened**
  - **In case of the implicit cursors, it is always FALSE after an SQL statement**

Example

- **What will the following block display?**

```
set serveroutput on
begin
if SQL%found then dbms_output.put_line(0) ;
 else
 dbms_output.put_line(1) ;
end if;
end;
/
```

a) 0
b) 1
c) Will return a compilation-time error
d) Will return an exception
e) NULL

- **What will the following block display?**

```
set serveroutput on
begin
If not SQL%found then
dbms_output.put_line(0) ;
 else
 dbms_output.put_line(1) ;
end if;
end;
/
```

a) 0
b) 1
c) Will return a compilation-time error
d) Will return an exception
e) NULL

Example

- **What will the following block display?**

```
set serveroutput on
begin
dbms_output.put_line(decode(SQL%ROWCOUNT,NULL,0,1)) ;
end;
/
```

a) 0
b) 1
c) Will return a compilation-time error
d) Will return an exception
e) NULL

- **What will the following block display?**

```
set serveroutput on
declare
n pls_integer;
begin
select decode(sql%rowcount,NULL,1,0) into n from dual;
dbms_output.put_line(n) ;
end;
/
```

a) 0
b) 1
c) Will return a compilation-time error
d) Will return an exception
e) NULL

Example

- **What will the following block display?**

```
set serveroutput on
begin
dbms_output.put_line(nvl(sql%rowcount,0)) ;
end;
/
```

a) 0
b) 1
c) Will return a compilation-time error
d) Will return an exception
e) NULL

- **What will the following block display?**

```
set serveroutput on
begin
 update angajati set salariul=salariul+10 where
ID_DEPARTAMENT=30;
 if SQL%ISOPEN then
  dbms_output.put_line('Is opened');
 else
 dbms_output.put_line('Is not opened');
 end if;
end;
/
```

a) Is opened
b) Is not opened
c) Will return a compilation-time error
d) Will return an exception
e) NULL

Implicit Cursors

- **Also known as SQL Cursors**
- **Oracle processes every SQL statement in a PL/SQL block as an implicit cursor, so implicit cursors store information regarding processing DML statements**
- **Is automatically closed after the statement gets executed**

Explicit cursors

- **Have names**
  - **Are unique identifiers in the block**
  - **Can’t appear in a expression**
  - **They cannot take values**
- **Are associated with SELECT statements that return more than one row. The INTO clause should not appear.**
- **The results returned by a cursor can be processed using**
  - **OPEN – LOOP – FETCH – CLOSE**
  - **Using FOR LOOP**

Explicit cursors

- **If the rows should be processed in an order, the ORDER BY clause should be used**
- **If an expression appears in the select statement, an alias should be used if the expression will be referenced later**
- **If we try to open a cursor that is already opened or to close an already closed one, exceptions will be raised**

Explicit cursors

- **When the cursor is opened, resources get allocated**
- **The query is processed.**
- **If FOR UPDATE is included than the rows get blocked**
- **The cursor is opened and the pointer is positioned before the first row**
- **At every FETCH the pointer advanced towards the last row (never in reverse)**
- **The information is read into PL/SQL variables**
- **When the pointer is positioned after the last row, the looping stops**
- **When the cursor is closed, resources get released**

Examples

```
declare
 cursor c is select id_angajat,prenume,nume,salariul from angajati order by salariul desc;
 r c%rowtype;
begin
 open c;
 if c%found then dbms_output.put_line('AAA');
  else dbms_output.put_line('BBB');
 end if;
  loop
 fetch c into r;
 exit when c%notfound;
 dbms_output.put_line(r.prenume||' '||r.nume||' '||r.salariul);
 end loop;
close c;
end;
/
```

Examples

```
declare
 cursor c is select id_angajat,prenume,nume, salariul from angajati order by salariul desc;
 r c%rowtype;
begin
 open c;
 fetch c into r;
  while c%found loop
 dbms_output.put_line(r.prenume||' '||r.nume||' '||r. salariul);
fetch c into r;
 end loop;
close c;
end;
/
```

Examples

```
set serveroutput on size 20000
declare
 cursor c is select id_angajat,prenume,nume, salariul from angajati order by salariul desc;
 r c%rowtype;
begin
 for r in c loop
 dbms_output.put_line(r.prenume||' '||r.nume||' '||r. salariul);
 end loop;
end;
/
```

Example using FOR

- **What will the following block display**

```
SET SERVEROUTPUT ON
declare
 cursor c is select id_angajat,prenume,nume,salariul from angajati order by salariul desc;
begin
 open c;
 for r in c loop
 dbms_output.put_line(r.prenume||' '||r.nume||' '||r. salariul);
 end loop;
 close c;
end;
```

a) **All the angajati**

b) **One employee**

c) **An exception**

d) **Two exception**

e) **Contains a syntax error**

Cursors with parameters

```
declare
  cursor c is select id_comanda from rand_comenzi group by id_comanda order by sum(pret*cantitate) desc;
  cursor d(p_id_comanda number) is select p.denumire_produs,o.pret,o.cantitate,o.cantitate*o.pret  rand_val
  from produse p join rand_comenzi o on p.id_produs =o.id_produs where o.id_comanda=p_id_comanda order by
rand_val desc;
  v_order_id number;
  v_val number:=0;
begin
  open c;
  fetch c into v_order_id;
  close c;
  for r in d(v_order_id) loop
     dbms_output.put_line(d%rowcount||' - '||r.denumire_produs ||': '||r.pret||'*'||r.cantitate||'='||r.rand_val);
     v_val := v_val + r.rand_val;
  end loop;
  dbms_output.put_line('Valoarea totala a comenzii '||v_order_id||' este '||v_val);
end;
/
```

Inline cursor

```
begin
 for r in (select id_angajat,prenume,nume,salariul from angajati order by salariul
  desc) loop
 dbms_output.put_line(r.prenume||' '||r.nume||' '||r. salariul);
 end loop;
end;
/
```

Cursors FOR UPDATE

```
declare
 cursor c(p_id number) is select id_angajat,id_departament ,prenume,nume,salariul from angajati where id_departament=p_id order by salariul desc for update;
 cursor d is select id_departament  from angajati group by id_departament  order by count(*) desc;
 n number(3);
begin
 open d;
 fetch d into n;
 close d;
 for r in c(n) loop
 update angajati set salariul=salariul+10 where current of c;
 dbms_output.put_line(r.id_departament||' '||r.prenume||' '||r.nume||' '||r.salariul);
 end loop;
end;
/
```

Dynamic SQL
REF CURSOR VARIABLES

- Is of REFERENCE type (similar to the POINTER type from C)
- Does not contain any data
- Can be declared as **REF CURSOR** that can be STRONG or WEAK whereas the **SYS_REFCURSOR** predefined type is always WEAK
  - **STRONG**: linked with a table:
    - *TYPE EMD_DATA_T IS REF CURSOR RETURN angajati%ROWTYPE;*
    - *X EMP_DATA_T;*
  - **WEAK**: not linked with a table:
    - X SYS_REFCURSOR;
- Can be opened for different queries
- Can be used as parameters is subprograms but cannot be declared in the SPECs of a Package
- Can be defined using BIND variables

Dynamic Cursor Example

```
declare
type t_c is ref cursor return angajati%rowtype;
c t_c;
 cursor d is select id_departament from angajati group by id_departament order by count(*) desc;
 n number(3);
 r angajati%rowtype;
begin
 open d;
 fetch d into n;
 close d;
 open c for select * from angajati where id_departament=n order by salariul desc;
 loop
 fetch c into r;
 exit when c%notfound;
 dbms_output.put_line(r.id_departament||' '||r.prenume||' '||r.nume||' '||r.salariul);
 end loop;
 close c;
end;
```

Dynamic Cursor Example

```
declare
type t_c is ref cursor return angajati%rowtype;
c t_c;
 cursor d is select id_departament from angajati group by id_departament order by count(*) desc;
 n number(3);
 r angajati%rowtype;
begin
 open d;
 fetch d into n;
 close d;
 open c for 'select * from angajati where id_departament=:1 order by salariul desc' using n;
 loop
 fetch c into r;
 exit when c%notfound;
 dbms_output.put_line(r.id_departament||' '||r.prenume||' '||r.nume||' '||r.salariul);
 end loop;
 close c;
end;
```

Dynamic Cursor Example

```
declare
type t_c is ref cursor;
c t_c;
 cursor d is select id_departament from angajati group by id_departament order by count(*) desc;
 n number(3);
 r angajati%rowtype;
begin
 open d;
 fetch d into n;
 close d;
 open c for 'select * from angajati where id_departament=:1 order by salariul desc' using n;
 loop
 fetch c into r;
 exit when c%notfound;
 dbms_output.put_line(r.id_departament||' '||r.prenume||' '||r.nume||' '||r.salariul);
 end loop;
 close c;
end;
```

Dynamic Cursor Example

```
declare
 c SYS_REFCURSOR;
 cursor d is select id_departament from angajati group by id_departament order by count(*) desc;
 n number(3);
 r angajati%rowtype;
begin
 open d;
 fetch d into n;
 close d;
 open c for 'select * from angajati where id_departament=:1 order by salariul desc' using n;
 loop
 fetch c into r;
 exit when c%notfound;
 dbms_output.put_line(r.id_departament||' '||r.prenume||' '||r.nume||' '||r.salariul);
 end loop;
 close c;
end;
```

Dynamic Cursor Example

```
VARIABLE dept_sel REFCURSOR /*might not work in PLSQL Dev*/
BEGIN
   OPEN :dept_sel FOR SELECT * FROM DEPARTMENTE;
 END;
/
PRINT dept_sel
```

With Functions

```
create or replace function
syscursor_dep return sys_refcursor is
 c sys_refcursor;
 cursor d is select id_departament
from angajati group by
id_departament order by count(*)
desc;
 n number(3);
 r angajati%rowtype;
begin
 open d;
 fetch d into n;
 close d;
 open c for 'select * from angajati
where id_departament=:1 order by
salariul desc' using n;
 return c;
end;
/

var rc refcursor;
exec :rc:=syscursor_dep;
print rc
```

Dynamic Blocks

```
declare
 p varchar2(128);
 x varchar2(5);
begin
 p:='begin dbms_output.put_line(''Message=''||:e); end;';
 x:='ABC';
 execute immediate p using x;
end;
```

Bibliography

- https://docs.oracle.com/cd/B14117_01/appdev.101/b10807/13_elems048.htm
- Gabriela Mihai, Suport de curs SGBD

```
/*Colectii
  1. index by table
  2. nested table
  3. varrray
  */
set SERVEROUTPUT on
DECLARE
  TYPE T_ANG IS TABLE OF VARCHAR2(100) INDEX BY PLS_INTEGER;
  V T_ANG;
BEGIN
  v(10001):='Alin';
  V(-10):='Ion';
  v(550):='Maria';
  v(7798997):='Mihaela';

  v(7798987):='Mihai';

  dbms_output.put_line('Numar elemente: '||v.count);
  dbms_output.put_line('Numar elemente: '||v.count);
  dbms_output.put_line('Indexul primului element: '||v.first);
  dbms_output.put_line('Indexul ultimului element: '||v.last);
  dbms_output.put_line('Primul element: '||v(v.first));
  --v.delete(550);

  FOR I IN v.first..v.last LOOP
   IF V.EXISTS(I) THEN
    dbms_output.put_line(V(I));
   END IF;
  END LOOP;

END;
/

DECLARE
  TYPE T_ANG IS TABLE OF VARCHAR2(100) INDEX BY PLS_INTEGER;
  V T_ANG;
  I PLS_INTEGER;
BEGIN
  v(10001):='Alin';
  V(-10):='Ion';
  v(550):='Maria';
  v(7798997):='Mihaela';

  v(7798987):='Mihai';

  dbms_output.put_line('Numar elemente: '||v.count);
  dbms_output.put_line('Numar elemente: '||v.count);
  dbms_output.put_line('Indexul primului element: '||v.first);
  dbms_output.put_line('Indexul ultimului element: '||v.last);
  dbms_output.put_line('Primul element: '||v(v.first));
  --v.delete(550);
  I:=V.FIRST;
  WHILE I IS NOT NULL LOOP
```

```
    Dbms_output.put_line(V(I));
    I:=V.NEXT(I);
  END LOOP;
END;
/

-- SA SE AFISEZE ANGAJATII (NUME SI PRENUME) CU SALARIILE INTRE 50000 SI 10000 SI CARE NU AU
COMISION

SELECT NUME, PRENUME, SALARIUL FROM ANGAJATI WHERE SALARIUL BETWEEN 5000 AND 10000
AND COMISION IS NOT NULL
ORDER BY SALARIUL DESC;

DECLARE
  TYPE T_ANG IS TABLE OF VARCHAR2(100) INDEX BY PLS_INTEGER;
  V T_ANG;
BEGIN
  SELECT NUME||' '||PRENUME||' ARE SAL '||SALARIUL BULK COLLECT INTO V FROM ANGAJATI
   WHERE SALARIUL BETWEEN 5000 AND 10000 AND COMISION IS NOT NULL
    ORDER BY SALARIUL DESC;
  DBMS_OUTPUT.PUT_LINE(V(V.FIRST));
  DBMS_OUTPUT.PUT_LINE(V(V.LAST));
  V.DELETE(20,25);
  FOR I IN V.FIRST..V.LAST LOOP
    IF V.EXISTS(I) THEN
    DBMS_OUTPUT.PUT_LINE(I||'->'||V(I));
    END IF;
  END LOOP;
END;
/
SELECT * FROM ANGAJATI;
DECLARE
  TYPE T_ANG IS TABLE OF ANGAJATI%ROWTYPE INDEX BY PLS_INTEGER;
  V T_ANG;
BEGIN
  SELECT * BULK COLLECT INTO V FROM ANGAJATI
   WHERE SALARIUL BETWEEN 5000 AND 10000 AND COMISION IS NOT NULL
    ORDER BY SALARIUL DESC;
  DBMS_OUTPUT.PUT_LINE(V(V.FIRST).NUME);
  DBMS_OUTPUT.PUT_LINE(V(V.LAST).NUME);
  V.DELETE(20,25);
  FOR I IN V.FIRST..V.LAST LOOP
    IF V.EXISTS(I) THEN
    DBMS_OUTPUT.PUT_LINE(I||'->'||V(I).NUME||' '||V(I).PRENUME||' ARE SALARIUL '||V(I).SALARIUL);
    END IF;
  END LOOP;
END;
/

DECLARE
  TYPE R_ANG IS RECORD(
    NUME VARCHAR2(20),
    PRENUME VARCHAR2(20),
    SALARIUL NUMBER);
```

```
  TYPE T_ANG IS TABLE OF R_ANG INDEX BY PLS_INTEGER;
  V T_ANG;
BEGIN
  SELECT NUME, PRENUME, SALARIUL BULK COLLECT INTO V FROM ANGAJATI
   WHERE SALARIUL BETWEEN 5000 AND 10000 AND COMISION IS NOT NULL
    ORDER BY SALARIUL DESC;
  DBMS_OUTPUT.PUT_LINE(V(V.FIRST).NUME);
  DBMS_OUTPUT.PUT_LINE(V(V.LAST).NUME);
  V.DELETE(20,25);
  FOR I IN V.FIRST..V.LAST LOOP
    IF V.EXISTS(I) THEN
    DBMS_OUTPUT.PUT_LINE(I||'->'||V(I).NUME||' '||V(I).PRENUME||' ARE SALARIUL '||V(I).SALARIUL);
    END IF;
  END LOOP;
END;
/

-- FOLOSIND O COLECTIE SA SE AFISEZE VALOAREA FIECAREI COMENZI


DECLARE
  TYPE R_ANG IS RECORD(
    ID_COMANDA NUMBER,
    VALOARE NUMBER);
  TYPE T_ANG IS TABLE OF R_ANG INDEX BY PLS_INTEGER;
  V T_ANG;
BEGIN
  SELECT ID_COMANDA,SUM(PRET*CANTITATE) BULK COLLECT INTO V FROM RAND_COMENZI
GROUP BY ID_COMANDA;
  FOR I IN V.FIRST..V.LAST LOOP
    IF V.EXISTS(I) THEN
    DBMS_OUTPUT.PUT_LINE(I||'->'||V(I).ID_COMANDA||' '||V(I).VALOARE);
    END IF;
  END LOOP;
  V.DELETE;
END;
/

SET SERVEROUTPUT ON
DECLARE
TYPE tab_ind IS TABLE OF NUMBER INDEX BY VARCHAR2(1);
t tab_ind;
i varchar2(1);
BEGIN
t('a') := ASCII('a'); t('A') := ASCII('A');
t('b') := ASCII('b'); t('B') := ASCII('B');
t('x') := ASCII('x'); t('X') := ASCII('X');
i := t.FIRST;
WHILE i IS NOT NULL LOOP
  DBMS_OUTPUT.PUT_LINE('t('||i ||')='||t(i));
  i := t.NEXT(i);
END LOOP;
END;
```

```
/


set serveroutput on
DECLARE
TYPE tab_ind IS TABLE OF angajati%ROWTYPE INDEX BY PLS_INTEGER;
t tab_ind;
BEGIN
SELECT * BULK COLLECT INTO t
FROM angajati minus
SELECT *
FROM angajati
WHERE ROWNUM<=50;
DBMS_OUTPUT.PUT_LINE(SQL%ROWCOUNT);
DBMS_OUTPUT.PUT_LINE('Numar de elemente '||t.COUNT);
FOR i IN t.FIRST..t.LAST LOOP
   DBMS_OUTPUT.PUT_LINE(t(i).prenume||' '|| t(i).nume);
END LOOP;
delete from angajati where 1=2;
DBMS_OUTPUT.PUT_LINE(SQL%ROWCOUNT);
FOR i IN t.FIRST..t.LAST loop
   update angajati set salariul=salariul+100 where id_angajat=t(i).id_angajat;
   DBMS_OUTPUT.PUT_LINE(SQL%ROWCOUNT);
end loop;
END;
/


DECLARE
TYPE tab_ind IS TABLE OF angajati%ROWTYPE; -- NESTED TABLE
t tab_ind;
BEGIN
SELECT * BULK COLLECT INTO t
FROM angajati minus
SELECT *
FROM angajati
WHERE ROWNUM<=50;
DBMS_OUTPUT.PUT_LINE(SQL%ROWCOUNT);
DBMS_OUTPUT.PUT_LINE('Numar de elemente '||t.COUNT);
FOR i IN t.FIRST..t.LAST LOOP
   DBMS_OUTPUT.PUT_LINE(t(i).prenume||' '|| t(i).nume);
END LOOP;
delete from angajati where 1=2;
DBMS_OUTPUT.PUT_LINE(SQL%ROWCOUNT);
FORALL i IN t.FIRST..t.LAST  update angajati set salariul=salariul+100 where id_angajat=t(i).id_angajat;
   DBMS_OUTPUT.PUT_LINE(SQL%ROWCOUNT);
END;
/

set serveroutput on
DECLARE
TYPE tab_imb IS TABLE OF NUMBER;
t tab_imb := tab_imb(1,20,3,40,5);
t_null tab_imb;
```

```
BEGIN
t.EXTEND(5);
FOR i IN 6..10 LOOP
t(i):=t(i-1)+5;
END LOOP;
DBMS_OUTPUT.PUT('Colectia are ' || t.COUNT ||' elemente: ');
FOR i IN t.FIRST..t.LAST LOOP
DBMS_OUTPUT.PUT(t(i) || ' ');
END LOOP;
DBMS_OUTPUT.NEW_LINE;
t.delete;
IF t IS NULL THEN
   DBMS_OUTPUT.PUT_LINE('Colectia este nula');
  else
   DBMS_OUTPUT.PUT_line('Colectia are ' || t.COUNT ||' elemente');
END IF;
t.extend(2);
t(1):=500;t(2):=700;
DBMS_OUTPUT.PUT_line('Colectia are ' || t.COUNT ||' elemente');
t := t_null;
IF t IS NULL THEN
   DBMS_OUTPUT.PUT_LINE('Colectia este nula');
  else
   DBMS_OUTPUT.PUT_line('Colectia are ' || t.COUNT ||' elemente');
END IF;
END;
/
```

```
set serveroutput on
DECLARE
TYPE tab_imb IS TABLE OF NUMBER;
t tab_imb := tab_imb(1,20,3,40,5);
t_null tab_imb;
BEGIN
t.EXTEND(5);
FOR i IN 6..10 LOOP
t(i):=t(i-1)+5;
END LOOP;
DBMS_OUTPUT.PUT('Colectia are ' || t.COUNT ||' elemente: ');
FOR i IN t.FIRST..t.LAST LOOP
DBMS_OUTPUT.PUT(t(i) || ' ');
END LOOP;
DBMS_OUTPUT.NEW_LINE;
t.delete;
IF t IS NULL THEN
   DBMS_OUTPUT.PUT_LINE('Colectia este nula');
  else
   DBMS_OUTPUT.PUT_line('Colectia are ' || t.COUNT ||' elemente');
END IF;
t.extend(2);
t(1):=500;t(2):=700;
DBMS_OUTPUT.PUT_line('Colectia are ' || t.COUNT ||' elemente');
t := t_null;
IF t IS NULL THEN
   DBMS_OUTPUT.PUT_LINE('Colectia este nula');
  else
   DBMS_OUTPUT.PUT_line('Colectia are ' || t.COUNT ||' elemente');
END IF;
END;
/

create type t_grade_celsius is table of number(5,2);
/
create table localitati(
id_loc number primary key,
denumire varchar2(100),
grade t_grade_celsius)
  nested table grade store as t_grade ;
insert into localitati values(1,'Constanta', t_grade_celsius(10,15,17,20));
insert into localitati values(2,'Bucuresti', t_grade_celsius(7,11,15,17,21));
commit;

declare
  v t_grade_celsius;
begin
null;
end;
/
drop table t_grade;
select * from localitati;
select column_value grade from table(select grade from localitati where id_loc=1);
```

```
select id_loc,denumire,b.* from localitati a,table(a.grade) b;
update localitati set grade=t_grade_celsius(10,15,17,20,27,21,21) where id_loc=1;

DECLARE
TYPE tab_vec IS VARRAY(50) OF NUMBER;
t tab_vec := tab_vec();
BEGIN
t.extend(10);
FOR i IN 1..10 LOOP
  t(i):=i;
END LOOP;
DBMS_OUTPUT.PUT('Colectia are ' || t.COUNT ||' elemente: ');
FOR i IN t.FIRST..t.LAST LOOP
    DBMS_OUTPUT.PUT(t(i) || ' ');
END LOOP;
DBMS_OUTPUT.NEW_LINE;
FOR i IN 1..10 LOOP
  IF i mod 2 = 1 THEN t(i):=null;
  --t.delete(i); -- in varray nu se poate sterge un anume element
  END IF;
END LOOP;
DBMS_OUTPUT.PUT('Colectia are ' || t.COUNT ||' elemente: ');
FOR i IN t.FIRST..t.LAST LOOP
    DBMS_OUTPUT.PUT(nvl(t(i), 0) || ' ');
END LOOP;
DBMS_OUTPUT.NEW_LINE;
--t.delete(1); -- wrong number or types of arguments in call to 'DELETE'
t.trim(3); -- sterge elemente de la finalul vectorului
DBMS_OUTPUT.PUT_LINE('Colectia are '||t.COUNT||' elemente');
t.DELETE;
DBMS_OUTPUT.PUT_LINE('Colectia are '||t.COUNT||' elemente');
END;
/


--- index by table ---
set serveroutput on
DECLARE
TYPE tab_ind IS TABLE OF angajati%ROWTYPE INDEX BY PLS_INTEGER;
t tab_ind;
BEGIN
SELECT * BULK COLLECT INTO t FROM angajati WHERE salariul between 5000 and 10000 order by salariul desc;
DBMS_OUTPUT.PUT_LINE('Numar de elemente '||t.COUNT);
t.delete(t.first);
DBMS_OUTPUT.PUT_LINE('Numar de elemente '||t.COUNT);
FOR i IN t.FIRST.. t.LAST LOOP
    DBMS_OUTPUT.PUT_LINE(t(i).prenume ||' '||t(i).nume ||'->'||i);
END LOOP;
END;
/


--- nested table ---
set serveroutput on
```

```
DECLARE
TYPE tab_ind IS TABLE OF angajati%ROWTYPE;
t tab_ind;
BEGIN
SELECT * BULK COLLECT INTO t FROM angajati WHERE salariul between 5000 and 10000 order by salariul desc;
DBMS_OUTPUT.PUT_LINE('Numar de elemente '||t.COUNT);
t.delete(t.first);
DBMS_OUTPUT.PUT_LINE('Numar de elemente '||t.COUNT);
FOR i IN t.FIRST.. t.LAST LOOP
   DBMS_OUTPUT.PUT_LINE(t(i).prenume ||' '||t(i).nume ||'->'||i);
END LOOP;
END;
/
```

```
--- varray ---
set serveroutput on
DECLARE
TYPE tab_ind IS varray(10) OF angajati%ROWTYPE;
t tab_ind;
BEGIN
SELECT * BULK COLLECT INTO t FROM angajati WHERE salariul between 5000 and 10000 order by salariul desc;
DBMS_OUTPUT.PUT_LINE('Numar de elemente '||t.COUNT);
--t.delete(t.first);
DBMS_OUTPUT.PUT_LINE('Numar de elemente '||t.COUNT);
FOR i IN t.FIRST.. t.LAST LOOP
   DBMS_OUTPUT.PUT_LINE(t(i).prenume ||' '||t(i).nume ||'->'||i);
END LOOP;
END;
/

CREATE OR REPLACE TYPE t_vect_proiecte AS VARRAY(10) OF varchar2(32);
/
CREATE TABLE echipe
( id_echipa NUMBER(4) PRIMARY KEY,
denumire VARCHAR2(40),
proiecte t_vect_proiecte);

insert into echipe values (1,'Inspire',t_vect_proiecte('HR Application','Trading Application'));
insert into echipe values (2,'Excel',t_vect_proiecte('Client Support','HR Application','Trading Application'));
insert into echipe values (3,'Global',t_vect_proiecte('Advanced Support','Big Data Analitics','Cloud Migration'));

select * from echipe;

update echipe set proiecte=t_vect_proiecte('Big Data Analitics','Cloud Migration')  where id_echipa=3;
select id_echipa,denumire,b.*  from echipe a, table(a.proiecte) b;
```

Cursorul

1. Implicit (cursorul SQL) - insert, update, delete
   SQL%FOUND
   SQL%NOTFOUND
   SQL%ROWCOUNT
2. Explicit

```
declare
  n number;
begin
  n:='46327'; -- conversie implicita
  n:=to_number('46327'); -- conversie explicita
end;
/
```

-- Sa se mareasca cu 5% salariul ang care au intermediat cel putin 3 comenzi. Sa se afiseze numarul salariilor modificate.

```
BEGIN
UPDATE ANGAJATI SET SALARIUL=SALARIUL*1.05 WHERE ID_ANGAJAT IN (SELECT ID_ANGAJAT
FROM COMENZI GROUP BY ID_ANGAJAT
  HAVING COUNT(*)>=3);
DBMS_OUTPUT.PUT_LINE('SALARII MARITE: '||SQL%ROWCOUNT);
ROLLBACK;
END;
/

SELECT * FROM ANGAJATI WHERE ID_ANGAJAT IN (SELECT ID_ANGAJAT FROM COMENZI GROUP BY
ID_ANGAJAT
  HAVING COUNT(*)>=3);
ROLLBACK;


BEGIN
UPDATE ANGAJATI SET SALARIUL=SALARIUL*1.05 WHERE ID_ANGAJAT IN (SELECT ID_ANGAJAT
FROM COMENZI GROUP BY ID_ANGAJAT
  HAVING COUNT(*)>=3);
IF SQL%FOUND THEN
    DBMS_OUTPUT.PUT_LINE('S-A MARIT CEL PUTIN UN SALARIU');
ELSE
    DBMS_OUTPUT.PUT_LINE('NU S-A MARIT NICIUN SALARIU');
END IF;
ROLLBACK;
END;
/

set serveroutput on
begin
if SQL%found then dbms_output.put_line(0) ;
 else
 dbms_output.put_line(1) ;
end if;
end;
/

begin
If not SQL%found  then dbms_output.put_line(0) ;
 else
 dbms_output.put_line(1) ;
end if;
```

```
end;
/

begin
dbms_output.put_line(decode(SQL%ROWCOUNT,NULL,0,1)) ;
  --EXCEPTION WHEN OTHERS THEN NULL;
end;
/

declare
n pls_integer;
begin
select decode(SQL%ROWCOUNT,NULL,1,0) into n from dual;

dbms_output.put_line(n) ;
end;
/

set serveroutput on
begin
--UPDATE ANGAJATI SET SALARIUL=5000 WHERE 1=2;
dbms_output.put_line(sql%rowcount);
dbms_output.put_line(nvl('','NU EXISTA')) ;
end;
/

set serveroutput on
begin
 update angajati set salariul=salariul+10 where ID_DEPARTAMENT=30;
 if SQL%ISOPEN then
  dbms_output.put_line('Is opened');
 else
 dbms_output.put_line('Is not opened');
 end if;
end;
/

-- CURSORUL EXPLICIT - SELECT

SELECT NUME,PRENUME,SALARIUL FROM ANGAJATI WHERE ID_ANGAJAT IN (SELECT ID_ANGAJAT
FROM COMENZI GROUP BY ID_ANGAJAT
  HAVING COUNT(*)>=3);

DECLARE
  CURSOR C IS SELECT NUME,PRENUME,SALARIUL FROM ANGAJATI WHERE ID_ANGAJAT IN (SELECT
ID_ANGAJAT FROM COMENZI GROUP BY ID_ANGAJAT
  HAVING COUNT(*)>=3);
  R C%ROWTYPE;
BEGIN

   IF NOT C%ISOPEN THEN
    OPEN C;
    dbms_output.put_line('S-a deschis');
   END IF;
```

```
  LOOP
  FETCH C INTO R;
  EXIT WHEN C%NOTFOUND;
  dbms_output.put_line('ANGAJATUL '||R.NUME||' '||R.PRENUME||' '||R.SALARIUL);
  END LOOP;
  CLOSE C;
END;
/
```

SGBD Oracle

Exceptii

Errors

- Type of errors:
  - Compilation-time
    - Are detected by the PL/SQL engine
    - Can’t be handled in the block because it isn’t yet executed
    - Must be corrected by the programmer
  - Run-time
    - In PL/SQL are called exceptions
    - Sometimes, they can occur from design faults, coding mistakes or even hardware failures but in other situations they are part of the design
    - When an exception occurs the block is terminated. The exception has to be handled in the current block (or in an outer block) so the program won’t terminate with an error
    - Using exception handlers for error-handling makes programs easier to write and understand, and reduces the likelihood of unhandled exceptions

Exception categories

- Predefined
  - A predefined exception is an internally defined exception that PL/SQL has given a name.
- Non-predefined
  - The runtime system raises internally defined exceptions implicitly (automatically).
  - An internally defined exception always has an error code, but does not have a name unless PL/SQL gives it one or you give it one.
- User-defined
  - Users can define exceptions in the declarative part of any PL/SQL anonymous block, subprogram, or package.
  - These exceptions must be raised explicitly

See: https://docs.oracle.com/cd/B19306_01/appdev.102/b14261.pdf (chapter 10)

Exception handling

```
BEGIN
 statements
EXCEPTION
  WHEN ex_name_1 THEN statements_1                    -- Exception handler
  WHEN ex_name_2 OR ex_name_3 THEN statements_2  -- Exception
handler
  WHEN OTHERS THEN statements_3                       -- Exception handler
END;
```

Error handling

```
set serveroutput on
declare
 fn angajati.prenume%type;
begin
 select prenume into fn from angajati where id_angajat=200;
 select prenume into fn from angajati where id_angajat=300;
exception
 when no_data_found then
   dbms_output.put_line('The employee wasn''t found!');
end;
```

Error handling

```
declare
 fn angajati.prenume%type;
 q number(2);
begin
 q:=1;
 select prenume into fn from angajati where id_angajat=200;
 q:=2;
 select prenume into fn from angajati where id_angajat=300;
exception
 when no_data_found then
   dbms_output.put_line('The employee wasn''t found at query '||q);
end;
```

Example

- **What will the following block return**

```
1)  declare
2)   fn varchar2(128);
3)   q number(2);
4)  begin
5)   for i in 1..500 loop
6)     if (i mod 2=1) then
7)         i:=i+1;
8)     end if;
9)     select prenume||'  '||nume into fn from angajati where id_angajat=i;
10)    dbms_output.put_line(fn);
11)  end loop;
12) end;
```

a) **An exception at the 5th line**

b) **An exception at the 6th line**

c) **An exception at the 7th line**

d) **A compilation-time error**

e) **The block will run successful**

Example

```
ORA-06550: line 7, column 8:
PLS-00363: expression 'I' cannot be used as an assignment target
ORA-06550: line 7, column 8:
PL/SQL: Statement ignored
06550. 00000 -  "line %s, column %s:\n%s"
*Cause:    Usually a PL/SQL compilation error.
*Action:
```

- What will the following block return

```
1)  declare
2)   fn varchar2(128);
3)   q number(2);
4)  begin
5)   for i in 1..500 loop
6)     if (i mod 2=1) then
7)  i:=i+1;
8)     end if;
9)     select prenume||' '||nume into fn from angajati where id_angajat=i;
10)    dbms_output.put_line(fn);
11)  end loop;
12) end;
```

a) An exception at the 5th line

b) An exception at the 6th line

c) An exception at the 7th line

d) A compilation-time error

e) The block will run successful

Example

- **What will the following block return**

```
1)  Set serveroutput on
2)  declare
3)   fn varchar2(128);
4)   q number(2);
5)  begin
6)   for i in 1..500 loop
7)    begin
8)    select prenume||' '||nume||' '||i into fn from angajati where id_angajat=i;
9)     exception when
10)     no_data_found then null;
11)   end;
12)   dbms_output.put_line(fn);
13)  end loop;
14) end;
```

a) **An exception at the 6th line**

b) **An exception at the 7th line**

c) **An error**

d) **Will display once, all the employees that exist with ID's between 1 and 500**

e) **Can display one employee more than once**

Example

- **What will the following block return**

```
1)  Set serveroutput on
2)  declare
3)   fn varchar2(128);
4)   q number(2);
5)  begin
6)   for i in 1..500 loop
7)    begin
8)    select prenume||' '||nume||' '||i into fn from angajati where id_angajat=i;
9)   dbms_output.put_line(fn);
10)    exception when no_data_found then null;
11)   end;
12) end loop;
13) end;
```

a) **An exception at the 6th line**

b) **An exception at the 7th line**

c) **A compilation-time error**

d) **Will display once, all the employees that exist with ID's between 1 and 500**

e) **Can display one employee more than once**

Error Code and Error Message Retrieval

- The error code can be retrieved using the SQLCODE function
  - For an internally defined exception, the numeric code is the number of the associated Oracle Database error (negative, except for NO_DATA_FOUND : +100).
  - For a user-defined exception, the numeric code is either +1 (default) or the error code associated with the exception by the EXCEPTION_INIT pragma.
- The error message can be retrieved using the SQLERRM function
- Outside an exception handler, or if the value of error_code is zero, SQLERRM returns ORA-0000.
- A SQL statement cannot invoke SQLCODE or SQLERRM.

Example

```
drop table occurred_exc;
create table occurred_exc (exc_user varchar2(32), exc_date date, exc_code number(32), exc_message varchar2(256));
begin
 raise no_data_found;
exception
  when others then
    insert into occurred_exc values(user,sysdate,SQLCODE,SQLERRM);
end;
```

- What will the block return?
  a) An exception
  b) A compilation-time error
  c) Will execute successfully

Example

```
declare
 c number(10);
 s varchar2(5);
begin
 raise no_data_found;
exception
  when others then
    c:=SQLCODE;
    s:=SQLERRM;
    insert into occurred_exc values(user,sysdate,c,s);
    commit;
end;
```

- **What will be the result?**
  a) **Will raise an exception that is handled**
  b) **Will return an exception that is not handled**
  c) **Will return an error**
  d) **Will execute successfully**

Example

```
set serveroutput on
declare
 c number(10);
 s varchar2(5);
begin
 raise no_data_found;
exception
  when others then
    c:=SQLCODE;
    begin
    dbms_output.put_line('Er1='||SQLERRM);
    s:=SQLERRM;
    dbms_output.put_line('Er2='||s);
    exception
      when others then
        dbms_output.put_line('Er3='||SQLERRM);
        insert into occurred_exc values(user,sysdate,c+1,s);
    end;
end;
```

Example

```
set serveroutput on
declare
 c number(10);
 s varchar2(5);
begin
 raise no_data_found;
exception
  when others then
    c:=SQLCODE;
    begin
    dbms_output.put_line('Er1='||SQLERRM);
    s:=SQLERRM;
    dbms_output.put_line('Er2='||s);
    exception
      when others then
        dbms_output.put_line('Er3='||SQLERRM);
        insert into occurred_exc values(user,sysdate,c+1,s);
    end;
end;
```

```
SQL>
Er1=ORA-01403: no data found
Er3=ORA-06502: PL/SQL: numeric or value error: character string buffer too small
ORA-01403: no data found
PL/SQL procedure successfully completed

SQL> select * from occurred_exc;
EXC_USER                         EXC_DATE                            EXC_CODE EXC_MESSAGE
-------------------------------- ----------- ---------------------------------- --------------------------
VLAD_ENG                         7/4/2018 8:                               101

SQL>
```

Example

```
Set serveroutput on
declare
cursor c is select prenume from angajati;
r c%rowtype;
begin
 begin
  fetch c into r;
dbms_output.put_line('Name='||r.prenume );
  exception when no_data_found then
    dbms_output.put_line('A');
  end;
 dbms_output.put_line('B');
 exception
 when others then
   dbms_output.put_line('C');
end;
```

What will the block display?

a) Name of a client
b) A B
c) B C
d) An exception because the cursor is not opened
e) C

Non predefined exception

- To handle error conditions (typically ORA- messages) that have no predefined name, you must use the OTHERS handler or the pragma EXCEPTION_INIT. A pragma is a compiler directive that is processed at compile time, not at run time.
- In PL/SQL, the pragma EXCEPTION_INIT tells the compiler to associate an exception name with an Oracle error number. That lets you refer to any internal exception by name and to write a specific handler for it. When you see an error stack, or sequence of error messages, the one on top is the one that you can trap and handle.
- You code the pragma EXCEPTION_INIT in the declarative part of a PL/SQL block, subprogram, or package using the syntax
  - *PRAGMA EXCEPTION_INIT(exception_name, -Oracle_error_number);*
- where *exception_name* is the name of a previously declared exception and the number is a negative value corresponding to an ORA- error number. The pragma must appear somewhere after the exception declaration in the same declarative section, as shown in the following example:

```
set SERVEROUTPUT on
declare
  is_not_null exception;
  pragma exception_init(is_not_null,-01400);
begin
 insert into angajati(id_angajat,prenume) values (2002, 'John');
   exception
    when is_not_null then
      dbms_output.put_line('Exceptie');
end;
/
```

User defined exceptions

- The procedure RAISE_APPLICATION_ERROR lets you issue user-defined ORA- error messages from stored subprograms. That way, you can report errors to your application and avoid returning unhandled exceptions.
- raise_application_error(error_number, message[, {TRUE | FALSE}]);
  - where error_number is a negative integer in the range -20000 .. -20999 and message is a character string up to 2048 bytes long. If the optional third parameter is TRUE, the error is placed on the stack of previous errors. If the parameter is FALSE (the default), the error replaces all previous errors.
- An application can call raise_application_error only from an executing stored subprogram (or method). When called, raise_application_error ends the subprogram and returns a user-defined error number and message to the application. The error number and message can be trapped like any Oracle error.
- In the following example, we call raise_application_error if no salary was increased:

```
BEGIN
  UPDATE angajati SET salariul = salariul + 100 where id_angajat = 1000;
  if SQL%NOTFOUND then
  raise_application_error(-20101, 'Angajatul nu exista');
  end if;
END;
/
```

Bibliography

- https://docs.oracle.com/cd/B14117_01/appdev.101/b10807/07_errs.htm#i7014

SGBD Oracle

Subprograme

Subprograms

- Are units that perform tasks
- Can be combined to form more complex programs
- Can be created:
  - At schema level – standalone subprograms
  - Inside a package – packed subprograms
  - Inside a PL/SQL block

Subprograms

- Parts:
  - Declarative part
  - Executable part
  - Exception handling
- Types
  - Procedures – perform actions, can return values through OUT or IN OUT parameters
  - Functions – usually return a single value. If the requirements are met, can be used in SQL statements like single-row functions
- Parameters
  - IN
  - OUT
  - IN OUT

Advantages

- SGA:
  - Shared pool
  - DB buffer cache
  - Redo log buffer
  - Large pool
  - Java pool

- The code is easier to maintain.
- Updates required by multiple applications can be done once
- The code is re-usable after the procedure is compiled successfully
- Contributed to applications' security
- Improves performance
  - In the data dictionary, exists both **source code** and **p-code (bytecode)**.
  - PL/SQL reads the compiled version of a procedure or package into the shared pool buffer of the SGA when it is invoked and not currently in the SGA.
  - Can be used by multiple users at once
  - *Least recently used* algorithm is used to manage the shared pool

Procedure

```
CREATE [OR REPLACE] PROCEDURE procedure_name
[(parameter_name [IN | OUT | IN OUT] type [, ...])]
{IS | AS}
BEGIN
  < procedure_body >
END [procedure_name];
```

Passing parameters

- ***get_orders(p_client number, p_category varchar2 default null, p_year number default null)***
  - Positional notation – get_orders(a,b,c), get_orders(101,’hardware1’,2015);
  - Named notation – get_orders(p_year=>2015, p_client=>101, p_category=>’hardware1’)
  - Exec get_orders(p_client=>101);
  - Mixed notation
    - get_orders(101,p_year=>2015) – OK,
    - get_orders(p_client=>a,b,c) – NOT OK

Executing a procedure

- Using EXECUTE/EXEC
  - It’s an SQL Plus statement that encloses the block in an anonymous block (BEGIN…END). Usually requires less memory compared to CALL
- Using CALL
  - It’s a SQL statement and it can only be used with SQL types. Can inhibit the propagation of exceptions.
- From another PL/SQL Block

Executing a procedure

```
CREATE OR REPLACE PROCEDURE get_total_orders
(p_id_cl IN comenzi. ID_CLIENT %type)
IS
v_val number;
BEGIN
Select sum(pret*cantitate) into v_val from rand_comenzi i,comenzi o where i. id_comanda=o. id_comanda and o. ID_CLIENT=p_id_cl;
Dbms_output.put_line('Total value= '||v_val);
END;
/

SQL> set serveroutput on
SQL> CALL get_total_orders(109);
SQL> EXECUTE get_total_orders(109);
begin
get_total_orders(109);
end;
/
```

Executing a procedure

```
CREATE OR REPLACE PROCEDURE get_total_orders_o
(p_id_cl IN comenzi. ID_CLIENT %type, p_val out number) is
BEGIN
Select nvl(sum(pret*cantitate),0) into p_val from rand_comenzi i,comenzi o where
i. id_comanda=o. id_comanda and o. ID_CLIENT=p_id_cl;
Dbms_output.put_line('Total value= '||p_val);
END;
/
```

```
declare  n number;
begin
get_total_orders_o(109,n);
dbms_output.put_line('Value='||n);
end;
```

```
VARIABLE n NUMBER
EXECUTE
get_total_orders_o(109,:n)
print n
```

```
VARIABLE n NUMBER
call
get_total_orders_o(109,:n);
print n
```

Executing a procedure

```
CREATE OR REPLACE PROCEDURE test_exc
IS
BEGIN
raise too_many_rows;
END;
/
```

Which displays the exception’s message?

```
call test_exc()
execute test_exc
```

Other clauses - AUTHID

- AUTHID DEFINER – execute with the owner’s permissions (default)
- AUTHID CURRENT_USER – executes with the current user’s permission

```
CREATE OR REPLACE PROCEDURE test_authid(n out number) authid current_user
IS
BEGIN
select count(*) into n from angajati;
END;
```

Other clauses - AUTONOMOUS_TRANSACTION

- The procedure is autonomous
- The main transaction gets suspended
- COMMIT or ROLLBACK must be specified in the procedure

What will return?

```
CREATE OR REPLACE PROCEDURE test_at(p_id number)
IS
PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
update angajati e set e.salariul=e.salariul+10 where e.id_angajat=p_id;
END;
/

declare
 s number;
begin
 select salariul into s from angajati where id_angajat=120;
 dbms_output.put_line('Before='||s);
 test_at(120);
 select salariul into s from angajati where id_angajat=120;
 dbms_output.put_line('After='||s);
end;
/
```

Functions

- Are named PL/SQL blocks that must return a value
- Are stored as schema objects
- Can be used in SQL statements if:
  - has only IN parameters
  - returns valid SQL data types
  - doesn’t contain COMMIT or ROLLBACK
  - used in SELECT statements, they can’t contain INSERT, UPDATE or DELETE
  - used in UPDATE or DELETE on a table, they can’t contain DML statements on the same table
  - they don’t contain DDL or DCL statements (using execute immediate)

Functions

```
CREATE [OR REPLACE] FUNCTION function_name
[(parameter_name [IN | OUT | IN OUT] type [, ...])]
RETURN return_datatype
{IS | AS}
BEGIN
  < function_body >
END [function_name];
```

Example

- Given the function:

```
CREATE OR REPLACE FUNCTION check_salary
(p_id angajati.id_angajat%type, p_sal number)
RETURN Boolean
IS
v_salariul angajati.salariul%type;
BEGIN
SELECT salariul into v_salariul from angajati
where id_angajat=p_id;
IF p_sal > v_salariul then
return true;
ELSE
return false;
end if;
EXCEPTION
WHEN no_data_found THEN
return NULL;
end;
/
```

What will ***select check_salary(109,5000) from dual*** return?

a) An exception
b) A compilation-time error
c) Always NULL
d) Just TRUE or FALSE

Example

- Given the function:

```
CREATE OR REPLACE FUNCTION check_salary_n
(p_id angajati.id_angajat%type, p_sal number)
RETURN number
IS
v_salariul angajati.salariul%type;
BEGIN
SELECT salariul into v_salariul from angajati
where id_angajat=p_id;
IF p_sal > v_salariul then
return 1;
ELSE
return -1;
end if;
EXCEPTION
WHEN no_data_found THEN
return NULL;
end;
/
```

What will ***select check_salary_n(109,5000) from dual*** return?

a) An exception
b) A compilation-time error
c) Always NULL
d) -1, 1 or NULL

Example

- Given the function:

```
CREATE OR REPLACE FUNCTION check_salary_n
(p_id angajati.id_angajat%type, p_sal number)
RETURN number
IS
v_salariul angajati.salariul%type;
BEGIN
SELECT salariul into v_salariul from angajati
where id_angajat=p_id;
IF p_sal > v_salariul then
return 1;
ELSE
return -1;
end if;
EXCEPTION
WHEN no_data_found THEN
return NULL;
end;
/
```

What will ***select nume,check_salary_n(id_angajat,5000) from angajati*** return?

a) An exception
b) A compilation-time error
c) Always NULL
d) -1, 1 or NULL for each employee

Example

- Given the same function, which of the following will work:
  - delete from angajati where check_salary_n(id_angajat,5000)=-1;
  - select * from comenzi where check_salary_n(id_angajat,5000)=-1;
  - delete from comenzi where check_salary_n(id_angajat,5000)=-1;

Will it work?

```
create or replace function get_dept_emps(p_dep in number) return sys_refcursor is
  dep sys_refcursor;
begin
     open dep for 'select nume,prenume from angajati where id_departament = :1' using
p_dep;
     return dep;
end;
/

variable x refcursor
exec :x:=get_dept_emps(80)
print x
```

Will it work?

```
declare
 x sys_refcursor;
begin
 x:=get_dept_emps(80);
 for r in x loop
   dbms_output.put_line(r.nume||' '||r.prenume);
end loop;
end;
/
```

Will it work?

```
set serveroutput on
declare
 x sys_refcursor;
 f_n varchar2(25);
 f_l varchar2(25);
begin
 x:=get_dept_emps(80);
 loop
 fetch x into f_n,f_l;
 exit when x%notfound;
   dbms_output.put_line(f_n||' '||f_l);
 end loop;
end;
/
```

Recursion in PL/SQL

```
CREATE OR REPLACE FUNCTION fibonacci(n NUMBER)
RETURN NUMBER
```

RESULT_CACHE -- If the cache contains the result from a previous call to the function with the same parameter values, the system returns the cached result to the invoker and does not reexecute the function body.

```
IS
BEGIN
IF n = 0 THEN
RETURN 0;
elsif n = 1 then return 1;
ELSE
RETURN fibonacci(n - 1) + fibonacci(n - 2);
END IF;
END;
/

variable x number
exec :x:=fibonacci(10)
```

F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}	F_{16}	F_{17}	F_{18}	F_{19}	F_{20}
0	1	1	2	3	5	8	13	21	34	55	89	144	233	377	610	987	1597	2584	4181	6765

Bibliography

- https://docs.oracle.com/cd/B14117_01/appdev.101/b10807/toc.htm
- Gabriela Mihai, Suport de curs SGBD

```
-- CURSORUL EXPLICIT - SELECT

SELECT NUME,PRENUME,SALARIUL FROM ANGAJATI WHERE ID_ANGAJAT IN (SELECT ID_ANGAJAT
FROM COMENZI GROUP BY ID_ANGAJAT
  HAVING COUNT(*)>=3);

set serveroutput on
DECLARE
  CURSOR C IS SELECT NUME,PRENUME,SALARIUL FROM ANGAJATI WHERE ID_ANGAJAT IN (SELECT
ID_ANGAJAT FROM COMENZI GROUP BY ID_ANGAJAT
  HAVING COUNT(*)>=3) order by SALARIUL desc;
  R C%ROWTYPE;
BEGIN

   IF NOT C%ISOPEN THEN
    OPEN C;
    dbms_output.put_line('S-a deschis');
   END IF;
   LOOP
   FETCH C INTO R;
   EXIT WHEN C%NOTFOUND or C%ROWCOUNT>3;
   dbms_output.put_line('ANGAJATUL '||R.NUME||' '||R.PRENUME||' '||R.SALARIUL);
   END LOOP;
   CLOSE C;
END;
/

DECLARE
  CURSOR C IS SELECT NUME,PRENUME,SALARIUL FROM ANGAJATI WHERE ID_ANGAJAT IN (SELECT
ID_ANGAJAT FROM COMENZI GROUP BY ID_ANGAJAT
  HAVING COUNT(*)>=3) order by SALARIUL desc FETCH FIRST 4 ROWS WITH TIES;
  R C%ROWTYPE;
BEGIN

   IF NOT C%ISOPEN THEN
    OPEN C;
    dbms_output.put_line('S-a deschis');
   END IF;
   LOOP
   FETCH C INTO R;
   EXIT WHEN C%NOTFOUND;
   dbms_output.put_line('ANGAJATUL '||R.NUME||' '||R.PRENUME||' '||R.SALARIUL);
   END LOOP;
   dbms_output.put_line(C%ROWCOUNT);
   CLOSE C;
END;
/


DECLARE
  CURSOR C IS SELECT NUME,PRENUME,SALARIUL FROM ANGAJATI WHERE ID_ANGAJAT IN (SELECT
ID_ANGAJAT FROM COMENZI GROUP BY ID_ANGAJAT
  HAVING COUNT(*)>=3) order by SALARIUL desc FETCH FIRST 4 ROWS WITH TIES;
```

```
BEGIN
  --OPEN C;
  FOR R IN C LOOP
   dbms_output.put_line(C%ROWCOUNT||'- ANGAJATUL '||R.NUME||' '||R.PRENUME||' '||R.SALARIUL);
  END LOOP;
  dbms_output.put_line(C%ROWCOUNT);
  --CLOSE C;
END;
/
-- CURSOR INLINE
BEGIN
  FOR R IN (SELECT NUME,PRENUME,SALARIUL FROM ANGAJATI WHERE ID_ANGAJAT IN (SELECT
ID_ANGAJAT FROM COMENZI GROUP BY ID_ANGAJAT
              HAVING COUNT(*)>=3) order by SALARIUL desc FETCH FIRST 4 ROWS WITH TIES) LOOP
   dbms_output.put_line('ANGAJATUL '||R.NUME||' '||R.PRENUME||' '||R.SALARIUL);
  END LOOP;
END;
/
-- CURSORUL CU PARAMETRU

-- SA SE AFISEZE ANGAJATII DIN DEP CU CEI MAI MULTI ANGAJATI

DECLARE
  V_ID_DEP NUMBER;
  CURSOR C(P_ID_DEP NUMBER) IS SELECT NUME,PRENUME,SALARIUL,ID_DEPARTAMENT FROM
ANGAJATI
     WHERE ID_DEPARTAMENT=P_ID_DEP;
BEGIN
 SELECT ID_DEPARTAMENT INTO V_ID_DEP FROM ANGAJATI GROUP BY ID_DEPARTAMENT ORDER
BY COUNT(*) DESC FETCH FIRST 1 ROW ONLY;
 DBMS_OUTPUT.PUT_LINE('DEP CU CEI MAI MULTI ANG ESTE '||V_ID_DEP);
 FOR R IN C(V_ID_DEP) LOOP
    DBMS_OUTPUT.PUT_LINE(R.NUME||' '||R.PRENUME||' '||R.SALARIUL);
 END LOOP;
END;
/


DECLARE
  V_ID_DEP NUMBER;
BEGIN
 SELECT ID_DEPARTAMENT INTO V_ID_DEP FROM ANGAJATI GROUP BY ID_DEPARTAMENT ORDER
BY COUNT(*) DESC FETCH FIRST 1 ROW ONLY;
 DBMS_OUTPUT.PUT_LINE('DEP CU CEI MAI MULTI ANG ESTE '||V_ID_DEP);
 FOR R IN (SELECT NUME,PRENUME,SALARIUL,ID_DEPARTAMENT FROM ANGAJATI
     WHERE ID_DEPARTAMENT=V_ID_DEP) LOOP
    DBMS_OUTPUT.PUT_LINE(R.NUME||' '||R.PRENUME||' '||R.SALARIUL);
 END LOOP;
END;
/

-- CURSORUL FOR UPDATE

-- SA MAREASCA CU 5% SALARIILE ANG DIN DEP CU CEI MAI MULTI ANG
```

```
DECLARE
  V_ID_DEP NUMBER;
  CURSOR C(P_ID_DEP NUMBER) IS SELECT NUME,PRENUME,SALARIUL,ID_DEPARTAMENT FROM
ANGAJATI
      WHERE ID_DEPARTAMENT=P_ID_DEP FOR UPDATE NOWAIT;--WAIT 1;
BEGIN
 SELECT ID_DEPARTAMENT INTO V_ID_DEP FROM ANGAJATI GROUP BY ID_DEPARTAMENT ORDER
BY COUNT(*) DESC FETCH FIRST 1 ROW ONLY;
 DBMS_OUTPUT.PUT_LINE('DEP CU CEI MAI MULTI ANG ESTE '||V_ID_DEP);
 FOR R IN C(V_ID_DEP) LOOP
    UPDATE ANGAJATI SET SALARIUL=SALARIUL*1.05 WHERE CURRENT OF C;
    DBMS_OUTPUT.PUT_LINE(R.NUME||' '||R.PRENUME||' '||R.SALARIUL);
 END LOOP;
 DBMS_OUTPUT.NEW_LINE;
  FOR R IN C(V_ID_DEP) LOOP
    DBMS_OUTPUT.PUT_LINE(R.NUME||' '||R.PRENUME||' '||R.SALARIUL);
 END LOOP;
   EXCEPTION WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE(SQLERRM);
END;
/
SELECT ID_ANGAJAT,NUME,PRENUME,SALARIUL,ID_DEPARTAMENT FROM ANGAJATI WHERE
ID_DEPARTAMENT=50;

declare
type t_c is ref cursor return angajati%rowtype;
c t_c;
 cursor d is select id_departament from angajati group by id_departament order by count(*) desc;
 n number(3);
 r angajati%rowtype;
begin
 open d;
 fetch d into n;
 close d;
 open c for select * from angajati where id_departament=n order by salariul desc;
 loop
 fetch c into r;
 exit when c%notfound;
 dbms_output.put_line(c%ROWCOUNT||' '||r.id_departament||' '||r.prenume||' '||r.nume||' '||r.salariul);
 end loop;
 close c;
end;
/

declare
 c SYS_REFCURSOR;
 cursor d is select id_departament from angajati group by id_departament order by count(*) desc;
 n number(3);
 r angajati%rowtype;
 r2 departamente%rowtype;
begin
 open d;
 fetch d into n;
```

```
 close d;
 open c for 'select * from angajati where id_departament=:1 order by salariul desc' using n;
 loop
 fetch c into r;
 exit when c%notfound;
 dbms_output.put_line(r.id_departament||' '||r.prenume||' '||r.nume||' '||r.salariul);
 end loop;
 close c;

 OPEN C for 'select * from departamente';
  loop
 fetch c into r2;
 exit when c%notfound;
 dbms_output.put_line(r2.id_departament||' '||r2.denumire_departament);
 end loop;
 close c;

end;
/

VARIABLE dept_sel REFCURSOR /*might not work in PLSQL Dev*/
BEGIN
   OPEN :dept_sel FOR SELECT * FROM DEPARTaMENTE;
 END;
/
PRINT dept_sel


create or replace function syscursor_dep return sys_refcursor is
 c sys_refcursor;
 cursor d is select id_departament from angajati group by id_departament order by count(*) desc;
 n number(3);
 r angajati%rowtype;
begin
 open d;
 fetch d into n;
 close d;
 open c for 'select * from angajati where id_departament=:1 order by salariul desc' using n;
 return c;
end;
/

var rc refcursor;
exec :rc:=syscursor_dep;
print rc

declare
   c sys_refcursor;
   r angajati%rowtype;
begin
   c:=syscursor_dep;
   loop
     fetch c into r;
     exit when c%notfound;
```

```
    dbms_output.put_line(r.nume);
   end loop;
end;
/


declare
 p varchar2(128);
 x varchar2(50);
begin
 p:='begin dbms_output.put_line("Message="||:e); end;';
 x:='ABdsdsaC';
 execute immediate p using x;
end;
/
drop table test11;

begin
  -- in blocuri plsql nu pot folosi direct instructiuni DDL sau DCL
  execute immediate 'create table test11(n number)';
  execute immediate 'insert into test11 values (10)';
end;
/

desc test11;

select * from test11;
```

```
/*Pachete
  - grupeaza functii si proceduri inrudite (e.g, cele pentru o apicatie de contabilitate)
  - sunt folosite frecvent impreuna
  - se pot stoca si variabile, constante, exceptii, cursoare, tipuri de date (type)
  - au 2 componente:
    - interfata publica (specificatiile) - antetul fct & procedurilor publice (obligatorie)
    - partea privata (body) - codul pentru functiile si procedurile publice
          + eventual functii si procedure private (pot fi apelate doar din interiorul pachetului)
          (optionala)

  */

CREATE OR REPLACE PACKAGE PACHET1 IS
 COTA_TVA CONSTANT NUMBER := 19;

 FUNCTION GET_VALOARE_CU_TVA(P_VALOARE NUMBER) RETURN NUMBER;
 PROCEDURE AFISEAZA_MESAJ(P_MESAJ VARCHAR2);

END;
/




CREATE OR REPLACE PACKAGE BODY PACHET1 IS

 FUNCTION GET_VALOARE_CU_TVA(P_VALOARE NUMBER) RETURN NUMBER
   IS
   BEGIN
     RETURN P_VALOARE*(1+COTA_TVA/100);
   END;
   PROCEDURE AFISEAZA_MESAJ(P_MESAJ VARCHAR2) IS
   BEGIN
    DBMS_OUTPUT.PUT_LINE(P_MESAJ);
   END;

END;
/


SET SERVEROUTPUT ON
BEGIN
   DBMS_OUTPUT.PUT_LINE('COTA TVA STANDARD '||PACHET1.COTA_TVA);
   PACHET1.AFISEAZA_MESAJ(PACHET1.COTA_TVA);

   PACHET1.AFISEAZA_MESAJ(PACHET1.GET_VALOARE_CU_TVA(P_VALOARE=>100));
   DBMS_OUTPUT.PUT_LINE('COTA SUPLIM '||PACHET1.COTA_SUPL);
   PACHET1.COTA_SUPL:=PACHET1.COTA_SUPL+1;
   -- VALORILE DIN VARIABILELE DE PACHET SUNT PERSISTENTE PE SESIUNEA CURENTA
   DBMS_OUTPUT.PUT_LINE('COTA SUPLIM '||PACHET1.COTA_SUPL);
   PACHET1.AFISEAZA_MESAJ(PACHET1.GET_VALOARE_CU_TVA(P_VALOARE=>100));
   DBMS_OUTPUT.PUT_LINE('7+5= '||PACHET1.ADUNA_NUMERE(7,5));
   DBMS_OUTPUT.PUT_LINE('7+5+3= '||PACHET1.ADUNA_NUMERE(7,5,3));
```

```
  DBMS_OUTPUT.PUT_LINE(PACHET1.get_val_comanda(2382));
END;
/

SELECT * FROM RAND_COMENZI;


-- SA SE ADAUGE LA PACHET1 O FUNCTIE CARE PRIMESTE ID-UL UNEI COMENZI SI
-- RETURNEAZA VALOAREA TOTALA A RESPECTIVEI COMENZI

SELECT C.*,PACHET1.GET_VAL_COMANDA(ID_COMANDA) VAL_COMANDA FROM COMENZI C
WHERE PACHET1.GET_VAL_COMANDA(ID_COMANDA)>50000;

SELECT DATA,PACHET1.GET_VAL_COMANDA(ID_COMANDA) VAL_COMANDA FROM COMENZI C
WHERE PACHET1.GET_VAL_COMANDA(ID_COMANDA)>50000;




================================================================

create or replace PACKAGE PACHET1 IS
 COTA_TVA CONSTANT NUMBER := 19;
 COTA_SUPL NUMBER := 1; -- VALORILE POT FI ACCESATE SI MODIFCATE DIRECT,
                        -- VALOAREA MODIFICATA FIIND VIZIBILA PENTRU SESIUNEA CURENTA

 FUNCTION GET_VALOARE_CU_TVA(P_VALOARE NUMBER) RETURN NUMBER;
 PROCEDURE AFISEAZA_MESAJ(P_MESAJ VARCHAR2);
 FUNCTION ADUNA_NUMERE(N1 NUMBER, N2 NUMBER) RETURN NUMBER;
 FUNCTION ADUNA_NUMERE(N1 NUMBER, N2 NUMBER, N3 NUMBER) RETURN NUMBER;
 -- PUTEM AVEA FUNCTII SI PROCEDURI CU ACEEASI DENUMIRE DAR CU PARAMETRII DIFERITI
    -- SUPRAINCARCARE
 FUNCTION GET_VAL_COMANDA(P_ID_COMANDA NUMBER) RETURN NUMBER;
END;


create or replace PACKAGE BODY PACHET1 IS
  FUNCTION ADUNA_NUMERE(N1 NUMBER, N2 NUMBER) RETURN NUMBER IS
  BEGIN
    RETURN N1+N2;
  END;
 -- FUNCTIILE PRIVATE TREBUIE MAI INTAI DEFINITE SI APOI FOLOSITE (FORWARD DECLARATION)

  FUNCTION ADUNA_NUMERE(N1 NUMBER, N2 NUMBER, N3 NUMBER) RETURN NUMBER IS
  BEGIN
    RETURN N1+N2+N3;
  END;

 FUNCTION GET_VALOARE_CU_TVA(P_VALOARE NUMBER) RETURN NUMBER
  IS
  BEGIN
    RETURN P_VALOARE*(1+ADUNA_NUMERE(COTA_TVA,COTA_SUPL)/100);
  END;
  PROCEDURE AFISEAZA_MESAJ(P_MESAJ VARCHAR2) IS
```

```
  BEGIN
    DBMS_OUTPUT.PUT_LINE(P_MESAJ);
  END;

 FUNCTION GET_VAL_COMANDA(P_ID_COMANDA NUMBER) RETURN NUMBER IS
   V_VAL NUMBER;
 BEGIN
   SELECT SUM(PRET*CANTITATE) INTO V_VAL FROM RAND_COMENZI WHERE
ID_COMANDA=P_ID_COMANDA;
   RETURN NVL(V_VAL,0);

 END;

END;
```

SGBD Oracle

Pachete

Packages

- Are schema objects that group logically related PL/SQL types, variables and subprograms.
- Have two parts:
  - Package specs (mandatory)
    - Public objects – it’s the interface with other programs
  - Package body (optional)
    - Public + private objects – contains the actual code
    - Contains the optional initialization part, which typically holds statements that initialize package variables.
    - The initialization part of a package is run just once per session, the first time it’s referenced

Advantages

- Separation between the public and the private objects
- Modularity and easier application design
- The private or even the public part can be encoded using *wrap* (https://docs.oracle.com/cd/B28359_01/appdev.111/b28370/wrap.htm#LNPLS01601)
- When a package object is referenced, the package is loaded in the SGA and it's ready to be executed
- Furthers calls to that package don't require I/O with the disk
- We can store overloaded subprograms

Question

- Can we have packages that only have specs?
- Can we have packages that only have a body?
- Can the package body be modified without modifying the specs?

Defining packages

- When the specs are modified the body is invalidated
- The body can be successfully compiled only after the specs are compiled successfully

Packages

```
create or replace PACKAGE emp_pack AS
  TYPE EmpRecTyp IS RECORD (id number, sal number, pren varchar2(25));
  CURSOR disp_emp RETURN EmpRecTyp;
  PROCEDURE increase_sal (
    id    NUMBER,
    sal_inc    NUMBER,
    new_sal out   NUMBER
    );
  PROCEDURE del_employee (id NUMBER);
END emp_pack;
/
```

Packages

```
create or replace PACKAGE BODY emp_pack AS
  CURSOR disp_emp RETURN EmpRecTyp IS
    SELECT id_angajat,salariul,prenume FROM angajati ORDER BY salariul DESC;
  PROCEDURE increase_sal (
    id    NUMBER,
    sal_inc    NUMBER,
    new_sal out   NUMBER
    ) IS
  BEGIN
   update angajati
     SET salariul = salariul +sal_inc where id_angajat=id
     returning salariul into new_sal;
  END increase_sal;
  PROCEDURE del_employee (id NUMBER) IS
  BEGIN
    DELETE FROM angajati WHERE id_angajat = id;
  END del_employee;
END emp_pack;
/
```

Testing

```
set SERVEROUTPUT ON
declare
 r emp_pack.EmpRecTyp;
begin
 if not emp_pack.disp_emp%isopen then open emp_pack.disp_emp;
 end if;

 loop
  fetch emp_pack.disp_emp into r;
  exit when emp_pack.disp_emp%notfound;
  dbms_output.put_line(r.pren||' '||r.sal);
 end loop;
 close emp_pack.disp_emp;
end;
/
```

The initialization part

```
create or replace PACKAGE BODY emp_pack AS
   vat  float;
[….]

   function get_with_vat(val number) return number is
    begin
     return val+val*vat;
    end;
begin
 vat:=0.2;
 insert into regiuni values(reg_id.nextval,'Region '||reg_id.currval);
END emp_pack;
```

How many regions get added?

```
begin
 dbms_output.put_line(emp_pack.get_with_vat(100));
 dbms_output.put_line(emp_pack.get_with_vat(110));
 dbms_output.put_line(emp_pack.get_with_vat(120));
end;
/
```

If it doesn’t work, what do we have to add?

```
drop package PACHET1;
create or replace PACKAGE PACHET1 IS
 COTA_TVA CONSTANT NUMBER := 19;
 COTA_SUPL NUMBER := 2; -- VALORILE POT FI ACCESATE SI MODIFCATE DIRECT,
                    -- VALOAREA MODIFICATA FIIND VIZIBILA PENTRU SESIUNEA CURENTA

 FUNCTION GET_VALOARE_CU_TVA(P_VALOARE NUMBER) RETURN NUMBER;
 PROCEDURE AFISEAZA_MESAJ(P_MESAJ VARCHAR2);
 FUNCTION ADUNA_NUMERE(N1 NUMBER, N2 NUMBER) RETURN NUMBER;
 FUNCTION SCADE_NUMERE(N1 NUMBER, N2 NUMBER) RETURN NUMBER;
 FUNCTION ADUNA_NUMERE(N1 NUMBER, N2 NUMBER, N3 NUMBER) RETURN NUMBER;
 -- PUTEM AVEA FUNCTII SI PROCEDURI CU ACEEASI DENUMIRE DAR CU PARAMETRII DIFERITI
    -- SUPRAINCARCARE
 FUNCTION GET_VAL_COMANDA(P_ID_COMANDA NUMBER) RETURN NUMBER;
END;

/
create or replace PACKAGE BODY PACHET1 IS
   FUNCTION ADUNA_NUMERE(N1 NUMBER, N2 NUMBER) RETURN NUMBER IS
   BEGIN
      RETURN N1+N2;
   END;
 -- FUNCTIILE PRIVATE TREBUIE MAI INTAI DEFINITE SI APOI FOLOSITE (FORWARD DECLARATION)

  FUNCTION ADUNA_NUMERE(N1 NUMBER, N2 NUMBER, N3 NUMBER) RETURN NUMBER IS
   BEGIN
      RETURN N1+N2+N3;
   END;

   FUNCTION SCADE_NUMERE(N1 NUMBER, N2 NUMBER) RETURN NUMBER IS
   BEGIN
     RETURN N1-N2;
   END;

 FUNCTION GET_VALOARE_CU_TVA(P_VALOARE NUMBER) RETURN NUMBER
   IS
   BEGIN
      RETURN P_VALOARE*(2+ADUNA_NUMERE(COTA_TVA,COTA_SUPL)/100);
   END;
   PROCEDURE AFISEAZA_MESAJ(P_MESAJ VARCHAR2) IS
   BEGIN
     DBMS_OUTPUT.PUT_LINE(P_MESAJ);
   END;

  FUNCTION GET_VAL_COMANDA(P_ID_COMANDA NUMBER) RETURN NUMBER IS
    V_VAL NUMBER;
  BEGIN
    SELECT SUM(PRET*CANTITATE) INTO V_VAL FROM RAND_COMENZI WHERE
ID_COMANDA=P_ID_COMANDA;
    RETURN NVL(V_VAL,0);
  END;

  begin
```

```
    COTA_SUPL := 18;
    dbms_output.put_line('Am apelat pachetul');
END;
/
set SERVEROUTPUT on
begin

 dbms_output.put_line('Cota inainte: '||PACHET1.COTA_SUPL);
 PACHET1.COTA_SUPL:=PACHET1.COTA_SUPL+1;
 dbms_output.put_line('Cota dupa: '||PACHET1.COTA_SUPL);
 dbms_output.new_line;
 dbms_output.put_line('Val comenzi: '||pachet1.get_val_an(2019));
end;
/

select c.*,pachet1.get_val_an(extract (year from data)) val_an,pachet1.get_val_comanda(c.id_comanda) val_c from
comenzi c
where pachet1.get_val_an(extract (year from data))>370000
order by pachet1.get_val_an(extract (year from data)) desc;


-- SA SE ADAUGE O FUNCTIE CARE PRIMESTE UN AN CALENDARISTIC SI RETURNEAZA VALOAREA
TOTALA A COMENZILOR DIN ANUL RESPECTIV


================================
create or replace PACKAGE PACHET1 IS
 COTA_TVA CONSTANT NUMBER := 19;
 COTA_SUPL NUMBER := 1; -- VALORILE POT FI ACCESATE SI MODIFCATE DIRECT,
                    -- VALOAREA MODIFICATA FIIND VIZIBILA PENTRU SESIUNEA CURENTA

 FUNCTION GET_VALOARE_CU_TVA(P_VALOARE NUMBER) RETURN NUMBER;
 PROCEDURE AFISEAZA_MESAJ(P_MESAJ VARCHAR2);
 FUNCTION ADUNA_NUMERE(N1 NUMBER, N2 NUMBER) RETURN NUMBER;
 FUNCTION ADUNA_NUMERE(N1 NUMBER, N2 NUMBER, N3 NUMBER) RETURN NUMBER;
 -- PUTEM AVEA FUNCTII SI PROCEDURI CU ACEEASI DENUMIRE DAR CU PARAMETRII DIFERITI
    -- SUPRAINCARCARE
 FUNCTION GET_VAL_COMANDA(P_ID_COMANDA NUMBER) RETURN NUMBER;
 FUNCTION GET_VAL_AN(P_AN NUMBER) RETURN NUMBER;
END;

create or replace PACKAGE BODY PACHET1 IS
   FUNCTION ADUNA_NUMERE(N1 NUMBER, N2 NUMBER) RETURN NUMBER IS
   BEGIN
      RETURN N1+N2;
   END;
 -- FUNCTIILE PRIVATE TREBUIE MAI INTAI DEFINITE SI APOI FOLOSITE (FORWARD DECLARATION)

  FUNCTION ADUNA_NUMERE(N1 NUMBER, N2 NUMBER, N3 NUMBER) RETURN NUMBER IS
   BEGIN
      RETURN N1+N2+N3;
   END;

   FUNCTION SCADE_NUMERE(N1 NUMBER, N2 NUMBER) RETURN NUMBER IS
```

```
  BEGIN
    RETURN N1-N2;
  END;

 FUNCTION GET_VALOARE_CU_TVA(P_VALOARE NUMBER) RETURN NUMBER
  IS
  BEGIN
     RETURN P_VALOARE*(2+ADUNA_NUMERE(COTA_TVA,COTA_SUPL)/100);
  END;
  PROCEDURE AFISEAZA_MESAJ(P_MESAJ VARCHAR2) IS
  BEGIN
    DBMS_OUTPUT.PUT_LINE(P_MESAJ);
  END;

  FUNCTION GET_VAL_COMANDA(P_ID_COMANDA NUMBER) RETURN NUMBER IS
    V_VAL NUMBER;
  BEGIN
    SELECT SUM(PRET*CANTITATE) INTO V_VAL FROM RAND_COMENZI WHERE
ID_COMANDA=P_ID_COMANDA;
    RETURN NVL(V_VAL,0);
  END;

  FUNCTION GET_VAL_AN(P_AN NUMBER) RETURN NUMBER IS
     V_VAL NUMBER;

  BEGIN
    SELECT SUM(PRET*CANTITATE) INTO V_VAL FROM RAND_COMENZI R JOIN COMENZI A
      ON R.ID_COMANDA=A.ID_COMANDA
    WHERE EXTRACT(YEAR FROM DATA)=P_AN;
    RETURN NVL(V_VAL,0);
  END;

  begin
    COTA_SUPL := 18;
    dbms_output.put_line('Am apelat pachetul');
END;
```

SGBD Oracle

Declansatori

Types of Triggers

- A trigger:
  - Is a PL/SQL block or a PL/SQL procedure associated with a table, view, schema, or database
  - Executes implicitly whenever a particular event takes place
  - Can be either of the following:
    - Application trigger: Fires whenever an event occurs with a particular application (e.g, dynamic actions in Apex)
    - Database trigger: Fires whenever a data event (such as DML) or system event (such as logon or shutdown) occurs on a schema or database

Guidelines for Designing Triggers

- Do not define triggers to duplicate or replace the functionality already built into the Oracle database. For example, implement integrity rules using declarative constraints, not triggers. To remember the design order for a business rule:
  - Use built-in constraints in the Oracle server, such as primary key, and so on;
  - Develop database triggers;
  - Use GUI validations (e.g., in Oracle Apex).
- Excessive use of triggers can result in complex interdependencies, which may be difficult to maintain. Use triggers when necessary, and be aware of recursive and cascading effects.

DML Triggers types

- A statement trigger fires once for a DML statement.
- A row level trigger fires once for each row affected

```
CREATE [OR REPLACE] TRIGGER trigger_name
 timing
 event1 [OR event2 OR event3]
ON object_name
[[REFERENCING OLD AS old | NEW AS new]
 FOR EACH ROW
 [WHEN (condition)]]
trigger_body
```

Types of DML Triggers

- The trigger type determines whether the body executes for each row or only once for the triggering statement.
  - A statement trigger:
    - Executes once for the triggering event
    - Is the default type of trigger
    - Fires once even if no rows are affected at all
  - A row trigger:
    - Executes once for each row affected by the triggering event
    - Is not executed if the triggering event does not affect any rows
    - Is indicated by specifying the `FOR EACH ROW` clause

Trigger timing

- When should the trigger fire?
  - `BEFORE`: Execute the trigger body before the triggering DML event on a **table.**
  - `AFTER`: Execute the trigger body after the triggering DML event on a **table**.
  - `INSTEAD OF`: Execute the trigger body instead of the triggering statement. This is used for **views** that are not otherwise modifiable.
- Note: If multiple triggers with the same timing (before/after/instead of) are defined for the same object, then the order of firing triggers is arbitrary.

Using Conditional Predicates (statement level)

```
CREATE OR REPLACE TRIGGER secure_emp BEFORE
INSERT OR UPDATE OR DELETE ON angajati
BEGIN
 IF (TO_CHAR(SYSDATE,'DY') IN ('SAT','SUN')) OR  (TO_CHAR(SYSDATE,'HH24')
      NOT BETWEEN '08' AND '18') THEN
   IF DELETING THEN RAISE_APPLICATION_ERROR(-20502,'You may delete from EMPLOYEES
table only during business hours.');
   ELSIF INSERTING THEN RAISE_APPLICATION_ERROR(-20500,'You may insert into EMPLOYEES
table only during business hours.');
   ELSIF UPDATING('SALARY') THEN
    RAISE_APPLICATION_ERROR(-20503, 'You may update SALARY only during business hours.');
   ELSE RAISE_APPLICATION_ERROR(-20504,'You may update EMPLOYEES table only during
normal hours.');
   END IF;
 END IF;
END;
/
```

Creating a DML Row Trigger

```
CREATE OR REPLACE TRIGGER restrict_salary
BEFORE INSERT OR UPDATE OF salariul ON angajati
FOR EACH ROW
BEGIN
  IF NOT (:NEW.id_functie IN ('AD_PRES', 'AD_VP'))
     AND :NEW.salariul > 15000 THEN
    RAISE_APPLICATION_ERROR (-20202,
      'Employee cannot earn more than $15,000.');
  END IF;
END;
/
```

Using OLD and NEW Qualifiers

- Using OLD and NEW Qualifiers
  - Within a ROW trigger, reference the value of a column before and after the data change by prefixing it with the OLD and NEW qualifiers.
  - Usage notes:
    - The OLD and NEW qualifiers are available only in ROW triggers.
    - Prefix these qualifiers with a colon (:) in every SQL and PL/SQL statement.
    - There is no colon (:) prefix if the qualifiers are referenced in the WHEN restricting condition.
  - **Note:** Row triggers can decrease the performance if you perform many updates on larger tables.

```
drop table audit_ang;
create table audit_ang(
user_name varchar2(15),
time_stamp date,
id number,
old_nume varchar2(15),
new_nume varchar2(15),
old_job varchar2(15),
new_job varchar2(15),
old_salariul number,
new_salariul number);
```

```
CREATE OR REPLACE TRIGGER
audit_ang_values
AFTER DELETE OR INSERT OR UPDATE ON
angajati
FOR EACH ROW
BEGIN
  INSERT INTO audit_ang(user_name,
time_stamp, id,
    old_nume, new_nume, old_job,
    new_job, old_salariul, new_salariul)
  VALUES (USER, SYSDATE, :OLD.id_angajat,
    :OLD.nume, :NEW.nume, :OLD.id_functie,
    :NEW.id_functie, :OLD.salariul,
:NEW.salariul);
END;
/
```

Restricting a Row Trigger: Example

```
CREATE OR REPLACE TRIGGER derive_commission_pct
BEFORE INSERT OR UPDATE OF salariul ON angajati
FOR EACH ROW
WHEN (NEW.id_functie = 'SA_REP')
BEGIN
 IF INSERTING THEN
   :NEW.comision := 0;
 ELSIF :OLD.comision IS NULL THEN
   :NEW.comision := 0;
 ELSE
   :NEW.comision := :OLD.comision+0.05;
 END IF;
END;
/
```

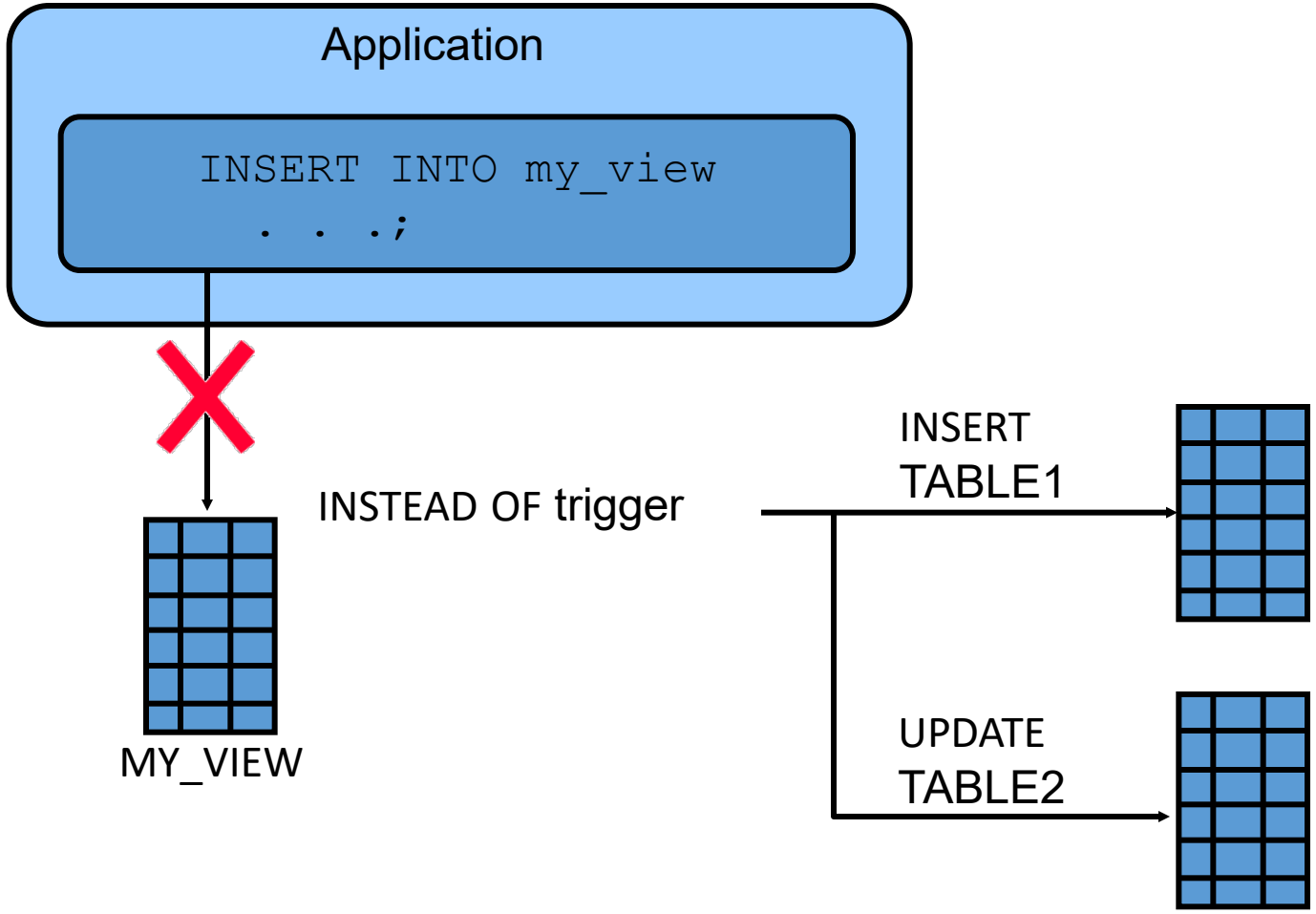
Trigger Execution Model

- A single DML statement can potentially fire up to four types of triggers:
  - `BEFORE` and `AFTER` statement triggers
  - `BEFORE` and `AFTER` row triggers
- A triggering event or a statement within the trigger can cause one or more integrity constraints to be checked. However, you can defer constraint checking until a `COMMIT` operation is performed.
- Triggers can also cause other triggers—known as cascading triggers—to fire.
- All actions and checks performed as a result of a SQL statement must succeed. If an exception is raised within a trigger and the exception is not explicitly handled, then all actions performed because of the original SQL statement are rolled back (including actions performed by firing triggers). This guarantees that integrity constraints can never be compromised by triggers.
- When a trigger fires, the tables referenced in the trigger action may undergo changes by other users' transactions. In all cases, a read-consistent image is guaranteed for the modified values that the trigger needs to read (query) or write (update).

INSTEAD OF Triggers

- Use INSTEAD OF triggers to modify data in which the DML statement has been issued against an **inherently nonupdatable view**.
- These triggers are called INSTEAD OF triggers because, unlike other triggers, the Oracle server fires the trigger instead of executing the triggering statement.
- These triggers are used to perform INSERT, UPDATE, and DELETE operations directly on the underlying tables. You can write INSERT, UPDATE, and DELETE statements against a view, and the INSTEAD OF trigger works invisibly in the background to make the right actions take place.
- **A view cannot be modified by normal DML statements** if the view query contains set operators, group functions, clauses such as GROUP BY, CONNECT BY, START, the DISTINCT operator, or joins. For example, if a view consists of more than one table, an insert to the view may entail an insertion into one table and an update to another.
- **Note:** If a view is inherently updatable and has INSTEAD OF triggers, then the triggers take precedence. INSTEAD OF triggers are row triggers. The CHECK option for views is not enforced when insertions or updates to the view are performed by using INSTEAD OF triggers. The INSTEAD OF trigger body must enforce the check.

Instead of triggers



Bibliography

- Oracle Database 12g: PL/SQL Fundamentals

Declansatori
- la nivel de instructiune/tabela before (tabele) insert (SE DECLANSEAZA 1 DATA/INSTRUCTIUNE)
- la nivel de rand after (tabele) update (SE DECLANSEAZA 1 DATA/RAND AFECTAT)
instead of (views) delete

```
CREATE OR REPLACE TRIGGER TRIG_INSTRUCTIUNE BEFORE UPDATE OR DELETE ON CLIENTI
BEGIN
  DBMS_OUTPUT.PUT_LINE('S-A DECLANSAT TRIGGER LA NIVEL DE INSTRUCTIUNE');
END;
/
SET SERVEROUTPUT ON

SELECT * FROM CLIENTI;

UPDATE CLIENTI SET LIMITA_CREDIT=LIMITA_CREDIT-100 WHERE EXTRACT(YEAR FROM
DATA_NASTERE)>1990;

CREATE OR REPLACE TRIGGER TRIG_RAND BEFORE UPDATE OR DELETE ON CLIENTI FOR EACH ROW
BEGIN
  DBMS_OUTPUT.PUT_LINE('S-A DECLANSAT TRIGGER LA NIVEL DE RAND PENTRU CLIENTUL
'||:NEW.ID_CLIENT);
  IF :NEW.ID_CLIENT>700 THEN
    RAISE_APPLICATION_ERROR(-20001,'ID MAI MARE DE 700');
  END IF;
END;
/

UPDATE CLIENTI SET LIMITA_CREDIT=LIMITA_CREDIT-100 WHERE EXTRACT(YEAR FROM
DATA_NASTERE)>1990;


SELECT COUNT(*) FROM PRODUSE;


-- SA SE CONSTRUIASCA UN TRIGGER CARE SA NU PERMITA MAI MULT DE 280 DE RANDURI IN TABELA
PRODUSE

CREATE OR REPLACE TRIGGER CHECK_NR_PROD AFTER INSERT ON PRODUSE
DECLARE
  N NUMBER;
BEGIN
  SELECT COUNT(*) INTO N FROM PRODUSE;
  IF N>280 THEN
    RAISE_APPLICATION_ERROR(-20001,'PREA MULTE PRODUSE'); -- NU FOLOSIM ROLLBACK
    -- RAISE-UL ARE ROLUL DE A ANULA OPERATIILE DE MODIFCARE
  END IF;
  --EXCEPTION WHEN OTHERS THEN NULL;
END;
/

DESCRIBE PRODUSE;

INSERT INTO PRODUSE(ID_PRODUS,DENUMIRE_PRODUS,CATEGORIE) VALUES (5001,'CPU I9','hardware8');
SELECT * FROM PRODUSE ORDER BY 1 DESC;


-- SA SE CONSRTUIASCA UN TRIGGER CARE SA NU PERMITA MODIFICAREA DE COMENZI DUPA ORA 8:00
```

```
-- UPDATING, DELETING, INSERTING RETURNEAZA TRUE SAU FALSE IN FUNCTIE DE INSTRUCTIUNEA CARE
A DECLANSAT TRIGGERUL

CREATE OR REPLACE TRIGGER CH_ORA_COM AFTER UPDATE OR INSERT OR DELETE ON COMENZI
BEGIN
  IF TO_CHAR(SYSDATE,'HH24:MI')>='08:00' AND UPDATING THEN
    RAISE_APPLICATION_ERROR(-20001,'NU SE PERMIT MODIFICARI DE COMENZI DUPA ORA 8');
  ELSIF TO_CHAR(SYSDATE,'HH24:MI')>='08:15' AND DELETING THEN
    RAISE_APPLICATION_ERROR(-20001,'NU SE PERMIT STERGERI DE COMENZI DUPA ORA 8:15');
  ELSIF TO_CHAR(SYSDATE,'HH24:MI')>='08:20' AND INSERTING THEN
    RAISE_APPLICATION_ERROR(-20001,'NU SE PERMIT ADAUGARI DE COMENZI DUPA ORA 8:20');
  END IF;
END;
/


SELECT * FROM COMENZI;

UPDATE COMENZI SET STARE_COMANDA=STARE_COMANDA+1 WHERE ID_COMANDA=2458;

DELETE FROM COMENZI
 WHERE ID_COMANDA=2458;

-- SA SE CONSRUIASCA UN TRIGGER CARE SA NU PERMITA MICSORAREA SALARIULUI UNUI ANG CARE A
INTERMEDIAT CEL PUTIN 3 COMENZI
-- CAND AVEM UN TRIGGER LA NIVEL DE RAND, NU INTEROGAM TABELA PE CARE E DEFINIT TRIGGERUL
=> MUTATING TRIGGER/TABLE
CREATE OR REPLACE TRIGGER CH_COM_ANG BEFORE UPDATE OF SALARIUL ON ANGAJATI FOR EACH
ROW
DECLARE
  N NUMBER;
BEGIN
    DBMS_OUTPUT.PUT_LINE('S-A DECLANSAT TRIGGERUL');
    IF :NEW.SALARIUL<:OLD.SALARIUL THEN
      SELECT COUNT(*) INTO N FROM COMENZI C -- JOIN ANGAJATI  A ON C.ID_ANGAJAT=A.ID_ANGAJAT
      WHERE C.ID_ANGAJAT=:NEW.ID_ANGAJAT;
       IF N>=3 THEN
         RAISE_APPLICATION_ERROR(-20003,'NU PUTEM MICSORA SALARIUL ACESTUI ANG');
       END IF;
    END IF;
END;
/

SELECT * FROM COMENZI ORDER BY ID_ANGAJAT;
UPDATE ANGAJATI SET SALARIUL=SALARIUL-100 WHERE ID_ANGAJAT=161;
UPDATE ANGAJATI SET COMISION=COMISION-0.1 WHERE ID_ANGAJAT=161;

CREATE OR REPLACE TRIGGER CH_COM_ANG BEFORE UPDATE OF SALARIUL ON ANGAJATI FOR EACH
ROW
WHEN (NEW.SALARIUL<OLD.SALARIUL)
DECLARE
  N NUMBER;
BEGIN
    DBMS_OUTPUT.PUT_LINE('S-A DECLANSAT TRIGGERUL');
      SELECT COUNT(*) INTO N FROM COMENZI C -- JOIN ANGAJATI  A ON C.ID_ANGAJAT=A.ID_ANGAJAT
      WHERE C.ID_ANGAJAT=:NEW.ID_ANGAJAT;
       IF N>=3 THEN
```

```
        RAISE_APPLICATION_ERROR(-20003,'NU PUTEM MICSORA SALARIUL ACESTUI ANG');
      END IF;

END;
/


CREATE OR REPLACE TRIGGER CH_COM_ANG BEFORE UPDATE OF SALARIUL ON ANGAJATI FOR EACH
ROW
WHEN (NEW.SALARIUL<OLD.SALARIUL)
DECLARE
  N NUMBER;
BEGIN
    DBMS_OUTPUT.PUT_LINE('S-A DECLANSAT TRIGGERUL');
      SELECT COUNT(*) INTO N FROM COMENZI C -- JOIN ANGAJATI  A ON C.ID_ANGAJAT=A.ID_ANGAJAT
      WHERE C.ID_ANGAJAT=:NEW.ID_ANGAJAT;
       IF N>=3 THEN
         :NEW.SALARIUL:=:OLD.SALARIUL; -- DOAR PENTRU TRIGGERI BEFORE
       END IF;

END;
/


UPDATE ANGAJATI SET SALARIUL=SALARIUL-100 WHERE ID_ANGAJAT=161;
SELECT * FROM ANGAJATI WHERE ID_ANGAJAT=161;
```

-- Sa se construiasca un declansator care sa nu

SELECT * FROM COMENZI WHERE ID_COMANDA IN (2424, 2414);

--2424, 2414

DELETE FROM ANGAJATI WHERE ID_ANGAJAT=153;

SELECT * FROM ANGAJATI WHERE ID_ANGAJAT=153;

-- SA SE CONSTRUIASCA UN TRIGGER CARE SA NU PERMITA STERGEREA UNUI ANGAJAT CARE A INTERMEDIAT MAI MULT DE 3 COMENZI
-- SA SE VERIFICE FUNCTIONAREA TRIGGERULUI

```
CREATE OR REPLACE TRIGGER CH_NR_COM BEFORE DELETE ON ANGAJATI FOR EACH ROW
DECLARE
 N NUMBER;
BEGIN
 SELECT COUNT(*) INTO N FROM COMENZI C --JOIN ANGAJATI A ON C.ID_ANGAJAT=A.ID_ANGAJAT
NU FACEM JOIN CU TABELA PE CARE E CONSTRUIT TRIGGERUL
 WHERE C.ID_ANGAJAT=:OLD.ID_ANGAJAT;
 IF N>3 THEN
    -- ROLLBACK; NU FOLOSIM ROLLBACK, FOLOSIM RAISE
   RAISE_APPLICATION_ERROR(-20001,'NU SE POATE STERGE ANGAJATUL '||:OLD.ID_ANGAJAT);
 END IF;

END;
/
```

SELECT * FROM COMENZI ORDER BY ID_ANGAJAT;
SET SERVEROUTPUT ON
DELETE FROM ANGAJATI WHERE ID_ANGAJAT=154;

ALTER TRIGGER CH_ORA_COM DISABLE;

ALTER TABLE COMENZI DROP CONSTRAINT COMENZI_ID_ANGAJAT_FK;

ALTER TABLE COMENZI ADD CONSTRAINT COMENZI_ID_ANGAJAT_FK FOREIGN KEY(ID_ANGAJAT) REFERENCES ANGAJATI;

-- SA SE CONSTRUIASCA O PROCEDURA INTR-UN PACHET CARE PRIMESTE CA PARAMETRII P_AN1 SI P_AN2. DACA P_AN1>P_AN2 SE VA RIDICA O EXCEPTIE DEFINITA DE UTILIZATOR
-- PROCEDURA VA CALCULA SI VA AFISA VALOAREA FIECAREI COMENZI DATE IN INTERVALUL P_AN1..P_AN2
-- VA RETURN PRINTR-UN PARAMETRU VALOAREA TOTALA A COMENZILOR AFISATE
-- VOM APELA PROCEDURA DINTR-UN BLOC ANONIM SI VOM TRATA EXPLICIT EXCEPTIA DIN PROCEDURA

2000..2004

CREATE OR REPLACE PACKAGE CALC_COMENZI IS

```
PROCEDURE GET_COMENZI_ANI(P_AN1 NUMBER, P_AN2 NUMBER, P_VAL OUT NUMBER);
  END;
/
CREATE OR REPLACE PACKAGE BODY CALC_COMENZI IS
     PROCEDURE GET_COMENZI_ANI(P_AN1 NUMBER, P_AN2 NUMBER, P_VAL OUT NUMBER) IS
      CURSOR C IS SELECT R.ID_COMANDA,SUM(PRET*CANTITATE) VALOARE FROM
        RAND_COMENZI R JOIN COMENZI C ON R.ID_COMANDA=C.ID_COMANDA WHERE
EXTRACT(YEAR FROM C.DATA) BETWEEN P_AN1 AND P_aN2
        GROUP BY R.ID_COMANDA;

    BEGIN
      IF P_AN1>P_AN2 THEN RAISE_APPLICATION_ERROR(-20001,'INTERVAL ERONAT');
      END IF;

      FOR R IN C LOOP
        DBMS_OUTPUT.PUT_LINE(R.ID_COMANDA||' ARE VALOAREA '||R.VALOARE);
        P_VAL:=NVL(P_VAL,0)+R.VALOARE;
      END LOOP;

    END;

END;
/

VARIABLE V_VAL NUMBER
EXEC CALC_COMENZI.GET_COMENZI_ANI(2017,2019, :V_VAL);

DECLARE
  E_INT EXCEPTION;
  PRAGMA EXCEPTION_INIT(E_INT,-20001);
  V_VAL NUMBER;
BEGIN
  CALC_COMENZI.GET_COMENZI_ANI(2017,2019, V_VAL);
  DBMS_OUTPUT.PUT_LINE('VALOARE TOTALA='||V_VAL);
  EXCEPTION WHEN E_INT THEN
    DBMS_OUTPUT.PUT_LINE('AM TRATAT EXCEPTIA');
END;
/
```

/*Structura examen -

Intrebari grila (4p) - 8-12 intrebari cu un singur raspuns corect (prima pagina)
Doua probleme de PL/SQL (5p) (a doua pagina, navigare secventiala)
- triggeri
- cursor, exceptii, colectii, functii, proceduri, pachete
1p din oficiu (1 grila tip doriti punctul din oficiu)

Timp de lucru 50-60 minute
Vor exista conturi dedicate pentru examen

Sa se construiasca un pachet (0.5p) care contine o functie care primeste ca parametru o categorie de produse si un an calendaristic.
Functia va returna valoarea totala a COMENZILOR PENTRU produsele din categoria respectiva in acel an (1.5p).
Functia va fi apelata la popularea unei colectii intr-un bloc anonim (1p). Se vor afisa elementele din colectie
Daca colectia nu contine niciun element, se va declansa o exceptie definita de utlizator. Sa se trateaza acea exceptie (0.5p).*/

```
CREATE OR REPLACE PACKAGE PACK_COMENZI IS
  FUNCTION GET_COM_CAT_AN(P_CATEGORIE VARCHAR2, P_AN NUMBER) RETURN NUMBER;
END;
/

CREATE OR REPLACE PACKAGE BODY PACK_COMENZI IS
  FUNCTION GET_COM_CAT_AN(P_CATEGORIE VARCHAR2, P_AN NUMBER) RETURN NUMBER IS
    V_VAL NUMBER;
  BEGIN
    SELECT SUM(PRET*CANTITATE) INTO V_VAL FROM COMENZI C JOIN RAND_cOMENZI R ON
C.ID_COMANDA=R.ID_COMANDA
                                        JOIN PRODUSE P ON R.ID_PRODUS=P.ID_PRODUS
    WHERE EXTRACT(YEAR FROM C.DATA)=P_AN AND CATEGORIE=P_CATEGORIE;
    RETURN NVL(V_VAL,0);
  END;
END;
/
DESCRIBE COMENZI;

SELECT
ID_PRODUS,DENUMIRE_PRODUS,CATEGORIE,PACK_COMENZI.GET_COM_CAT_AN(CATEGORIE,2019)
VALOARE FROM PRODUSE;

SELECT * FROM COMENZI;

SET SERVEROUTPUT ON
DECLARE
   TYPE T_RAND IS RECORD(
   ID_PRODUS NUMBER,
   DENUMIRE_PRODUS VARCHAR2(100),
   CATEGORIE VARCHAR2(50),
   VALOARE NUMBER);

   TYPE T_COLECTIE IS TABLE OF T_RAND;
```

```
   V T_COLECTIE;
   E_COLECTIE EXCEPTION;
   PRAGMA EXCEPTION_INIT(E_COLECTIE,-20001);
BEGIN
  SELECT
ID_PRODUS,DENUMIRE_PRODUS,CATEGORIE,PACK_COMENZI.GET_COM_CAT_AN(CATEGORIE,2019)
VALOARE
  BULK COLLECT INTO V
  FROM PRODUSE WHERE PRET_LISTA>10000;
  IF V.COUNT=0 THEN
     RAISE_APPLICATION_ERROR(-20001,'NU SUNT ELEMENTE IN COLECTIE');
  END IF;

  FOR I IN V.FIRST..V.LAST LOOP
    DBMS_OUTPUT.PUT_LINE(I||'->'||V(I).ID_PRODUS||' '||V(I).DENUMIRE_PRODUS||' '||V(I).CATEGORIE||'
'||V(I).VALOARE);
  END LOOP;

  EXCEPTION
   WHEN E_COLECTIE THEN
     DBMS_OUTPUT.PUT_LINE('A APARUT EXCEPTIA '||SQLERRM);
END;
/

/*
SA SE CONTRUIASCA UN TRIGGER CARE SA NU PERMITA MAI MULT DE 5 COMENZI IN ANUL
CURENT. SA SE TESTEZE TRIGGERUL.
*/




CREATE OR REPLACE TRIGGER RESTR_NR_COM BEFORE INSERT OR UPDATE ON COMENZI
DECLARE
   N NUMBER;
BEGIN
  SELECT COUNT(*) INTO N FROM COMENZI WHERE EXTRACT(YEAR FROM DATA)=EXTRACT(YEAR
FROM SYSDATE);
  IF N>5 THEN
    -- nu rollback
    RAISE_APPLICATION_ERROR(-20002,'MAI MULT DE 5 COMENZI IN ANUL CURENT');
  END IF;
END;
/

SELECT * FROM COMENZI ORDER BY DATA DESC;

INSERT INTO COMENZI values (5000,SYSDATE,'online',109,1,null);
INSERT INTO COMENZI values (5001,SYSDATE,'online',109,1,null);
INSERT INTO COMENZI values (5002,SYSDATE,'online',109,1,null);
INSERT INTO COMENZI values (5003,SYSDATE,'online',109,1,null);
INSERT INTO COMENZI values (5004,SYSDATE,'online',109,1,null);
INSERT INTO COMENZI values (5005,SYSDATE,'online',109,1,null);
INSERT INTO COMENZI values (5006,SYSDATE,'online',109,1,null);
```

--SA SE CONTRUIASCA UN TRIGGER CARE SA NU PERMITA COMENZI INTERMEDIATE DE UN ANG CU SALARIUL>15000. SA SE TESTEZE TRIGGERUL.

```
CREATE OR REPLACE TRIGGER VER_SAL_ANG BEFORE INSERT OR UPDATE ON COMENZI FOR EACH
ROW
DECLARE
   V_SAL NUMBER;
BEGIN
  DBMS_OUTPUT.PUT_LINE(:NEW.ID_ANGAJAT);
  SELECT SALARIUL INTO V_SAL FROM ANGAJATI A --JOIN COMENZI C ON
A.ID_ANGAJAT=C.ID_ANGAJAT -> table VLAD.COMENZI is mutating, trigger/function may not see it
    WHERE A.ID_ANGAJAT=:NEW.ID_ANGAJAT;
  IF V_SAL>15000   THEN
    RAISE_APPLICATION_ERROR(-20005,'ACEST ANG NU POATE INTERMDIA COMENZI');
  END IF;

END;
/

SELECT * FROM ANGAJATI ORDER BY SALARIUL DESC;

INSERT INTO COMENZI values (5007,SYSDATE-365,'online',109,1,102);

UPDATE COMENZI SET ID_ANGAJAT=102 WHERE ID_COMANDA=5000;
```