

**Modul Number: 0750362**

**Module Name: Database Applications**

**Teacher: Eman Alnaji**

**Part IV**

**PL/SQL**

**What is PL/SQL?**

* Procedural programming language
	+ Uses detailed instructions
	+ Processes statements sequentially
* Combines SQL commands with procedural instructions
* Used to perform sequential processing using an Oracle database
* PL/SQL supports variables, conditions, loops and exceptions.
* PL/SQL blocks can include control flow and DML statements.

**When is PL/SQL Useful?**

* When something is too complicated for SQL
* When conditional branching and looping are needed

**Basic Structure:**



**PL/SQL Program Lines:**

* May span multiple text editor lines
* Each line ends with a semicolon
* Text is not case sensitive

**Comment Statements:**

* Block of comments are delimited with /\* \*/

/\* <comment that spans more than one line of code> \*/

* Single comment line starts with 2 hyphens

 -- comment on a single line

**Variables:**

Variables can have:

* any SQL data type, such as CHAR, DATE, or NUMBER
* or any PL/SQL data type, such as BOOLEAN or BINARY\_INTEGER.
* Reference data types:
	+ Reference a database item
	+ Assume data type of item
		- %TYPE: assumes data type of field
		- %ROWTYPE: assumes data type of entire row
* Syntax for declaring a variable:

*variable\_name data\_type\_declaration*;

* Examples:
	+ part\_no NUMBER(4);
	+ in\_stock BOOLEAN;

**Arithmetic Operators:**

O a 

**Assigning Values to Variables:**

**First:** Assignment Statement

* Assignment operator: **:=**
* Variable being assigned to a new value is on left side of assignment operator
* New value is on right side of operator

student\_name := ‘John Miller’;

student\_name := current\_student;

 tax := price \* tax\_rate;

**Second:** Selecting database value into a variable.

SELECT salary \* 0.10
INTO bonus
FROM employee
WHERE SSN = SSN\_In;

**Displaying PL/SQL Output in SQL\*Plus**

* Command to activate memory buffer in SQL\*Plus to enable output from PL/SQL programs:

 **SQL> SET SERVEROUTPUT ON**

* Command to output data from a PL/SQL program in SQL\*Plus:

 **DBMS\_OUTPUT.PUT\_LINE(‘*output string*’);**

 **DBMS\_OUTPUT.PUT\_LINE(‘Employee Salary: ’|| Salary);**

**- - || is a concatenation operator**

**Executing a PL/SQL Program in SQL\*Plus**

* Copy program code from Notepad to SQL\*Plus
* Type / to execute

**Character String Functions in PL/SQL**

* Concatenating strings: joining 2 or more character strings into a single string
* Concatenation operator: ||

**s\_first\_name := ‘Sarah’**

**s\_last\_name := ‘Miller’**

**s\_full\_name := s\_first\_name || ‘ ’ || s\_last\_name**

**PL/SQL Character String Functions**

* These functions were discussed before, but here are some examples using them in PL/SQL
* **RTRIM:** removes blank trailing spaces

 cust\_address := RTRIM(cust\_address);

* **LENGTH:** returns string length (number of characters)

 address\_length := LENGTH(cust\_address);

* **UPPER, LOWER:** changes characters to all upper or lower case

 s\_name := UPPER(s\_name);

 s\_name := LOWER(s\_name);

* **INSTR:**  searches a string and looks for a matching substring and returns its starting position

starting\_position := INSTR(string\_being\_searched, search\_string>);

blank\_position := INSTR(‘Sarah Miller’, ‘ ’);

* **SUBSTR:**  extracts a specific number of characters from a string, starting at a given point

extracted\_string := SUBSTR(*string\_being\_searched*, *starting\_point*, *number\_of\_characters\_to\_extract*);

s\_first\_name := SUBSTR(‘Sarah Miller’, 1,5);

**NULL Values in Assignment Statements**

* Until a value is assigned to a variable, the variable’s value is NULL
* Performing an arithmetic value on a NULL value always results in a NULL value
* Advice: Always initialize variable values

**PL/SQL Selection Structures (IF Statement)**

* **IF/END IF:**

**IF *condition* THEN**

***program statements***

**END IF;**

* **IF/ELSE/END IF:**

**IF *condition* THEN**

***program statements***

**ELSE**

***alternate program statements***

 **END IF;**

* **IF/ELSIF:**

**IF *condition1* THEN**

***program statements;***

**ELSIF *condition2* THEN**

 ***alternate program statements;***

**ELSIF *condition3* THEN**

 ***alternate program statements;***

**. . .**

**ELSE**

 ***alternate program statements;***

**END IF;**

**PL/SQL Comparison Operators**



**Evaluating NULL Conditions in IF/THEN Structures**

* **If a condition evaluates as NULL, then it is FALSE**
* **How can a condition evaluate as NULL?**
	+ **It uses a BOOLEAN variable that has not been initialized**
	+ **It uses any other variable that has not been initialized**

**Example:**

IF acct\_balance >= debit\_amt THEN

 UPDATE accounts SET bal = bal - debit\_amt

 WHERE account\_id = acct;

ELSE

 INSERT INTO temp

 VALUES (acct, acct\_balance, 'Insufficient funds');

END IF;

**PL/SQL Loops**

* Loop: repeats one or more program statements multiple times until an exit condition is reached
	+ Pretest loop: exit condition is tested before program statements are executed
	+ Posttest loop: exit condition is tested after program statements are executed

**LOOP … EXIT Loop**



**LOOP … EXIT WHEN Loop**



**WHILE Loop**



**Numeric FOR Loop**



**Examples:**

FOR num IN 1..500 LOOP

 INSERT INTO roots VALUES (num, SQRT(num));

END LOOP;

WHILE salary <= 2500 LOOP

 SELECT salary, mgr\_ssn, lname

 INTO salary, mgr\_ssn, last\_name

 FROM employee

 WHERE ssn = mgr\_ssn;

 END LOOP;

**Cursors**

A cursor is a pointer to a private SQL area that stores results of a SELECT statement.

**Types of Cursors**

* Implicit
* Explicit

**Implicit Cursors**

* Created automatically every time you use an INSERT, UPDATE, DELETE, or SELECT command
* Doesn’t need to be declared
* Can be used to assign the output of a SELECT command to one or more PL/SQL variables
* Can only be used if query returns one and only one record

**Explicit Cursors**

* Must be declared in program DECLARE section
* Can be used to assign the output of a SELECT command to one or more PL/SQL variables
* Can be used if query returns multiple records or no records

**Using an Explicit Cursor**

* Declare the cursor
* Open the cursor
* Fetch the cursor result into PL/SQL program variables
* Close the cursor

**Declaring an Explicit Cursor**

DECLARE

 CURSOR *cursor\_name* IS *SELECT\_statement*;

**Opening an Explicit Cursor**

OPEN *cursor\_name*;

**Fetching Explicit Cursor Records**

FETCH *cursor\_name*

INTO *variable\_name(s)*;

**Closing an Explicit Cursor**

CLOSE *cursor\_name*;

**Processing an Explicit Cursor**

* LOOP ..EXIT WHEN approach:

OPEN *cursor\_name*;

LOOP

 FETCH *cursor\_name* INTO v*ariable\_name(s)*;

 EXIT WHEN *cursor\_name*%NOTFOUND:

END LOOP;

CLOSE *cursor\_name*;

* Cursor FOR Loop approach:

FOR *variable\_name(s)* in *cursor\_name* LOOP

 *additional processing statements*;

END LOOP;

**Explicit Cursor Attributes**



**Using Reference Data Types in Explicit Cursor Processing**

* Declaring a ROWTYPE reference variable:

DECLARE

 *reference\_variable\_name cursor\_name*%ROWTYPE;

* Referencing a ROWTYPE reference variable:

*reference\_variable\_name.database\_field\_name*

**PL/SQL Example1:**

DECLARE

 Emp\_name VARCHAR2(10);

 Cursor c1 IS SELECT Ename FROM Emp\_tab

 WHERE Deptno = 20;

BEGIN

 OPEN c1;

 LOOP

 FETCH c1 INTO Emp\_name;

 EXIT WHEN c1%NOTFOUND;

 DBMS\_OUTPUT.PUT\_LINE(Emp\_name);

 END LOOP;

END;

**PL/SQL Example2:**

DECLARE

Emp\_number INTEGER := 9999;

Emp\_name emp.empname%type;

BEGIN

 SELECT Ename INTO Emp\_name

 FROM Emp\_tab

 WHERE Empno = Emp\_number; -- no such number

 DBMS\_OUTPUT.PUT\_LINE('Employee name is ' || Emp\_name);

EXCEPTION

 WHEN NO\_DATA\_FOUND THEN

 DBMS\_OUTPUT.PUT\_LINE('No such employee: ' || Emp\_number);

 END;

**PL/SQL Example3:**

**DECLARE**

 **CURSOR** c1 **is**

 **SELECT** fname, ssn, salary **FROM** employee

 **ORDER BY** salary **DESC**; --start w/ highest paid emp

 my\_ename **VARCHAR2(10**);

 my\_empno **CHAR(9**);

 my\_sal **NUMBER(10,2**);

**BEGIN**

 **OPEN** c1;

 **FETCH** c1 **INTO** my\_ename, my\_empno, my\_sal;

 **WHILE** C1%FOUND **LOOP**

 **DBMS\_OUTPUT.PUT\_LINE** (MY\_EMPNO||','||MY\_SAL);

 **UPDATE** employee

 **SET** salary = salary \* 1.1

 **WHERE** ssn = my\_empno;

/\*By this statement you will update only the employees retrieved by the cursor.\*/

 **FETCH** c1 **INTO** my\_ename, my\_empno, my\_sal;

 **END LOOP;**

**CLOSE c1;**

**COMMIT; /\* This will save all updates applied on Employee table \*/**

**EXCEPTION**

 **WHEN OTHERS THEN**

 **ROLLBACK;**

**/\*If any error occurred, any updates applied on the employee table will not be saved and reversed as the state it was before applying this program \*/**

**END;**

**PL/SQL Exception Handling**

* All error handling statements are placed in the EXCEPTION program block
* Exception handler: program command that provides information about an error, and suggest correction actions

**Predefined Exceptions**

* Common errors that have been given predefined names that appear instead of error numbers



**Exception Handler Syntax for Predefined Exceptions**



**Exercise:**

**Apply the following:**

* **Create a new table: Employee2 (**
	+ **Fname, LName, SSN, Salary, Salary2, Dsc**

**Write a PL/SQL block that applies the following:**

* **For each employee (in the original table employee), insert a new record in table employee2, with the same information (fname, lname, ssn, salary), and compute salary2 and dsc, as follows:**
	+ **If the employee worked more than 40 hours on all projects**
		- **Put “Good” in his description and give him a 10% raise**
	+ **Otherwise: Put “Bad” in his description**
* **Find the employee with the highest salary and print his name and SSN**
* **Display the name and salary of employee with SSN “111997788”.**
	+ **If no employee has that SSN, display “Employee not found”**