DATABASE PROGRAMMING WITH PL SQL 1 / 2

ORACLE ACADEMY

Contenido

Section 1 – Fundamentals

1-1 Introduction to PL/SQL

Procedural Constructs	Programming language features such as reusable/callable program units, modular blocks, cursors, constants, variables, assignment statements, conditional control statements, and loops.
PL/SQL	Oracle Corporations standard procedural language for relational databases which allows basic program logic and control flow to be combined with SQL statements.

MySQL	ORACLE

set @prom = (select avg(salary) from employees);
select avg(salary) into @prom from employees;
select @prom from dual;
	define prom = (select avg(salary) from employees);
	select &prom promedio from dual;

set @stmt = 'SELECT employee_id, salary FROM employees';
execute IMMEDIATE @stmt;
	DEFINE colname=salary ;
	SELECT employee_id, &colname FROM employees;

Procedural Language extension to SQL.

```
-- anonymous procedures
set serveroutput on
declare
  prom number;
begin
  select avg(salary) into prom from employees;
  dbms_output.put_line('promedio: ' || prom);
end;
/
```

The basic unit in a PL/SQL program is a block.

All PL/SQL programs consist of blocks as modules.

```
set serveroutput on;
declare
  numero number := 5;
  cadena varchar2(100);
  i integer default 1;
begin
  loop
    cadena := numero || ' x ' ||
        i || ' = ' || (numero*i);
    dbms_output.put_line(cadena);
    EXIT WHEN i >= 10;
    i := i + 1;
  end loop;
end;
```

```
CREATE OR REPLACE PROCEDURE tabla(numero NUMBER)
IS
cadena VARCHAR2(100);
BEGIN
    FOR i IN reverse 1..10 LOOP
      cadena := numero ||
         ' x ' || i || ' = ' || (numero*i);
      dbms_output.put_line(cadena);
    END LOOP;
END;
/
```

```
CREATE OR REPLACE FUNCTION factorial(m number)
  RETURN number
IS
  r number default 1;
  n number := m;
BEGIN
   while (n > 0) loop
      r := r * n;
      n := n - 1;
   end loop;
   return r;
END;
/
```

```
DECLARE
  CURSOR cursor_employees IS
    SELECT * FROM employees
      where department_id = 60;
  v_contador number := 0;
BEGIN
  dbms_output.put_line('No. LastName');
  FOR r_emp in cursor_employees LOOP
      v_contador := v_contador +1 ;
      dbms_output.put_line(v_contador ||'  '||
                          r_emp.last_name);
  END LOOP;
END;
```

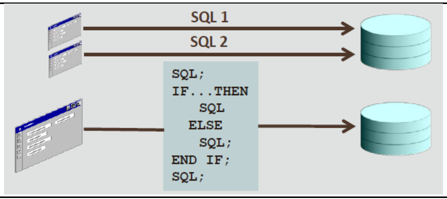
```
DECLARE
CURSOR cursor_employees IS SELECT * FROM employees;
BEGIN
    FOR c_emp in cursor_employees LOOP
        IF c_emp.job_id= 'SA_REP' AND c_emp.hire_date<='05-Feb-2005' THEN
            UPDATE employees SET job_id= 'SR_SA_REP'
            WHERE employee_id= c_emp.employee_id;
        ELSIF c_emp.job_id= 'MK_REP' AND c_emp.hire_date<= '05-Feb-2005' THEN
            UPDATE employees SET job_id= 'SR_MK_REP'
            WHERE employee_id= c_emp.employee_id;
        ELSIF c_emp.job_id = 'ST_CLERK' AND c_emp.hire_date<='05-Feb-2005' THEN
            UPDATE employees SET job_id= 'SR_ST_CLRK'
            WHERE employee_id= c_emp.employee_id;
        END IF;
    END LOOP;
END;
```

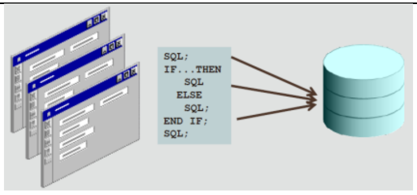
1-2 Benefits of PL/SQL

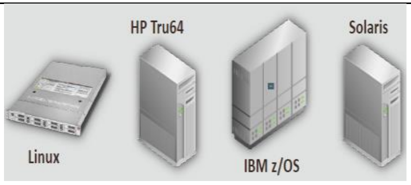
Portability	The ability for PL/SQL programs to run anywhere an Oracle server runs.
Blocks	The basic unit of PL/SQL programs - also known as modules.
Exceptions	An error that occurs in the database or in a user's program during runtime.

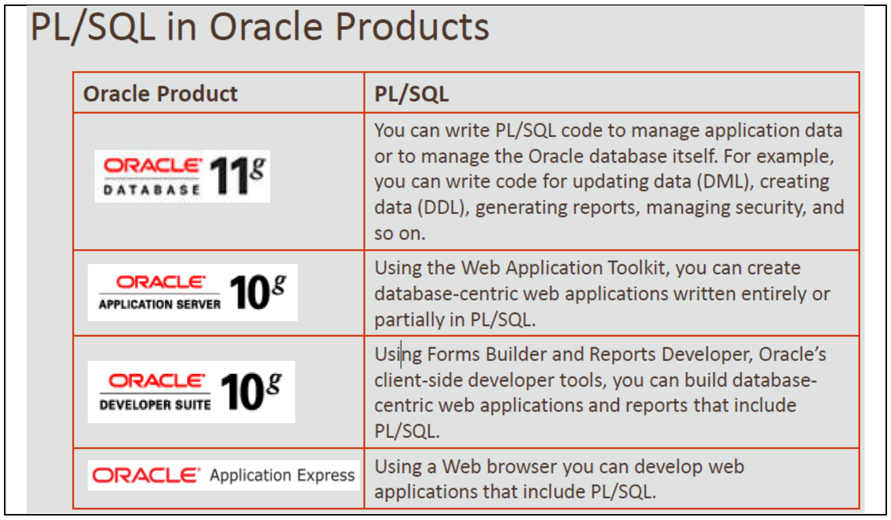
You can nest blocks inside other blocks to build powerful programs.
PL/SQL is integrated in Oracle tools, such as Oracle Forms Developer, Oracle Report Builder, and Application Express.
You can write portable program packages and create libraries that can be reused in different environments.
Exception Handling:

- If no data is found then…
- If too many rows are found then…
- If an invalid number is calculated then…









PL/SQL in Oracle Products

Oracle Product	PL/SQL
ORACLE DATABASE 11g	You can write PL/SQL code to manage application data or to manage the Oracle database itself. For example, you can write code for updating data (DML), creating data (DDL), generating reports, managing security, and so on.
ORACLE APPLICATION SERVER 10g	Using the Web Application Toolkit, you can create database-centric web applications written entirely or partially in PL/SQL.
ORACLE DEVELOPER SUITE 10g	Using Forms Builder and Reports Developer, Oracle's client-side developer tools, you can build database-centric web applications and reports that include PL/SQL.
ORACLE Application Express	Using a Web browser you can develop web applications that include PL/SQL.

1-3 Creating PL/SQL Blocks

Anonymous PL/SQL block	Unnamed blocks of code not stored in the database and do not exist after they are executed
Function	A program that computes and returns a single value
Subprograms	Named PL/SQL blocks that are stored in the database and can be declared as procedures or functions
Compiler	Software that checks and translates programs written in high-level programming languages into binary code to execute
Procedure	A program that performs an action, but does not have to return a value

Application Express	Browser-based, database-driven, application development environment.
SQL Workshop	A component of Application Express.
Application Builder	A component of Application Express.
SQL Developer	An IDE for database development and management.
JDeveloper	An IDE for Java-based development.
NetBeans	An IDE for Java, HTML5, PHP, and C++.

Anonymous Blocks	Procedure: Performs an action	Function: Computes and returns a value
[DECLARE] --variable declarations BEGIN -- statements [EXCEPTION] END;	PROCEDURE name IS --variable declarations BEGIN --statements [EXCEPTION] END;	FUNCTION name RETURN datatype IS --variable declaration(s) BEGIN --statements RETURN value; [EXCEPTION] END;

Section	**Description**	**Inclusion**

Declarative (DECLARE)	Contains declarations of all variables, constants, cursors, and user-defined exceptions that are referenced in the executable and exception sections.	Optional
Executable (BEGIN … END;)	Contains SQL statements to retrieve data from the database and PL/SQL statements to manipulate data in the block. Must contain at least one statement.	Mandatory
Exception (EXCEPTION)	Specifies the actions to perform when errors and abnormal conditions arise in the executable section.	Optional

EXCEPTIONS

```
set SERVEROUTPUT ON;
DECLARE
    v_first_name VARCHAR2(25);
    v_last_name  VARCHAR2(25);
BEGIN
    SELECT first_name, last_name
    INTO v_first_name, v_last_name
    FROM employees
    WHERE last_name= 'King';
    DBMS_OUTPUT.PUT_LINE ('The employee of the month is: '  ||
                          v_first_name|| ' ' || v_last_name|| '.');
EXCEPTION
    WHEN TOO_MANY_ROWS THEN
        DBMS_OUTPUT.PUT_LINE ('Your select statement retrieved multiple rows.
                            Consider using a cursor or changingthe search criteria.');
END;
/
```

```
CREATE OR REPLACE PROCEDURE print_date
IS
v_date VARCHAR2(30);
BEGIN
    SELECT TO_CHAR(SYSDATE,'Mon DD, YYYY') INTO v_date FROM DUAL;
    DBMS_OUTPUT.PUT_LINE(v_date);
END;
```

```
begin
    PRINT_DATE;
end;
/

call    print_date();
execute print_date();
```

```
CREATE OR REPLACE FUNCTION tomorrow(p_today IN DATE)
    RETURN DATE
IS
    v_tomorrow DATE;
BEGIN
    SELECT p_today+1 INTO v_tomorrow FROM DUAL;
    RETURN v_tomorrow;
END;
```

```
BEGIN
DBMS_OUTPUT.PUT_LINE(TOMORROW(SYSDATE));
END;
/

SELECT TOMORROW(SYSDATE) FROM DUAL;
```

```
CREATE OR REPLACE FUNCTION factorial(m number)
  RETURN number
IS
  r number default 1;
  n number := m;
BEGIN
   while (n > 0) loop
      r := r * n;
      n := n - 1;
   end loop;
   return r;
END;
/
```

```
set SERVEROUTPUT ON
BEGIN
DBMS_OUTPUT.PUT_LINE(factorial(5));
END;
/


select factorial(5) from dual;
```

Section 2 – Defining Variables and Datatypes

2-1 Using Variables in PL/SQL

Variables	Used for storage of data and manipulation of stored values.
Parameters	Values passed to a program by a user or by another program to customize the program.

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

```
set SERVEROUTPUT ON
DECLARE
/* Declaracion de
   Variables */
    v_counter INTEGER := 0;
    v_contador number(3) DEFAULT 0;
    v_name VARCHAR2(20) := 'John';
    v_date Date default SYSDATE;
    c_pi constant number(5,4) := 3.1416;
    v_activo BOOLEAN := True;
BEGIN
    v_counter:= v_counter + 1;
    -- SELECT SYSDATE INTO v_date FROM DUAL;
    DBMS_OUTPUT.PUT_LINE(v_counter || ' ' || v_name || ' ' || v_date);
END;
/
```

2-2 Recognizing PL/SQL Lexical Units

Literals	An explicit numeric, character string, date, or Boolean value that is not represented by an identifier. 'UPA' != 'Upa'
Delimiters	Symbols that have special meaning to an Oracle database.
Reserved words	Words that have special meaning to an Oracle database and cannot be used as identifiers.
Comments	Describe the purpose and use of each code segment and are ignored by PL/SQL.
Identifiers	A name, up to 30 characters in length, given to a PL/SQL object. **Not sensitive** May include $ (dollar sign), _ (underscore), or # (hashtag) . v**C**ounter$ = vcounter$
Lexical Units	Building blocks of any PL/SQL block and are sequences of characters including letters, digits, tabs, returns, and symbols.

Partial List of Reserved Words

ALL	CREATE	FROM	MODIFY	SELECT
ALTER	DATE	GROUP	NOT	SYNONYM
AND	DEFAULT	HAVING	NULL	SYSDATE
ANY	DELETE	IN	NUMBER	TABLE
AS	DESC	INDEX	OR	THEN
ASC	DISTINCT	INSERT	ORDER	UPDATE
BETWEEN	DROP	INTEGER	RENAME	VALUES
CHAR	ELSE	INTO	ROW	VARCHAR2
COLUMN	EXISTS	IS	ROWID	VIEW
COMMENT	FOR	LIKE	ROWNUM	WHERE

Delimiters

Symbol	Meaning	Symbol	Meaning
+	addition operator	<>	inequality operator
-	subtraction/negation operator	!=	inequality operator
*	multiplication operator	||	concatenation operator
/	division operator	--	single-line comment indicator
=	equality operator	/*	beginning comment delimiter
'	character string delimiter	*/	ending comment delimiter
;	statement terminator	**	exponent
		:=	assignment operator

2-3 Recognizing Data Types

Object	A schema object with a name, attributes, and methods.
Scalar	Hold a single value with no internal components.
Composite	Contain internal elements that are either scalar (record) or composite (record and table)
Reference	Hold values, called pointers, that point to a storage location.
LOB	Hold values, called locators, that specify the location of large objects (such as graphic images) that are stored out of line. (text, images, video, audio) up to 4GB
BFILE	Store large **binary files** outside of the database.
BLOB	Store large unstructured or structured **binary objects**.
CLOB	Store large blocks of **character** data in the database.
NCLOB	Store large blocks of single-byte or fixed width multi-byte NCHAR data in the database. National language character large object (NCLOB)

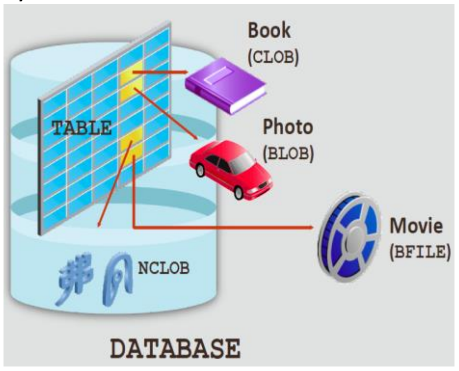
PL/SQL supports five categories of data type

Data Type	Description
Scalar	Holds a single value with no internal elements. Character (char [1..32767], varchar2, long(2GB) Number(number, pls_integer) Date(date, Timestamp) Boolean (True, False, Null)
Composite	Contains multiple internal elements that can be manipulated individually. Record(One row), Table, Varray RECORD v_emp_record employees%ROWTYPE; v_emp_record.first_name
Large Object (LOB)	Holds values called locators that specify the location of large objects (such as graphic images) that are stored out of line.
Reference	Holds values called pointers that point to a storage location.
Object	It is a schema object with a name, attributes, and methods. An object data type is similar to the class mechanism supported by C++ and Java.

LOB Data Type

CLOB, BLOB, and NCLOB data is stored in the database, either inside or outside of the row.

BFILE data is stored in operating system files outside the database.



2-4 Using Scalar Data Types

BOOLEAN	A datatype that stores one of the three possible values used for logical calculations: TRUE, FALSE, or NULL.
%TYPE	Attribute used to declare a variable according to another previously declared variable or database column. PL/SQL determines the data type and size of the variable.

***Identifier table_name.column_name*%TYPE;**
***Identifier identifier*%TYPE;**

```
set SERVEROUTPUT ON
DECLARE
    v_valid       BOOLEAN := True;
    v_id          employees.employee_id%TYPE default 100;
    v_last_name   VARCHAR2(25);
    v_salary      employees.salary%TYPE;
    v_new_salary  v_salary%TYPE;
BEGIN
    select last_name, salary into v_last_name, v_salary
    from employees where employee_id = v_id;
    IF v_valid THEN
        DBMS_OUTPUT.PUT_LINE(v_Last_name || ' ' || (v_salary+1));
    ELSE
        DBMS_OUTPUT.PUT_LINE('Test is FALSE');
    END IF;
END;
/
```

2-5 Writing PL/SQL Executable Statements

Explicit conversion	Converts values from one data type to another by using built-in functions.
Implicit conversion	Converts data types dynamically if they are mixed in a statement.

Character Functions:			Number Functions:			Date Functions:	
ASCII	LENGTH	RPAD	ABS	EXP	ROUND	ADD_MONTHS	MONTHS_BETWEEN
CHR	LOWER	RTRIM	ACOS	LN	SIGN	CURRENT_DATE	ROUND
CONCAT	LPAD	SUBSTR	ASIN	LOG	SIN	CURRENT_TIMESTAMP	SYSDATE
INITCAP	LTRIM	TRIM	ATAN	MOD	TAN	LAST_DAY	TRUNC
INSTR	REPLACE	UPPER	COS	POWER	TRUNC		

Implicit Conversions (It's not recommended)	Explicit Conversions	
	TO_NUMBER()	ROWIDTONCHAR()
	TO_CHAR()	HEXTORAW()
	TO_CLOB()	RAWTOHEX()
	CHARTOROWID()	RAWTONHEX()
	ROWIDTOCHAR()	TO_DATE()

Operator	Operation
**	Exponentiation
+, -	Identity, negation
*, /	Multiplication, division
+, -, ||	Addition, subtraction, concatenation
=, <, >, <=, >=, <>, !=, IS NULL, LIKE, BETWEEN, IN	Comparison
NOT	Logical negation
AND	Conjunction
OR	Inclusion

Statements can continue over several lines:
v_quote := 'The only thing that we can know is that we know
nothing and that is the highest flight of human reason.';
 	
Numbers can be simple values or scientific notation: v_salary number := 2E4;
v_good_sal := v_sal BETWEEN 5000 AND 15000;
	DECLARE
	x VARCHAR2(20);
	BEGIN
	x := '123' + '456' ;
	DBMS_OUTPUT.PUT_LINE(x);
	END;

2-6 Nested Blocks and Variable Scope

Block label	A name given to a block of code which allows access to the variables that have scope, but are not visible.
Variable scope	Consists of all the blocks in which the variable is either local (the declaring block) or global (nested blocks within the declaring block) .
Variable visibility	The portion of the program where the variable can be accessed without using a qualifier.

<<outer>>
optional Label with
any name

```
DECLARE
  v_outer_var VARCHAR2(20):='GLOBAL';
BEGIN
  DECLARE
    v_inner_var VARCHAR2(20):='LOCAL';
  BEGIN
    DBMS_OUTPUT.PUT_LINE(v_inner_var);
    DBMS_OUTPUT.PUT_LINE(v_outer_var);
  END;
  DBMS_OUTPUT.PUT_LINE(v_outer_var);
END;
```

```
<<outer>>
DECLARE
  v_father_name VARCHAR2(20):='Patrick';
  v_date_of_birth DATE:='20-Apr-1972';
BEGIN
  DECLARE
    v_child_name VARCHAR2(20):='Mike';
    v_date_of_birth DATE:='12-Dec-2002';
  BEGIN
    DBMS_OUTPUT.PUT_LINE('Father''s Name: '||v_father_name);
    DBMS_OUTPUT.PUT_LINE('Date of Birth: ' ||outer.v_date_of_birth);
    DBMS_OUTPUT.PUT_LINE('Child''s Name: ' ||v_child_name);
    DBMS_OUTPUT.PUT_LINE('Date of Birth: ' ||v_date_of_birth);
  END;
  DBMS_OUTPUT.PUT_LINE('');
  DBMS_OUTPUT.PUT_LINE('Date of Birth: ' || v_date_of_birth);
END;
```

2-7 Good Programming Practices

Category	**Case Convention**	**Examples**
SQL keywords	Uppercase	SELECT, INSERT
PL/SQL keywords	Uppercase	DECLARE, BEGIN, IF
Data types	Uppercase	VARCHAR2, BOOLEAN
Identifiers (variables, etc.)	Lowercase	v_salary, emp_cursor, c_tax_rate, p_empno
Tables and columns	Lowercase	employees, dept_id, salary, hire_date

Variables starting with v_ Constants starting with c_ Parameters starting with p_
indent each level of code
use %TYPE

Section 3 – Using SQL in PL/SQL

3-1 Review of SQL DML

DELETE	Statement used to remove existing rows in a table.
INSERT	Statement used to add new rows to a table.
MERGE	Statement used to INSERT and/or UPDATE a target table, based on matching values in a source table. UPSERT
UPDATE	Statement used to modify existing rows in a table.
DDL	When you create, change, or delete an object in a database.
DML	When you change data in an object (for example, by inserting or deleting rows).

- Insert Explicit

```
INSERT INTO  employees (employee_id, first_name, last_name, email, hire_date, job_id,  salary)
                         VALUES (305, 'Kareem', 'Naser', 'naserk@oracle.com', SYSDATE, 'SA_REP', NULL);
```

- Insert Implicit

```
INSERT INTO employees  VALUES (
  305, 'Kareem', 'Naser', 'naserk@oracle.com', '111-222-3333', SYSDATE, 'SA_REP', 7000, NULL, NULL, NULL,NULL);
```

DELETE FROM employees WHERE department_id= 80;	If the WHERE clause is omitted, ALL rows will be deleted

UPDATE employees
  SET salary = 11000, commission_pct= .3
  WHERE employee_id= 176;
	If the WHERE clause is omitted, ALL rows will be modified.

1.-
CREATE TABLE bonuses (
employee_id NUMBER(6,0) NOT NULL,
bonus NUMBER(8,2) DEFAULT 0);
	2.-
	INSERT INTO bonuses(employee_id)
	  SELECT employee_id FROM employees
	  WHERE salary < 10000;

3.-
MERGE INTO bonuses b
  USING employees e
    ON (b.employee_id= e.employee_id)
  WHEN MATCHED THEN
    UPDATE SET b.bonus= e.salary * .05;
  WHEN not MATCHED THEN
    INSERT VALUES(e.employee_id, e.bonus);	

3-2 Retrieving Data in PL/SQL

You cannot use DDL and DCL directly in PL/SQL, except to use Dynamic SQL “Execute Immediate” statement.

Handle Style	**Description**
DDL	CREATE TABLE, ALTER TABLE, DROP TABLE
DCL	GRANT, REVOKE

```
SELECT select_list
INTO {variable_name [, variable_name]...
     | record_name}
FROM table
[WHERE condition];
```

```
set SERVEROUTPUT ON
DECLARE
  v_id  employees.employee_id%TYPE:= 100;
  r_emp employees%ROWTYPE;
BEGIN
  SELECT * INTO r_emp
  FROM employees
  WHERE employee_id = v_id;                -- whithout where
  if SQL%FOUND THEN
     DBMS_OUTPUT.PUT_LINE(r_emp.last_name||' '||r_emp.salary);
  End if;
END;
```

3-3 Manipulating Data in PL/SQL

Implicit cursors	Defined automatically by Oracle for all SQL data manipulation statements, and for queries that return only one row. An implicit cursor is always automatically named “SQL”
Explicit cursors	Defined by the programmer for queries that return more than one row.
MERGE	Statement selects rows from one table to update and/or insert into another table. The decision whether to update or insert into the target table is based on a condition in the ON clause.
INSERT	Statement adds new rows to the table.
DELETE	Statement removes rows from the table.
UPDATE	Statement modifies existing rows in the table.

Cursor Attributes for Implicit Cursors

Attribute	**Description**
SQL%FOUND	Boolean attribute that evaluates to TRUE if the most recent SQL statement returned at least one row.
SQL%NOTFOUND	Boolean attribute that evaluates to TRUE if the most recent SQL statement did not return even one row.
SQL%ROWCOUNT	An integer value that represents the number of rows affected by the most recent SQL statement.

```
set SERVEROUTPUT ON
DECLARE
   v_sal_increase employees.salary%TYPE:= 800;
BEGIN
   UPDATE copy_emp
   SET salary = salary + v_sal_increase
   WHERE job_id =  'ST_CLERK';
   DBMS_OUTPUT.PUT_LINE(SQL%ROWCOUNT || ' rows updated.');
END;
```

3-4 Using Transaction Control Statements

Transaction	An inseparable list of database operations, which must be executed either in its entirety or not at all.
ROLLBACK	Used for discarding any changes that were made to the database after the last COMMIT.
SAVEPOINT	Used to mark an intermediate point in transaction processing.
COMMIT	Statement used to make database changes permanent.
END	Keyword used to signal the end of a PL/SQL block, not the end of a transaction.

```
BEGIN
   INSERT INTO pairtable VALUES (1, 2);
   COMMIT;
END;
```

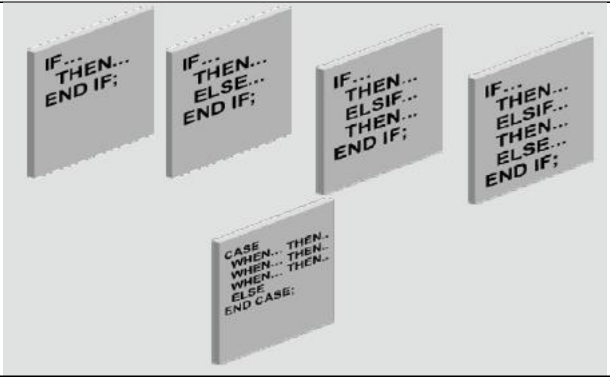
```
BEGIN
  INSERT INTO pairtable VALUES (7, 8);
  SAVEPOINT my_sp_1;
  INSERT INTO pairtable VALUES (9, 10);
  SAVEPOINT my_sp_2;
  INSERT INTO pairtable VALUES (11, 12);
  ROLLBACK to my_sp_1;
  INSERT INTO pairtable VALUES (13, 14);
  COMMIT;
END;
```

Section 4 – Program Structures to Control Execution Flow

4-1 Conditional Control: IF Statements

IF	Statement that enables PL/SQL to perform actions selectively based on conditions.
LOOP	Control structures – Repetition statements that enable you to execute statements in a PL/SQL block repeatedly.
Condition	An expression with a TRUE or FALSE value that is used to make a decision.
CASE	An expression that determines a course of action based on conditions and can be used outside a PL/SQL block in a SQL statement.

IF statement and CASE expressions	Three types of loop control structures

 IF... THEN... END IF;
IF... THEN... ELSE... END IF;
IF... THEN... ELSIF... THEN... END IF;
IF... THEN... ELSIF... THEN... ELSE... END IF;
CASE WHEN... THEN.. WHEN... THEN.. WHEN... THEN.. ELSE END CASE;
	LOOP
	FOR
	WHILE

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

```
set SERVEROUTPUT ON
DECLARE
    v_myage NUMBER := 10;
BEGIN
    IF v_myage > 0 AND v_myage< 11 THEN
        DBMS_OUTPUT.PUT_LINE('I am a child');
    ELSIF v_myage< 20 THEN
        DBMS_OUTPUT.PUT_LINE('I am young');
    ELSIF v_myage< 30 THEN
        DBMS_OUTPUT.PUT_LINE('I am in my twenties');
    ELSIF v_myage< 40 THEN
        DBMS_OUTPUT.PUT_LINE('I am in my thirties');
    ELSE
        DBMS_OUTPUT.PUT_LINE('I am mature');
    END IF;
END;
```

4-2 Conditional Control: Case Statements

Logic Tables	Shows the results of all possible combinations of two conditions.
CASE statement	A block of code that performs actions based on conditional tests.
CASE expression	An expression that selects a result and returns it into a variable.

Logic Tables

AND	TRUE	FALSE	NULL
TRUE	TRUE	Ex. FALSE	NULL
FALSE	FALSE	FALSE	FALSE
NULL	NULL	FALSE	NULL

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

NOT	
TRUE	FALSE
FALSE	TRUE
NULL	NULL

CASE Statements

```
DECLARE
  v_age NUMBER := 10;
  v_txt varchar2(50);
BEGIN
  CASE v_age
    WHEN  0 THEN v_txt := 'unborn';
    WHEN 10 THEN v_txt := 'teenager';
    ELSE v_txt := 'I do not know';
  END CASE;
  DBMS_OUTPUT.PUT_LINE(V_TXT);
END;
```

```
DECLARE
    v_age NUMBER := 10;
    v_txt varchar2(50);
BEGIN
    CASE
        WHEN v_age < 11 THEN v_txt := 'child';
        WHEN v_age < 20 THEN v_txt := 'young';
        WHEN v_age < 30 THEN v_txt := 'twenties';
        ELSE v_txt := 'I am mature';
    END CASE;
    DBMS_OUTPUT.PUT_LINE(V_TXT);
END;
```

CASE Expression Syntax

```
variable_name:=
   CASE selector
      WHEN expression1 THEN result1
      WHEN expression2 THEN result2
      ...
      WHEN expressionN THEN resultN
      [ELSE resultN+1]
   END;
```

```
variable_name:= CASE
   WHEN search_condition1 THEN result1
   WHEN search_condition2 THEN result2
   ...
   WHEN search_conditionN THEN resultN
   [ELSE resultN+1]
END;
```

```
DECLARE
 v_grade CHAR(1) := 'A';
 v_appraisal VARCHAR2(20);
BEGIN
 v_appraisal:=
   CASE v_grade
    WHEN 'A' THEN 'Excellent'
    WHEN 'B' THEN 'Very Good'
    ELSE 'No such grade'
   END;
 DBMS_OUTPUT.PUT_LINE(v_appraisal);
END;
```

```
DECLARE
 v_grade CHAR(1) := 'A';
 v_appraisal VARCHAR2(20);
BEGIN
 v_appraisal :=
   CASE
    WHEN v_grade = 'A' THEN 'Excellent'
    WHEN v_grade IN ('B','C') THEN 'Good'
    ELSE 'No such grade'
   END;
 DBMS_OUTPUT.PUT_LINE (v_appraisal);
END;
```

4-3 Iterative Control: Basic Loops

Basic Loop	Encloses a sequence of statements between the keywords LOOP and END LOOP and must execute at least once.
EXIT	Statement to terminate a loop.

Without the EXIT statement, the loop would never end (an infinite loop)

```
BEGIN
  LOOP
    statements;
    EXIT [WHEN condition];
  END LOOP;
END;
```

```
DECLARE
  v_counter NUMBER(2) := 1;
BEGIN
  LOOP
    DBMS_OUTPUT.PUT_LINE(v_counter);
    v_counter := v_counter + 1;
    EXIT WHEN v_counter > 5;
  END LOOP;
END;
```

```
DECLARE
  v_counter NUMBER := 1;
BEGIN
  LOOP
    DBMS_OUTPUT.PUT_LINE(v_counter);
    v_counter := v_counter + 1;
    IF v_counter > 5 THEN EXIT;
    END IF;
  END LOOP;
END;
```

4-4 Iterative Control: While and For Loops

WHILE Loop	Repeats a sequence of statements until the controlling condition is no longer TRUE.
FOR Loop	Repeats a sequence of statements until a set number of iterations have been completed.

```
WHILE condition LOOP
  statement1;
  statement2;
  . . .
END LOOP;
```

```
FOR counter IN [REVERSE] lower..upper LOOP
  statement1;
  statement2;
. . .
END LOOP;
```

```
DECLARE
 v_counter NUMBER(2) := 1;
BEGIN
 WHILE v_counter < 5 LOOP
  DBMS_OUTPUT.PUT_LINE(v_counter);
  v_counter := v_counter + 1;
 END LOOP;
END;
```

```
DECLARE
 v_limit NUMBER(2) := 5;
BEGIN
 FOR i IN 1..v_limit LOOP
  DBMS_OUTPUT.PUT_LINE(i);
 END LOOP;
END;
```

4-5 Iterative Control: Nested Loops

```
Declare
  r varchar(50);
BEGIN
  FOR i IN 1..3 LOOP
    FOR j IN REVERSE 1..5 LOOP
      r := i || ' X ' || j || ' = ' || i*j;
      DBMS_OUTPUT.PUT_LINE(r);
    END LOOP;
    DBMS_OUTPUT.PUT_LINE('');
  END LOOP;
END;
```

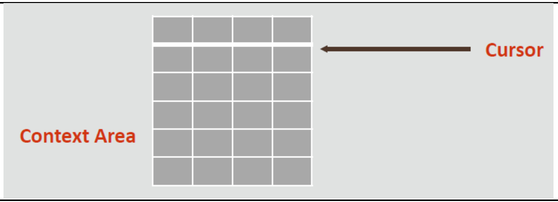
```
DECLARE
 i PLS_INTEGER:= 0;
 j PLS_INTEGER:= 5;
 v_r varchar2(50);
BEGIN
  <<outer_loop>>    -- Label
  LOOP
    i := i + 1;
    j := 5;
    EXIT WHEN i> 3;
    <<inner_loop>>  -- Label
    LOOP
      v_r := i ||' X ' || j || ' = ' || i*j;
      DBMS_OUTPUT.PUT_LINE(v_r);
      j := j-1;
      EXIT WHEN j = 0;
    END LOOP inner_loop;
    DBMS_OUTPUT.PUT_LINE('');
  END LOOP -- outer_loop;
END;
```

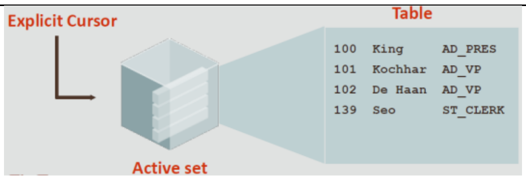
S e m e s t e r 1 , M I D T E R M

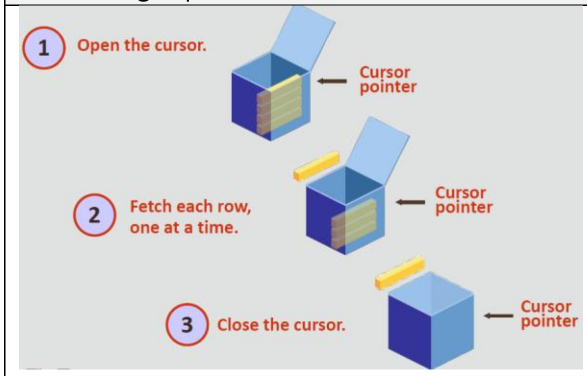
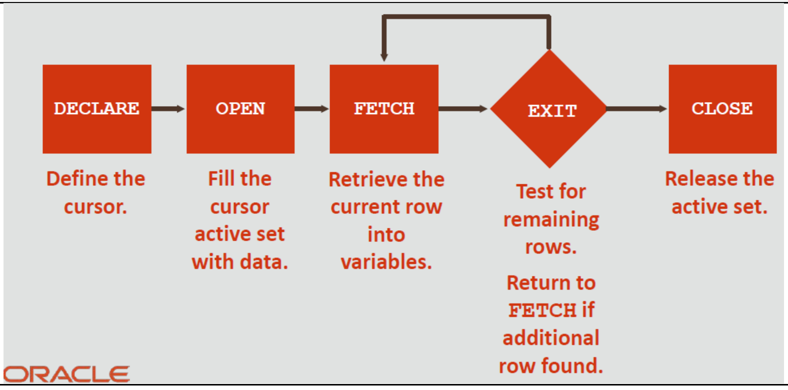
Section 5 – Using Cursors and Parameters

5-1 Introduction to Explicit Cursors

Implicit Cursor	Defined automatically by Oracle for all SQL DML statements, and for SELECT statements that return only one row
Explicit Cursor	Declared by the programmer for queries that return more than one row
Cursor	A label for a context area or a pointer to the context area
Context Area	An allocated memory area used to store the data processed by a SQL statement
Active set	The set of rows returned by a multiple row query in an explicit cursor operation
OPEN	Statement that executes the query associated with the cursor, identifies the active set, and positions the cursor pointer to the first row
FETCH	Statement that retrieves the current row and advances the cursor to the next row either until there are no more rows or until a specified condition is met
CLOSE	Disables a cursor, releases the context area, and undefines the active set





Controlling Explicit Cursors	Steps for Using Explicit Cursors
 1 Open the cursor. Cursor pointer 2 Fetch each row, one at a time. Cursor pointer 3 Close the cursor. Cursor pointer	 DECLARE OPEN FETCH EXIT CLOSE Define the cursor. Fill the cursor active set with data. Retrieve the current row into variables. Test for remaining rows. Return to FETCH if additional row found. Release the active set. ORACLE

```
DECLARE
CURSOR cur_depts IS
  SELECT department_id, department_name
  FROM departments;
BEGIN
    FOR c_dep in cur_depts LOOP
      DBMS_OUTPUT.PUT_LINE(c_dep.department_id||' '||c_dep.department_name);
    END LOOP;
END;
```

```
DECLARE
  CURSOR cur_depts IS
    SELECT department_id, department_name  FROM departments;
  v_department_id   departments.department_id%TYPE;
  v_department_name departments.department_name%TYPE;
BEGIN
  OPEN cur_depts;
  LOOP
    FETCH cur_depts INTO v_department_id, v_department_name;       -- 2 variables
    EXIT WHEN cur_depts%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE(v_department_id||' '||v_department_name);
  END LOOP;
  CLOSE cur_depts;
END;
```

```
DECLARE
  CURSOR cur_depts_emps IS
    SELECT department_name, COUNT(*) AS how_many
    FROM departments d, employees e
    WHERE d.department_id = e.department_id
    GROUP BY d.department_name
    HAVING COUNT(*) > 1;
...
```

5-2 Using Explicit Cursor Attributes

Attribute	Type	Description
Record		A composite data type in PL/SQL, consisting of a number of fields each with their own name and data type
%ROWTYPE		Declares a record with the same fields as the cursor on which it is based
%ISOPEN	Boolean	Evaluates to TRUE if the cursor is open.
%NOTFOUND	Boolean	Evaluates to TRUE if the most recent fetch did not return a row.
%FOUND	Boolean	Evaluates to TRUE if the most recent fetch returned a row; opposite of %NOTFOUND.
%ROWCOUNT	Number	Evaluates to the total number of rows FETCHed so far.

```
set SERVEROUTPUT ON
DECLARE
    CURSOR cur_emps IS
        SELECT *FROM employees
        WHERE department_id= 60;
    r_emp cur_emps%ROWTYPE;
BEGIN
    OPEN cur_emps;
    LOOP
        FETCH cur_emps INTO r_emp;
        EXIT WHEN cur_emps%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE(
          r_emp.employee_id|| ' - ' ||
          r_emp.last_name);
    END LOOP;
    CLOSE cur_emps;
END;
```

```
DECLARE
    CURSOR cur_emps_dept IS
        SELECT last_name, department_name
        FROM employees e, departments d
        WHERE e.department_id=d.department_id;
    r_emp_dep cur_emps_dept%ROWTYPE;
BEGIN
    OPEN cur_emps_dept;
    LOOP
        FETCH cur_emps_dept INTO r_emp_dep;
        EXIT WHEN cur_emps_dept%NOTFOUND OR
                  cur_emps_dept%ROWCOUNT> 10;
        DBMS_OUTPUT.PUT_LINE(
            r_emp_dep.last_name||' - '||
            r_emp_dep.department_name);
    END LOOP;
    CLOSE cur_emps_dept;
END;
```

```
IF NOT cur_emps%ISOPEN THEN
   OPEN cur_emps;
END IF;
LOOP
FETCH cur_emps...
```

5-3 Cursor FOR Loops

Cursor FOR loop	Automates standard cursor-handling operations such as OPEN, FETCH, %NOTFOUND, and CLOSE so that they do not need to be coded explicitly

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

   ■  Without declare the cursor
BEGIN
  FOR r_emp IN (SELECT * FROM employees
                WHERE department_id= 50)
  LOOP
    DBMS_OUTPUT.PUT_LINE(r_emp.last_name);
  END LOOP;
END;
```

```
DECLARE
CURSOR cur_dep IS
  SELECT department_id, department_name
    FROM departments
    ORDER BY department_id;
BEGIN
    FOR r_dep IN cur_dep LOOP
      EXIT WHEN cur_dep%ROWCOUNT > 5;
      DBMS_OUTPUT.PUT_LINE(r_dep.department_id ||
                           ' ' || r_dep.department_name);
    END LOOP;
END;
```

5-4 Cursors with Parameters

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

```
OPEN cursor_name(parameter_value1,
                 parameter_value2, ...);

FOR r_emp IN cur_emp(60) LOOP
```

```
DECLARE
  CURSOR cursor_employees( p_dep number)  IS
    SELECT * FROM employees
    where department_id = p_dep;
  v_contador number := 0;
BEGIN
  dbms_output.put_line('No. LastName');
  FOR r_emp in cursor_employees(90) LOOP
      v_contador := v_contador +1 ;
      dbms_output.put_line(v_contador ||'  '||
                           r_emp.last_name);
  END LOOP;
END;
```

```
DECLARE
    CURSOR cur_emp(p_dep integer) IS
        SELECT * FROM employees
        WHERE department_id= p_dep;
    r_emp cur_emp%ROWTYPE;
BEGIN
    OPEN cur_emp(90);
    LOOP
        FETCH cur_emp INTO r_emp;
        EXIT WHEN cur_emp%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE(
          r_emp.employee_id|| ' - ' ||
          r_emp.last_name);
    END LOOP;
    CLOSE cur_emp;
END;
```

```
CREATE OR REPLACE PROCEDURE pr_Emp(p_dep NUMBER)
IS
CURSOR cursor_employees(p_dep number)  IS
  SELECT * FROM employees
  where department_id = p_dep;
v_contador number := 0;
BEGIN
  dbms_output.put_line('No. LastName');
  FOR r_emp in cursor_employees(p_dep) LOOP
      v_contador := v_contador +1 ;
      dbms_output.put_line(v_contador ||'  '||
                          r_emp.last_name);
  END LOOP;
END;
/
call pr_Emp(90);
```

```
CREATE PROCEDURE pr_Emp2(p_dep NUMBER)
IS
CURSOR cursor_employees  IS
   SELECT * FROM employees
   where department_id = p_dep;
v_contador number := 0;
BEGIN
  dbms_output.put_line('No. LastName');
  FOR r_emp in cursor_employees LOOP
      v_contador := v_contador +1 ;
      dbms_output.put_line(v_contador ||
                          '  '|| r_emp.last_name);
  END LOOP;
END;
/
call pr_Emp2(90);      -- little aesthetic.
```

5-5 Using Cursors For Update

FOR UPDATE	Declares that each row is locked as it is being fetched so other users cannot modify the rows while the cursor is open
NOWAIT	A keyword used to tell the Oracle server not to wait if the requested rows have already been locked by another user

```
CURSOR cursor_name IS
SELECT... FROM...
FOR UPDATE [OF column_reference] [NOWAIT | WAIT n];
```

n = number of seconds to wait and check whether the rows are unlocked.

If the cursor is based on a join of two tables, we may want to lock the rows of one table but not the other
To do this, we specify **any column** of the table we want to lock.

It also allows us to modify the rows ourselves using a … WHERE CURRENT OF *cursor-name*

```
DECLARE
  CURSOR cur_eds IS
    SELECT employee_id, salary, department_name
    FROM my_employees e, my_departments d
    WHERE e.department_id = d.department_id
    FOR UPDATE OF salary NOWAIT;
BEGIN
  FOR r_eds IN cur_eds LOOP
    UPDATE my_employees
      SET salary = r_eds.salary * 1.1
      WHERE CURRENT OF cur_eds;
  END LOOP;
END;
```

5-6 Using Multiple Cursors

Explain the need for using multiple cursors to produce multi-level reports

```
set SERVEROUTPUT ON
DECLARE
  CURSOR cur_dep IS
     SELECT * FROM departments;
  CURSOR cur_emp (p_dep NUMBER) IS
     SELECT * FROM employees WHERE department_id = p_dep;
BEGIN
  FOR r_dep IN cur_dep LOOP
    DBMS_OUTPUT.PUT_LINE(upper(r_dep.department_name));
    FOR r_emp IN cur_emp (r_dep.department_id) LOOP
      DBMS_OUTPUT.PUT_LINE(r_emp.last_name);
    END LOOP;
    DBMS_OUTPUT.PUT_LINE('');
  END LOOP;
END;
```

Section 6 – Using Composite Datatypes

6-1 User-Defined Records

6-2 Indexing Tables of Records