

**Modul Number: 0750362**

**Module Name: Database Applications**

**Teacher: Eman Alnaji**

**Part V**

**PL/SQL (Cont.)**

**Exceptions (Cont.):**

**Undefined Exceptions:**

* Less-common errors that have not been given predefined names
* ORA- error code appears
* Exception handler tests for ORA- error code and provides alternate error message

**User-Defined Exceptions:**

* Errors that will not cause a run-time error, but will violate business rules

 (i.e. they are created for logical errors)

* Programmer creates a custom error message

**Example of a User-Defined Exception:**

**Declare**

 **huge\_quantity exception; -- Declaration of the exception**

 **V\_qty number(10);**

 **v\_msg varchar2(100);**

**Begin**

 **v\_qty := &V\_qty; -- Here this value is requested from the user**

 **if v\_qty > 1000 then**

 **v\_msg := ‘very huge quantity’;**

 **raise huge\_quantity; -- Raising the exception huge\_quantity**

 **else**

 **v\_msg := ‘Good’;**

 **end if;**

 **dbms\_output.put\_line (v\_msg);**

**Exception**

 **when huge\_quantity then -- Handling of exception huge\_quantity**

 **dbms\_output.put\_line (v\_msg);**

**End;**

**Using an error number for a User-Defined Exception:**

* Oracle provides the numbers from -20000 to -20999 to User-Defined Excpetions.
* In the previous example, you can handle the huge\_quantity exception using an error number.

**Exception**

**when huge\_quantity then**

 **raise\_application\_error (-20100, v\_msg);**

 **End;**

**Nested PL/SQL Program Blocks**

* An inner program block can be nested within an outer program block



**Exception Handling in Nest Program Blocks**

* If an exception is raised and handled in an inner block, program execution resumes in the outer block



* Exceptions raised in inner blocks can be handled by exception handlers in outer blocks

**Advanced PL/SQL Programs**

**Anonymous PL/SQL Programs**

* PL/SQL blocks that we have written so far.
* Write code in text editor, execute it in SQL\*Plus
* Code can be stored as text in file system
* Program cannot be called by other programs, or executed by other users
* Cannot accept or pass parameter values

**Named PL/SQL Programs**

* Can be created:
	+ Using text editor & executed in SQL\*Plus
	+ Using Procedure Builder (an application installed within Oracle Developer Package).
* Can be stored:
	+ As compiled objects in database
	+ As source code libraries in file system
* Can be called by other programs
* Can be executed by other users

**Named Program Locations**

* Server-side
	+ Stored in database as database objects
	+ Execute on the database server
* Client-side
	+ Stored in the client workstation file system
	+ Execute on the client workstation

**Named Program Types**

* Program Units
	+ Procedures
	+ Functions
* Packages
* Triggers

**Program Units**

* Procedures
	+ Can receive and pass multiple parameter values
	+ Can call other program units
* Functions
	+ Like procedures, except they return a single value

**Parameters**

* Variables used to pass data values in/out of program units
* Declared in the procedure/function header
* Parameter values are passed when the procedure/function is called from the calling program

**Parameter Modes**

* IN
	+ Incoming values, read-only (default)
* OUT
	+ Outgoing values, write-only
* IN OUT
	+ Can be both incoming and outgoing

**Procedures - Functions**

**Creating a Procedure**



**Executing a Procedure (in SQLPlus)**



**Calling a Procedure from another Procedure or Function**



**Parameter Types**

* Formal parameters: declared in procedure header
* Actual parameters: values placed in parameter list when procedure is called
* Values correspond based on order



**Dropping a Procedure**



**Creating a Function**



**Function Syntax Details**

* RETURN command in header specifies data type of value the function will return
* RETURN command in body specifies actual value returned by function

**Calling a Function**

* Can be called from either named or anonymous PL/SQL blocks
* Can be called within SQL queries



**Note:** return\_value should be a declared variable.

**Example1:**

**Create a procedure that prints all employees for a given department number.**

CREATE OR REPLACE PROCEDURE Get\_emp\_names (V\_dno IN NUMBER)

IS

 Emp\_name VARCHAR2(30);

 CURSOR c1 IS

 SELECT fname FROM employee

 WHERE dno = v\_dno;

BEGIN

 OPEN c1;

 LOOP

 FETCH c1 INTO Emp\_name;

 EXIT WHEN C1%NOTFOUND;

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

 END LOOP;

 CLOSE c1;

END;

**Execution:**



**Example2:**

Assuming that the salary field in table employee stores the annual salary of an employee, create a function that returns the monthly salary of a given employee.

CREATE OR REPLACE FUNCTION Mon\_Sal (v\_ssn employee.ssn%type)

RETURN NUMBER

IS

 Monthly\_sal NUMBER(10,2);

BEGIN

 SELECT round (salary/12)

 INTO Monthly\_sal

 FROM Employee

 WHERE ssn = v\_ssn;

RETURN (Monthly\_sal);

END;

/

**Execution:**



**Packages**

* Can contain:
	+ Global variable declarations
	+ Cursors
	+ Procedures
	+ Functions

**Package Components**

* Specification
	+ Used to declare all public variables, cursors, procedures, functions
* Body
	+ Contains underlying code for procedures and functions

**Creating a Package Specification in SQL\*Plus**



**Creating a Package Body in SQL\*Plus**



**Calling a Program Unit That Is In a Package**



**Example:**



