PL/SQL (Procedural Language Extension to SQL)

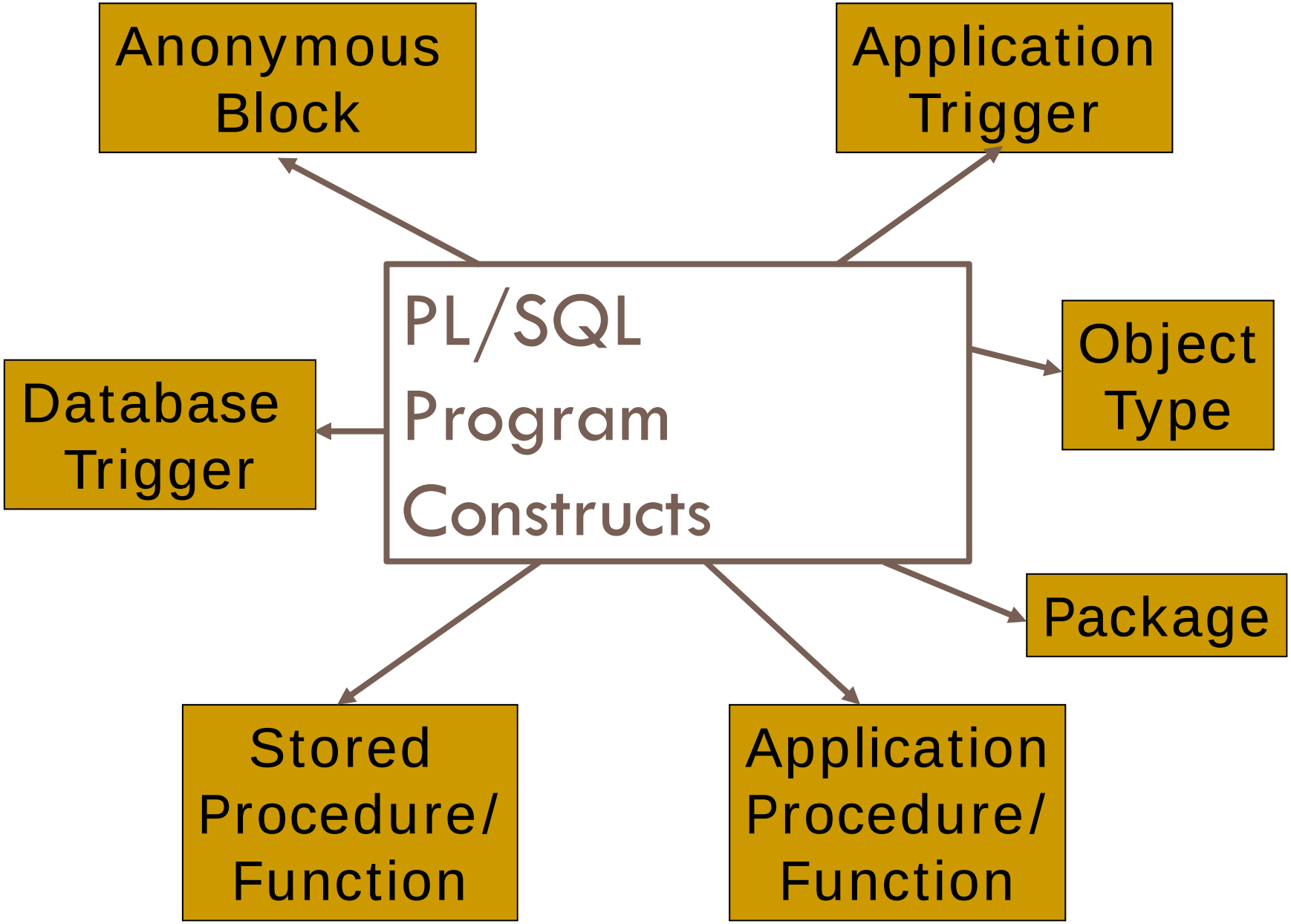
COMP810: Data Warehousing and Big Data

Learning Outcomes

- Introduction to PL/SQL
  - Be able to write a simple PL/SQL program
  - Be able to write simple stored procedures and stored functions
- Cursors (slide 36 onwards)
  - Understand the difference between implicit and explicit cursors
  - Be able to manipulate an explicit cursor in a PL/SQL program

PL/SQL

- PL/SQL is an extension to SQL with design features of programming languages
- Data manipulation (DML) and query statements of SQL are included within procedural units of code


Anonymous Block
Application Trigger
PL/SQL Program Constructs
Database Trigger
Object Type
Package
Stored Procedure/ Function
Application Procedure/ Function

PL/SQL Blocks

- PL/SQL code is built of Blocks, with a unique structure.
- There are two types of blocks in PL/SQL:

1. **Anonymous Blocks:** have no name (like scripts)
   - can be written and executed immediately in SQLPLUS
   - can be used in a trigger
2. **Named Blocks:**
   - Procedures
   - Functions

PL/SQL Block Structure

1. Anonymous Blocks

- DECLARE – Optional
  - Variable, cursors, constants
- BEGIN – Mandatory
  - SQL statements
  - PL/SQL statements
- EXCEPTION – Optional
  - Actions to perform when errors occur
- END; -Mandatory

NOTE the INTO clause this is mandatory and must occur Between the SELECT and FROM clauses


```
DECLARE
  qty_on_hand NUMBER(5);
BEGIN
  SELECT quantity INTO qty_on_hand
  FROM inventory
  WHERE product = 'TENNIS RACKET'
  FOR UPDATE OF quantity;
  IF qty_on_hand > 0 THEN -- check quantity
      UPDATE inventory SET quantity = quantity - 1
      WHERE product = 'TENNIS RACKET';
      INSERT INTO purchase_record VALUES ('Tennis
      racket purchased', SYSDATE);
  ELSE INSERT INTO purchase_record
      VALUES ('Out of tennis rackets', SYSDATE);
  END IF;
  COMMIT;
END;
```

Declaring PL/SQL Variables

- Syntax

  Identifier [CONSTANT] *datatype* [NOT NULL] {:= | DEFAULT *expr*];

- Examples

```
Declare
  v_hiredate   DATE;
  v_deptno     NUMBER(2) NOT NULL := 10;
  v_location   VARCHAR2(13) := 'Auckland';
  c_comm       CONSTANT NUMBER := 1400;
```

PL/SQL Datatypes

- VARCHAR2 (maximum_length)
- NUMBER [(precision, scale)]
- DATE
- CHAR [(maximum_length)]
- LONG/LONG RAW
- LOB Types - CLOB, BLOB (large objects)
- BOOLEAN
- BINARY_INTEGER
- PLS_INTEGER (identical to binary integer)

Reference variables - %Type Attribute

- Declare a variable based on a database column or another previously declared variable (very useful)
- Prefix %type with the database table and column or the previously declared variable name.
- Examples

```
V_ename          emp.ename%TYPE;
V_balance        NUMBER(7,2);
V_min_balance    v_balance%TYPE := 10;
```

Bind Variables

- Bind variables are also known as host variables.
- They are declared in the host SQL environment and are accessed by a PL/SQL block

```
SQL>  VARIABLE g_double NUMBER
SQL>  DECLARE
            v_num NUMBER(2);
      BEGIN
            v_num :=5;
            :g_double := v_num * 2;
      END;
      /
SQL>  PRINT g_double
```

PL/SQL procedure successfully completed.

G_DOUBLE
10

Substitution Variables


```
SQL>  VARIABLE g_double NUMBER
SQL>  DECLARE
              v_num NUMBER(2);
      BEGIN
              v_num := &p_num;
              :g_double := v_num * 2;
      END;
      /

Enter value for p_num: 7
old   4:    v_num := &p_num;
new   4:    v_num := 7;
PL/SQL procedure successfully completed.

SQL>  PRINT g_double
G_DOUBLE
  ----------
      14
```

Printing in PL/SQL – Using DBMS_OUTPUT.PUT_LINE


```
SQL>    SET SERVEROUTPUT ON
SQL>    VARIABLE g_double NUMBER
SQL>    DECLARE
                v_num NUMBER(2);
        BEGIN
                 v_num := &p_num;
                 :g_double := v_num * 2;
                DBMS_OUTPUT.PUT_LINE ('DOUBLE OF  '|| TO_CHAR(v_num) ||
                ' IS ' ||   TO_CHAR(:g_double));
        END;
         /

Enter value for p_num: 7
old   4:      v_num := &p_num;
new   4:      v_num := 7;
DOUBLE OF  7 IS 14

PL/SQL procedure successfully completed.
```

PL/SQL Decision Control Structures

- Use IF/THEN structure to execute code if condition is true
  - IF *condition* THEN

    commands that execute if condition is TRUE;

    END IF;
- If condition evaluates to NULL it is considered false

PL/SQL Decision Control Structures

- Use IF/THEN/ELSE to execute code if condition is true or false
  - IF *condition* THEN

    commands that execute if condition is TRUE;

    ELSE

    commands that execute if condition is FALSE;

    END IF;
- Can be nested – be sure to end nested statements

Control Structures
IF Statement

- Use IF/ELSIF to evaluate many conditions:
- Syntax (similar to Case statement in other languages)

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

Iterative Control
Basic LOOP

- Syntax

```
LOOP
  Statement1;
  …
  EXIT [WHEN condition];
END LOOP;
```

The Numeric FOR Loop

- Syntax

```
FOR counter in [REVERSE]
  Lower_bound..upper_bound LOOP
Statement1;
Statement2;
…
END LOOP;
```

Iterative Control
WHILE Loop

- Syntax

```
WHILE condition LOOP
  Statement1;
  Statement2;
  …
END LOOP;
```

Stored Procedures

- A procedure is a named PL/SQL block that performs an action (a set of related tasks).
- A procedure can be stored in the database, as a database object, for repeated execution.
  - A procedure can be invoked repeatedly (called by name from an application).
- Procedures can serve as building blocks for an application

Block Structure for PL/SQL Stored Procedures

2. Named Blocks (stored procedures)

```
Header
IS
   Declaration section
BEGIN
   Executable section
EXCEPTION
   Exception section
END;
```

SQL*Plus: Named Block example

```
SQL> ed test    --opens Notepad
SQL> create or replace procedure test
  2  is
  3  begin
  4  dbms_output.put_line ('Hello World');
  5  End test;
  6  /
Procedure created.
SQL> show errors
No errors.
SQL> set serveroutput on
SQL> execute test;
Hello World
PL/SQL procedure successfully completed.
```

Note

Another Named Block example

```
CREATE OR REPLACE PROCEDURE debit_account (acct_id INTEGER,
   amount NUMBER)
```



```
IS
v_old_balance NUMBER;
v_new_balance NUMBER;
e_overdrawn EXCEPTION;
BEGIN
   SELECT bal INTO v_old_balance FROM accts
       WHERE acct_no = acct_id;
   v_new_balance := v_old_balance - amount;
   IF v_new_balance < 0 THEN
       DBMS_OUTPUT.PUT_LINE (‘Account is Out of Funds');
   ELSE
       UPDATE accts SET bal = v_new_balance
       WHERE acct_no = acct_id;
       Commit;
   END IF;
END debit_account;
```

Subprogram Parameters

- Transfer values to and from the subprogram through parameters
- Subprogram parameters have three modes
  - IN, (the default) passes values to a subprogram
  - OUT, must be specified, returns values to the caller
  - IN OUT, must be specified, passes values to a subprogram and returns updated values to the caller

Parameter Examples

- IN Parameter Example
- OUT Parameter Example
- IN OUT Parameter Example

Exercise

- Given a product table description

```
SQL> desc prod
 Name                    Null?    Type
 ----------------- -------- ------------
 ProdID                  NOT NULL NUMBER(6)
 Description                      VARCHAR2(30)
```

- Create a procedure called DEL_PROD to delete a product. Include the necessary exception handling

Exercise

- Given a Employee table description

```
SQL> desc emp
 Name                      Null?    Type
 ------------------------- -------- -------------
 EMPNO                     NOT NULL NUMBER(4)
 ENAME                              VARCHAR2(10)
 JOB                                VARCHAR2(9)
 MGR                                NUMBER(4)
 HIREDATE                           DATE
 SAL                                NUMBER(7,2)
 COMM                               NUMBER(7,2)
 DEPTNO                             NUMBER(2)
```

- Create a procedure called Qemp to query the EMP table, and print the sal and job for an employee.

Invoking a Procedure From a Stored Procedure


```
CREATE OR REPLACE PROCEDURE process_emps
IS
  CURSOR emp_cursor IS
  SELECT empno
  FROM emp;
BEGIN
  FOR emp_rec IN emp_cursor LOOP
      raise_salary(emp_rec.empno);
  END LOOP;
COMMIT;
END process_emps;
/
```

Stored Functions

- A function is a named PL/SQL block that returns a value.
- A function can be stored in the database, as a database object, for repeated execution.
- A function can be called as part of an expression.

Stored Function Example


```
CREATE OR REPLACE FUNCTION get_sal (v_id
  IN emp.empno%TYPE)
RETURN NUMBER
IS
  v_salary emp.sal%TYPE :=0;
BEGIN
  SELECT sal
  INTO v_salary
  FROM emp
  WHERE empno = v_id;
  RETURN (v_salary);
END get_sal;
/
```

Executing Functions

- We can use a host variable to execute and test the function

```
VARIABLE g_salary NUMBER
EXECUTE :g_salary := get_sal(7934)

PRINT g_salary
```

- User-defined function can be called from any SQL expression wherever a built-in function can be called

Exercise

- Create a function called Q_PROD to return a product description to a host variable.

```
SQL> desc prod
 Name                    Null?    Type
 ----------------- -------- ----
 PRODID                  NOT NULL NUMBER(6)
 DESCRIP                          VARCHAR2(30)
```

- Create a function ANNUAL_COMP to return the annual salary when passed an employee's monthly salary and annual commission.

Stored Function Restrictions

- A user-defined function must be a ROW function not a GROUP function.
- A user-defined function only takes IN parameters.
- When called from a SELECT statement the function cannot modify any database tables.
- When called from an INSERT, UPDATE, or DELETE statement, the function cannot query or modify any database tables modified by that statement.

Comparing Procedures and Functions

Procedure	Function
Execute as a PL/SQL statement	Invoke as part of an expression
No RETURN datatype	Must contain a RETURN datatype
Can return one or more values	Must return a value

Programming Guidelines

- Document code with comments
- Develop a case convention for the code
- Develop naming convention for identifiers and other objects
- Enhance readability by indenting

Cursors

- Pointer to a memory location that the DBMS uses to process a SQL query
- Used to retrieve and manipulate database data

SQL Statements in PL/SQL

- Extract a row of data from the database by using the SELECT command. Only a single set of values can be returned (Implicit Cursor).
- Make changes to rows in the database by using DML (Data Manipulation Language) commands
- Control transactions with the COMMIT, ROLLBACK, or SAVEPOINT command.

SELECT Statements in PL/SQL


```
DECLARE
  V_deptno NUMBER(2);
  V_loc   VARCHAR2(15);
BEGIN
  SELECT deptno, loc
  INTO v_deptno, v_loc
  FROM dept
  WHERE dname = 'SALES'
…
END;
```

NOTE the INTO clause this is mandatory and must occur Between the SELECT and FROM clauses

SQL Cursor

- A cursor is an SQL work area
- Two type of cursors
  - Implicit cursors
  - Explicit cursors
- PL/SQL implicitly declares a cursor for all SQL data manipulation statements and queries that return only one row.
- For queries that return more than one row the programmer must explicitly declare a cursor
- IMPORTANT!

SQL Implicit Cursor Attributes

SQL%ROWCOUNT	Number of rows affected by the most recent SQL statement
SQL%FOUND	Boolean attribute that evaluates to TRUE if the most recent SQL statement affects one or more rows
SQL%NOTFOUND	Boolean attribute that evaluate to TRUE if the most recent SQL does not affect any rows
SQL%ISOPEN	Always evaluates to FALSE because PL/SQL closes implicit cursors immediately after they are executed

PL/SQL Records

- Similar in structure to records in a 3GL
- Convenient for fetching a row of data from a table for processing.

…

```
TYPE emp_record_type IS RECORD
   (ename          VARCHAR2(10),
    Job            VARCHAR2(9),
    Sal            NUMBER(7,2));
emp_record         emp_record_type;
```

…

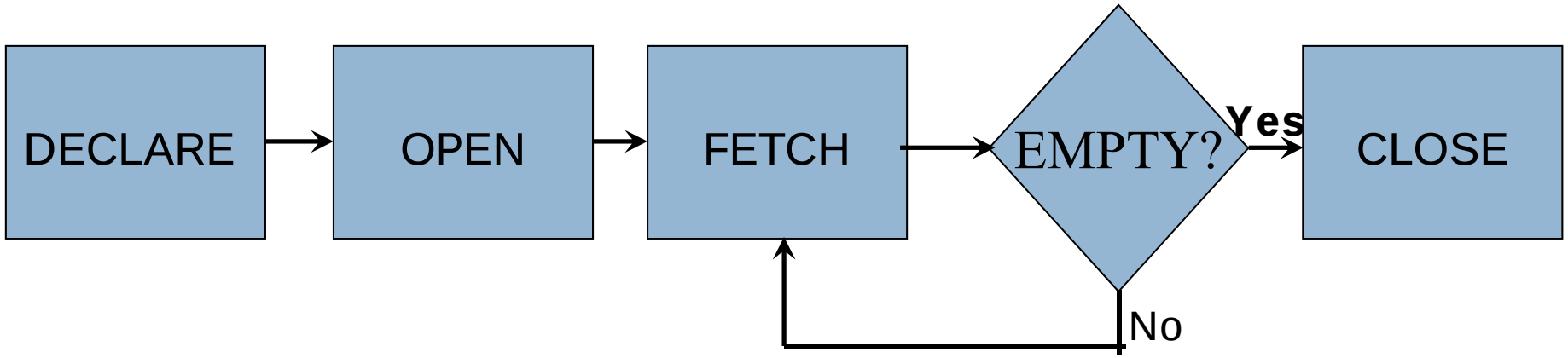
The %ROWTYPE Attribute

- Declare a variable according to a collection of columns in a database table or view.
- Prefix %ROWTYPE with the database table.
- Fields in the record take their name and datatypes from the columns of the table or view.

```
DECLARE
   Emp_record          emp%ROWTYPE;
```

Explicit Cursors

- Explicit cursors are named SQL work areas to manipulate queries returning more than one row.
- Use DECLARE, OPEN, FETCH and CLOSE to control explicit cursors.



Declaring the Cursor

- Syntax

```
CURSOR cursor_name IS
  Select_statement;
v_empno          emp.empno%Type
v_eName          emp.ename%Type
v_deptRec dept%RowType
```

- Examples

```
DECLARE
   CURSOR emp_cursor IS
      SELECT empno, ename
      FROM emp;
   CURSOR dept_cursor IS
      SELECT *
      FROM dept;
```

Opening the Cursor

- Syntax

```
OPEN cursor_name;
```

- Example

```
OPEN emp_cursor;
```

- Open the cursor to execute the query and identify the active set.
- The cursor now points to the first row in the active set

Fetching Data From the Cursor

- Syntax

```
FETCH cursor_name INTO [variable1, variable2,
  …]|record_name];
```

- Example

```
FETCH emp_cursor INTO v_empNo, v_eName;
FETCH dept_cursor INTO v_deptRec;
```

- Retrieve the current row values into variable(s) or record.
- Include the same number of variables.

Closing the Cursor

- Syntax

```
CLOSE cursor_name;
```

- Close the cursor after completing the processing of the rows

SQL Explicit Cursor Attributes

%ROWCOUNT	Evaluate to the total number of rows returned so far
%FOUND	Boolean attribute that evaluates to TRUE if the most recent fetch returns a row
%NOTFOUND	Boolean attribute that evaluate to TRUE if the most recent fetch does not return a row
%ISOPEN	Evaluates to TRUE if the cursor is open

Controlling Multiple Fetches

- Process several rows from an explicit cursor using a loop
- Fetch a row with each iteration
- Use the %NOTFOUND attribute to write a test for an unsuccessful fetch

Example Cursor


```
DECLARE
  V_empno       emp.empno%TYPE;
  V_ename       emp.ename%TYPE;
  CURSOR emp_cursor IS
    SELECT empno, ename FROM emp;
BEGIN
  OPEN emp_cursor;
  LOOP
    FETCH emp_cursor INTO v_empno, v_ename;
    EXIT WHEN emp_cursor%NOTFOUND;
    … do something with the cursor row
  END LOOP;
  CLOSE emp_cursor;
END;
```

Cursor FOR Loops

- Syntax

```
FOR record_name IN cursor_name LOOP
  Statement1;
  Statement2;
  …
END LOOP;
```

- *Implicit (automatic) open, fetch and close occur.*
- *The record is implicitly declared.*

Example Cursor For Loop


```
DECLARE
  CURSOR emp_cursor IS
    SELECT empno, ename
    FROM emp;
BEGIN
  FOR emp_record IN emp_cursor LOOP
    … do required processing with
    emp_record
  END LOOP;
END;
```

References

- Chapter 10, Oracle 11g PL/SQL User's Guide and Reference, For cursors:
  - Chapter 1 page 16,
  - Chapter 6 pages 6-17
  - Chapter 13 Language Elements