**Day 08**

**PL/SQL Control Structures & Procedures, Functions and Packages**

After successful completion of this session you will:

* learn about PL/SQL Control Structures
* learn about Stored Procedure, Function and Package

## 8.1 PL/SQL Control Structures

Control structures are the most important PL/SQL extension to SQL. Not only does PL/SQL let you manipulate Oracle data, it lets you process the data using conditional, iterative, and sequential flow-of-control statements such as IF-THEN-ELSE, CASE, FOR-LOOP, WHILE-LOOP, EXIT-WHEN, and GOTO.

#### 8.1.1 Conditional Control With IF-THEN

#### Often, it is necessary to take alternative actions depending on circumstances. The IF-THEN statement lets you run a sequence of statements conditionally. The forms of the statement can be IF-THEN, IF-THEN-ELSE, or IF-THEN-ELSEIF-ELSE. The IF clause checks a condition; the THEN clause defines what to do if the condition is true; and the ELSE clause defines what to do if the condition is false or null.

[Example 8-1](http://docs.oracle.com/cd/B25329_01/doc/appdev.102/b25108/xedev_plsql.htm" \l "CHDHFGGE) Using a Simple IF-THEN Statement in PL/SQL

DECLARE

sal NUMBER(8,2);

bonus NUMBER(6,2);

hiredate DATE;

partid NUMBER(6) := 6; -- use participant 6 for testing

BEGIN

-- retrieve the salary and the date that participant was hired, the date is

-- checked to calculate the amount of the bonus for the participant

SELECT salary, hire\_date INTO sal, hiredate FROM participant

WHERE par\_id = partid;

IF hiredate > TO\_DATE('01-JAN-10') THEN

bonus := sal/20;

DBMS\_OUTPUT.PUT\_LINE('Bonus for participant: ' || partid || ' is: ' || bonus );

END IF;

END;

/

[Example 8-2](http://docs.oracle.com/cd/B25329_01/doc/appdev.102/b25108/xedev_plsql.htm#CHDHFGGE) shows the use of IF-THEN-ELSEIF-ELSE to determine the salary raise an participant receives based on the hire date of the participant.

[Example 8-2](http://docs.oracle.com/cd/B25329_01/doc/appdev.102/b25108/xedev_plsql.htm" \l "CHDHFGGE) Using the IF-THEN-ELSEIF Statement in PL/SQL

DECLARE

bonus NUMBER(6,2);

partid NUMBER(6) := 4;

hiredate DATE;

BEGIN

-- retrieve the date that participant was hired, the date is checked

-- to determine the amount of the bonus for the participant

SELECT hire\_date INTO hiredate FROM works WHERE par\_id = partid;

IF hiredate > TO\_DATE('01-JAN-03') THEN

bonus := 500;

ELSIF hiredate > TO\_DATE('01-JAN-01') THEN

bonus := 1000;

ELSE

bonus := 1500;

END IF;

DBMS\_OUTPUT.PUT\_LINE('Bonus for participant: ' || partid || ' is: ' || bonus );

END;

/

#### 8.1.2 Conditional Control With the CASE Statement

To choose among several values or courses of action, you can use CASE constructs. The CASE expression **evaluates a condition and returns a value for each case**. The case statement evaluates a condition, and performs an action, such as an entire PL/SQL block, for each case. When possible, rewrite lengthy IF-THEN-ELSIF statements as CASE statements because the CASE statement is more readable and more efficient.

[Example 8-3](http://docs.oracle.com/cd/B25329_01/doc/appdev.102/b25108/xedev_plsql.htm" \l "CHDHFGGE) Using the CASE-WHEN Statement in PL/SQL

DECLARE

grade CHAR(1);

BEGIN

grade := 'B';

CASE grade

WHEN 'A' THEN DBMS\_OUTPUT.PUT\_LINE('Excellent');

WHEN 'B' THEN DBMS\_OUTPUT.PUT\_LINE('Very Good');

WHEN 'C' THEN DBMS\_OUTPUT.PUT\_LINE('Good');

WHEN 'D' THEN DBMS\_OUTPUT.PUT\_LINE('Fair');

WHEN 'F' THEN DBMS\_OUTPUT.PUT\_LINE('Poor');

ELSE DBMS\_OUTPUT.PUT\_LINE('No such grade');

END CASE;

END;

/

[Example 8-4](http://docs.oracle.com/cd/B25329_01/doc/appdev.102/b25108/xedev_plsql.htm#BABCBICG) determines the salary raise a participant receives based on the current salary of the participant and the job ID. This complex example combines the CASE expression with IF-THEN-ELSE statements.

[Example 8-4](http://docs.oracle.com/cd/B25329_01/doc/appdev.102/b25108/xedev_plsql.htm" \l "BABCBICG) Using the IF-THEN\_ELSE and CASE Statement in PL/SQL

DECLARE -- declare variables

partid NUMBER(6) := 5;

deptname VARCHAR2(15);

sal NUMBER(8,2);

sal\_raise NUMBER(3,2); -- this is the rate of increase for the raise

BEGIN

-- retrieve the job ID and salary for the participant and

-- assign the values to variables deptname and sal

SELECT dept\_name, salary INTO deptname, sal from works WHERE par\_id = partid;

CASE -- determine the salary raise rate based on participant department

WHEN deptname = 'IT' THEN

IF sal < 20000 THEN sal\_raise := .08;

ELSE sal\_raise := .07;

END IF;

WHEN deptname = 'Marketing' THEN

IF sal < 18000 THEN sal\_raise := .07;

ELSE sal\_raise := .06;

END IF;

WHEN deptname = 'Finance' THEN

IF sal < 15000 THEN sal\_raise := .06;

ELSE sal\_raise := .05;

END IF;

WHEN deptname = 'Planning' THEN

IF sal < 17000 THEN sal\_raise := .05;

ELSE sal\_raise := .04;

END IF;

ELSE -- if no conditions met, then the following

DBMS\_OUTPUT.PUT\_LINE('No raise for this department ' || deptname);

END CASE;

-- display the percent raise for the participant

DBMS\_OUTPUT.PUT\_LINE('Percent salary raise for participant: ' || partid || ' is: ' || sal\_raise );

END;

/

#### 8.1.3 Iterative Control With LOOPs

LOOP statements let you run a sequence of statements multiple times. Place the keyword LOOP before the first statement and the keywords END LOOP after the last statement in the sequence. The FOR-LOOP statement lets you specify a range of integers, then run a sequence of statements once for each integer in the range. In [Example 8-5](http://docs.oracle.com/cd/B25329_01/doc/appdev.102/b25108/xedev_plsql.htm#BABGFHBG) the loop displays the number and the square of the number for numbers 1 to 10.Note that you **do not** have to declare or initialize the counter in the FOR-LOOP and any valid identifier can be used for the name, such as loop\_counter.

[Example 8-5](http://docs.oracle.com/cd/B25329_01/doc/appdev.102/b25108/xedev_plsql.htm" \l "BABGFHBG) Using the FOR-LOOP in PL/SQL

BEGIN

FOR loop\_counter IN 1..10 LOOP

DBMS\_OUTPUT.PUT\_LINE('Number: ' || TO\_CHAR(loop\_counter)

|| ' Square: ' || TO\_CHAR(loop\_counter\*\*2));

END LOOP;

END;

/

The WHILE-LOOP statement associates a condition with a sequence of statements. Before each iteration of the loop, the condition is evaluated. If the condition is true, the sequence of statements is executed, then control resumes at the top of the loop. If the condition is false or null, the loop is bypassed and control passes to the next statement. In [Example 8-6](http://docs.oracle.com/cd/B25329_01/doc/appdev.102/b25108/xedev_plsql.htm#BABFAHFH), the loop displays the number and the cube of the number while the number is less than or equal to 10.

[Example 8-6](http://docs.oracle.com/cd/B25329_01/doc/appdev.102/b25108/xedev_plsql.htm" \l "BABFAHFH) Using WHILE-LOOP for Control in PL/SQL

DECLARE -- declare variables

i NUMBER := 1; -- loop counter, initialize to one

i\_cubed NUMBER;

BEGIN

-- use WHILE LOOP to process data

WHILE i <= 10 LOOP

i\_cubed := i\*\*3;

DBMS\_OUTPUT.PUT\_LINE('Number: ' || TO\_CHAR(i)

|| ' Cube: ' || TO\_CHAR(i\_cubed));

i := i + 1;

END LOOP;

END;

/

The EXIT-WHEN statement lets you complete a loop if further processing is impossible or undesirable. When the EXIT statement is encountered, the condition in the WHEN clause is evaluated. If the condition is true, the loop completes and control passes to the next statement. In [Example 8-7](http://docs.oracle.com/cd/B25329_01/doc/appdev.102/b25108/xedev_plsql.htm#BABBCHGE), the loop completes when the value of total exceeds 25,000:

[Example 8-7](http://docs.oracle.com/cd/B25329_01/doc/appdev.102/b25108/xedev_plsql.htm" \l "BABBCHGE) Using the EXIT-WHEN Statement in PL/SQL

DECLARE -- declare and assign values to variables

total NUMBER(9) := 0;

counter NUMBER(6) := 0;

BEGIN

LOOP

counter := counter + 1; -- increment counter variable

total := total + counter \* counter; -- compute total

-- exit loop when condition is true

EXIT WHEN total > 25000; -- LOOP until condition is met

END LOOP;

DBMS\_OUTPUT.PUT\_LINE('Counter: ' || TO\_CHAR(counter)

|| ' Total: ' || TO\_CHAR(total)); -- display results

END;

/

### 8.2 Local PL/SQL Procedures and Functions in PL/SQL Blocks

Procedures and functions (subprograms) are **named** PL/SQL blocks that can be called with a set of parameters from inside of a PL/SQL block. A **procedure** is a subprogram that performs a specific action. You specify the name of the procedure, its parameters, its local variables, and the BEGIN-END block that contains its code and handles any exceptions. A **function** is a subprogram that computes and returns a value. Functions and procedures are **structured alike, except that functions return a value**.

When passing parameters to functions and procedures, the parameters can be declared as IN or OUT or IN OUT parameters.

* IN indicates that you must supply a value for the argument when calling the function or procedure. This is the default.
* OUT indicates that the function or procedure will set the value of the argument.
* IN OUT indicates that a value for the argument can be supplied by you and can be set by the function or procedure.

[Example 8-8](http://docs.oracle.com/cd/B25329_01/doc/appdev.102/b25108/xedev_plsql.htm" \l "CHDHBFDD) Declaring a Local PL/SQL Procedure With IN OUT Parameters

DECLARE -- declare variables and subprograms

cityname VARCHAR2(20) := 'dhaka';

countryname VARCHAR2(25) := 'bangladesh';

-- declare a local procedure which can only be used in this block

PROCEDURE upper\_name ( v1 IN OUT VARCHAR2, v2 IN OUT VARCHAR2) AS

BEGIN

v1 := UPPER(v1); -- change the string to uppercase

v2 := UPPER(v2); -- change the string to uppercase

END upper\_name;

-- start of executable part of block

BEGIN

DBMS\_OUTPUT.PUT\_LINE(cityname || ' ' || countryname);--display initial values

upper\_name (cityname, countryname); -- call the procedure with parameters

DBMS\_OUTPUT.PUT\_LINE(cityname || ' ' || countryname); -- display new values

END;

/

[Example 8-9](http://docs.oracle.com/cd/B25329_01/doc/appdev.102/b25108/xedev_plsql.htm#CHDGBCFH) is an example of a declaration of a PL/SQL function in a PL/SQL block. Note that the value returned by the function is used directly in the DBMS\_OUTPUT.PUT\_LINE statement. Note that the v1 and v2 variables are declared as IN parameters to a subprogram. An IN parameter passes an initial value that is read inside of a subprogram, any update to the value of the parameter inside of the subprogram is not accessible outside of the subprogram.

[Example 8-9](http://docs.oracle.com/cd/B25329_01/doc/appdev.102/b25108/xedev_plsql.htm" \l "CHDGBCFH) Declaring a Local PL/SQL Function With IN Parameters

DECLARE -- declare variables and subprograms

cityname VARCHAR2(20) := 'DHAKA';

countryname VARCHAR2(25) := 'BANGLADESH';

-- declare local function which can only be used in this block

FUNCTION init\_name ( v1 IN VARCHAR2, v2 IN VARCHAR2)

RETURN VARCHAR2 AS

v3 VARCHAR2(45); -- this variable is local to the function

BEGIN

-- build a string that will be returned as the function value

v3 := v1 || ' + ' || v2 || ' = ' || INITCAP(v1) || ' ' || INITCAP(v2);

RETURN v3; -- return the value of v3

END init\_name;

-- start of executable part of block

BEGIN

-- call the function and display results

DBMS\_OUTPUT.PUT\_LINE(init\_name (cityname, countryname));

END;

/

[Example 8-10](http://docs.oracle.com/cd/B25329_01/doc/appdev.102/b25108/xedev_plsql.htm" \l "BABIIBDF) Declaring a Complex Local Procedure in a PL/SQL Block (both a variable and a numeric literal are passed as a parameter)

DECLARE -- declare variables and subprograms

partid NUMBER;

-- declare local procedure for this block

PROCEDURE avg\_min\_max\_sal (partid IN NUMBER) IS

desig VARCHAR2(20);

avg\_sal NUMBER;

min\_sal NUMBER;

max\_sal NUMBER;

BEGIN

-- determine the designation for the participant

SELECT designation INTO desig FROM works WHERE par\_id = partid;

-- calculate the average, minimum, and maximum salaries for that designation

SELECT AVG(salary), MIN(salary), MAX(salary) INTO avg\_sal, min\_sal, max\_sal

FROM works WHERE designation = desig;

-- display data

DBMS\_OUTPUT.PUT\_LINE ('Participant ID: ' || partid || ' Designation: ' || desig);

DBMS\_OUTPUT.PUT\_LINE ('The average salary for designation: ' || desig

|| ' is ' || TO\_CHAR(avg\_sal));

DBMS\_OUTPUT.PUT\_LINE ('The minimum salary for designation: ' || desig

|| ' is ' || TO\_CHAR(min\_sal));

DBMS\_OUTPUT.PUT\_LINE ('The maximum salary for designation: ' || desig

|| ' is ' || TO\_CHAR(max\_sal));

END avg\_min\_max\_sal; -- end of local procedure

-- start executable part of block

BEGIN

-- call the procedure with several employee IDs

partid := 6;

avg\_min\_max\_sal(partid);

avg\_min\_max\_sal(4);

END;

/

**8.3 Stored Procedures, Functions and Packages**

Stored procedures and functions (subprograms) can be compiled and stored in an Oracle Database XE, ready to be executed. Once compiled, it is a schema object known as a stored procedure or stored function, which can be referenced or called any number of times by multiple applications connected to Oracle Database XE. Both stored procedures and functions can accept parameters when they are executed (called). To execute a stored procedure or function, you only need to include its object name.

A package is a schema object that groups logically related PL/SQL types, variables, and subprograms. Packages usually have two parts, a specification (called the spec) and a body. The specification is the interface to the package. It declares the types, variables, constants, exceptions, and subprograms that can be referenced from outside of the package. The body defines the code for the subprograms.

### 8.3.1 Creating Stored Procedures With SQL CREATE PROCEDURE

The SQL CREATE PROCEDURE statement lets you create stored procedures that are stored in the database. You can use the optional OR REPLACE clause to modify an existing procedure without first dropping the procedure.

[Example 8-11](http://docs.oracle.com/cd/B25329_01/doc/appdev.102/b25108/xedev_programs.htm#BABGBEDE) is an example of a simple stored procedure that displays current date.

CREATE OR REPLACE PROCEDURE today\_is AS

BEGIN

-- display the current system date in long format

DBMS\_OUTPUT.PUT\_LINE( 'Today is ' || TO\_CHAR(SYSDATE, 'DL') );

END today\_is;

/

-- to call the procedure today\_is, you can use the following block

BEGIN

today\_is(); -- the parentheses are optional here

END;

/

### 8.3.1.1 Creating a Stored Procedure That Uses Parameters

When you create a procedure or function, you can specify parameters that are passed to the procedure or function when it is called (or invoked).

[Example 8-12](http://docs.oracle.com/cd/B25329_01/doc/appdev.102/b25108/xedev_programs.htm" \l "BABGBEDE) Creating a Stored Procedure That Uses Parameters

-- including OR REPLACE is more convenient when updating a subprogram

-- IN is the default for parameter declarations so it could be omitted

CREATE OR REPLACE PROCEDURE award\_bonus (part\_id IN NUMBER, bonus\_rate IN NUMBER)

AS

-- declare variables to hold values from table columns, use %TYPE attribute

part\_sal works.salary%TYPE;

-- declare an exception to catch when the salary is NULL

salary\_missing EXCEPTION;

BEGIN -- executable part starts here

-- select the column values into the local variables

SELECT salary INTO part\_sal FROM works WHERE par\_id = part\_id;

-- check if the salary for the participant is null, raise an exception

IF part\_sal IS NULL THEN

RAISE salary\_missing;

ELSE

IF part\_sal < 35000 THEN

-- if the salary is less than 35000, increase the salary by the bonus rate

-- for this example, do not make the actual update to the salary

-- UPDATE works SET salary = salary + salary \* bonus\_rate

-- WHERE par\_id = part\_id;

DBMS\_OUTPUT.PUT\_LINE('Participant ' || part\_id || ' receives a bonus: '

|| TO\_CHAR(part\_sal \* bonus\_rate) );

ELSE

DBMS\_OUTPUT.PUT\_LINE('Participant ' || part\_id

|| ' receives no bonus.');

END IF;

END IF;

EXCEPTION -- exception-handling part starts here

WHEN salary\_missing THEN

DBMS\_OUTPUT.PUT\_LINE('Participant ' || part\_id ||

' does not have a value for salary. No update.');

WHEN OTHERS THEN

NULL; -- for other exceptions do nothing

END award\_bonus;

/

-- the following BEGIN..END block calls, or executes, the award\_bonus procedure

-- using participant IDs 2 and 8 with the bonus rate 0.05 (5%)

BEGIN

award\_bonus(2, 0.05);

award\_bonus(8, 0.05);

END;

/

### 8.3.1.2 Creating Stored Functions With the SQL CREATE FUNCTION Statement

The SQL CREATE FUNCTION statement lets you create stored functions that are stored in an Oracle database. These stored (schema level) subprograms can be accessed from SQL. You can use the optional OR REPLACE clause to modify an existing function. [Example 8-13](http://docs.oracle.com/cd/B25329_01/doc/appdev.102/b25108/xedev_programs.htm#BABJEGJA) is an example of a function that returns a character string that contains the upper case instructor name and designation of an instructor. The example also show how to run (call) the function.

[Example 8-13](http://docs.oracle.com/cd/B25329_01/doc/appdev.102/b25108/xedev_programs.htm#BABJEGJA) Creating a Stored Function That Returns a String

CREATE OR REPLACE FUNCTION inst\_designation (instid NUMBER)

RETURN VARCHAR2 IS

instname instructor.inst\_id%TYPE; -- declare a variable same as inst\_name

desig instructor.designation%TYPE; --declare a variable same as designation

BEGIN

SELECT inst\_name, designation INTO instname, desig FROM instructor

WHERE inst\_id = instid;

RETURN ( 'Participant: ' || instid || ' - ' || UPPER(instname)

|| ', ' || UPPER(desig) );

END inst\_designation;

/

-- you can use the following block to call the function

DECLARE

instid NUMBER := 4; -- pick an instructor ID to test the function

BEGIN

-- display the output of the function

DBMS\_OUTPUT.PUT\_LINE(inst\_designation(instid) );

END;

/

-- you can also call a function from a SQL SELECT statement

-- using the dummy DUAL table

SELECT inst\_designation(4) FROM DUAL;

[Example 8-14](http://docs.oracle.com/cd/B25329_01/doc/appdev.102/b25108/xedev_programs.htm#BABGAGAG) is an example of a stored function that returns the calculated salary ranking for a specific participant based on the current minimum and maximum salaries of participant in the same department.

[Example 8-14](http://docs.oracle.com/cd/B25329_01/doc/appdev.102/b25108/xedev_programs.htm" \l "BABGAGAG) Creating a Stored Function That Returns a Number

CREATE OR REPLACE FUNCTION par\_sal\_ranking (parid NUMBER)

RETURN NUMBER IS

minsal works.salary%TYPE; -- declare a variable same as salary

maxsal works.salary%TYPE; -- declare a variable same as salary

deptname works.dept\_name%TYPE; -- declare a variable same as dept\_name

sal works.salary%TYPE; -- declare a variable same as salary

BEGIN

-- retrieve the department and salary for the specific participant ID

SELECT dept\_name, salary INTO deptname, sal FROM works WHERE par\_id = parid;

-- retrieve the minimum and maximum salaries for participant with the same dept.

SELECT MIN(salary), MAX(salary) INTO minsal, maxsal FROM works

WHERE dept\_name = deptname;

-- return the ranking as a decimal, based on the following calculation

RETURN ((sal - minsal)/(maxsal - minsal));

END par\_sal\_ranking;

/

-- create a PL/SQL block to call the function, you can use another subprogram too

-- because a function returns a value, it is called as part of a line of code

DECLARE

parid NUMBER := 5; -- pick an participant ID to test the function

BEGIN

-- display the output of the function, round to 2 decimal places

DBMS\_OUTPUT.PUT\_LINE('The salary ranking for participant ' || parid || ' is:'

|| ROUND(par\_sal\_ranking(parid),2) );

END;

/

### 8.3.2 Calling Stored Procedures or Functions

You can call a stored subprogram from a BEGIN ... END block or from another subprogram or a package. When calling a stored procedure or function, you can write the actual parameters using the following type of notation:

* **Positional notation:** You specify the same parameters in the same order as they are declared in the procedure. This notation is compact, but you must specify the parameters (especially literals) in the correct order.
* **Named notation:** You specify the name of each parameter and its value. An arrow (=>) serves as the association operator. The **order** of the parameters is **not significant**.
* **Mixed notation:** You specify the first parameters with positional notation, then switch to named notation for the last parameters.

[Example 8-15](http://docs.oracle.com/cd/B25329_01/doc/appdev.102/b25108/xedev_programs.htm" \l "BABBHHAI) Techniques for Calling Stored Procedures or Functions: this example calls the stored procedure in [Example 8-14](http://docs.oracle.com/cd/B25329_01/doc/appdev.102/b25108/xedev_programs.htm#BABBHHAI).

-- use a PL/SQL block to execute the procedure

BEGIN

award\_bonus(7, 0.05);

END;

/

-- using named notation for the parameters, rather than positional

BEGIN

award\_bonus(bonus\_rate=>0.1, emp\_id=>6);

END;

/

### 8.3.3 Creating Packages With the SQL CREATE PACKAGE Statement

To create packages, use the SQL CREATE PACKAGE and CREATE PACKAGE BODY statements. You can use these SQL statements in the SQL Commands page, the Script Editor page, the Object Browser page, or SQL Command Line (SQL\*Plus). In [Example 8-16](http://docs.oracle.com/cd/B25329_01/doc/appdev.102/b25108/xedev_programs.htm#BABCEFJH) and [Example 8-17](http://docs.oracle.com/cd/B25329_01/doc/appdev.102/b25108/xedev_programs.htm#BABEHEAB), the OR REPLACE option is used to update an existing package without having to first drop the package.

In [Example 8-16](http://docs.oracle.com/cd/B25329_01/doc/appdev.102/b25108/xedev_programs.htm#BABCEFJH), the par\_actions package specification contains two procedures that update the employees table and one function that provides information. The **package specification** provides the **declaration** of the subprograms. The **package body** provides the **contents** of the subprograms.

[Example 8-16](http://docs.oracle.com/cd/B25329_01/doc/appdev.102/b25108/xedev_programs.htm" \l "BABCEFJH) Creating a Package Specification

CREATE OR REPLACE PACKAGE works\_actions AS -- package specification

PROCEDURE add\_works(parid NUMBER, orgid NUMBER, deptname VARCHAR2, desig VARCHAR2, parsal NUMBER, hdate DATE);

PROCEDURE remove\_works (parid NUMBER);

FUNCTION par\_sal\_ranking (parid NUMBER) RETURN NUMBER;

END works\_actions;

/

In [Example 8-17](http://docs.oracle.com/cd/B25329_01/doc/appdev.102/b25108/xedev_programs.htm#BABEHEAB), the works\_actions package body is created. The package body provides the contents of the subprograms in the package specification.

[Example 8-17](http://docs.oracle.com/cd/B25329_01/doc/appdev.102/b25108/xedev_programs.htm" \l "BABCEFJH) Creating a Package Body

CREATE OR REPLACE PACKAGE BODY works\_actions AS -- package body

-- code for procedure add\_works, which adds a new works record

PROCEDURE add\_works(parid NUMBER, orgid NUMBER, deptname VARCHAR2, desig VARCHAR2, parsal NUMBER, hdate DATE)

IS

min\_sal works.salary%TYPE; -- variable to hold minimum salary for desig

max\_sal works.salary%TYPE; -- variable to hold maximum salary for desig

seq\_value NUMBER; -- variable to hold next sequence value

BEGIN

INSERT INTO works VALUES (parid, orgid, deptname, design, parsal, hdate);

SELECT min\_salary INTO min\_sal FROM works WHERE org\_id = orgid;

SELECT max\_salary INTO max\_sal FROM works WHERE org\_id = orgid;

IF parsal > max\_sal THEN

DBMS\_OUTPUT.PUT\_LINE('Warning: ' || TO\_CHAR(parsal)

|| ' is greater than the maximum salary '

|| TO\_CHAR(max\_sal) || ' for the organization ' || orgid );

ELSIF sal < min\_sal THEN

DBMS\_OUTPUT.PUT\_LINE('Warning: ' || TO\_CHAR(sal)

|| ' is less than the minimum salary '

|| TO\_CHAR(min\_sal) || ' for the organization ' || orgid );

END IF;

END add\_works;

-- code for procedure remove\_works, which removes an existing record

PROCEDURE remove\_works (parid NUMBER) IS

partid works.par\_id%TYPE;

deptname works.dept\_name%TYPE

desig works.designation%TYPE

BEGIN

SELECT par\_id, dept\_name, designation INTO partid, deptname, desig FROM works WHERE par\_id = parid;

DELETE FROM works WHERE par\_id = parid;

DBMS\_OUTPUT.PUT\_LINE('Participant: ' || TO\_CHAR(parid) || ', '

|| deptname || ', ' || desig || ' has been deleted.');

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('Participant ID: ' || TO\_CHAR(parid) || ' not found.');

END remove\_works;

-- code for function par\_sal\_ranking, which calculates the salary ranking of

-- the participant based on the minimum and maximum salaries for the same department

FUNCTION par\_sal\_ranking (parid NUMBER) RETURN NUMBER IS

minsal works.salary%TYPE; -- declare a variable same as salary

maxsal works.salary%TYPE; -- declare a variable same as salary

deptname works.dept\_name%TYPE; -- declare a variable same as dept\_name

sal works.salary%TYPE; -- declare a variable same as salary

BEGIN

-- retrieve the department and salary for the specific participant ID

SELECT dept\_name, salary INTO deptname, sal FROM works WHERE par\_id = parid;

-- retrieve the minimum and maximum salaries for participant with the same dept.

SELECT MIN(salary), MAX(salary) INTO minsal, maxsal FROM works

WHERE dept\_name = deptname;

-- return the ranking as a decimal, based on the following calculation

RETURN ((sal - minsal)/(maxsal - minsal));

END par\_sal\_ranking;

/

END works\_actions;

/

### 8.3.3.1 Calling Procedures and Functions in Packages

To call the procedures or functions of the works\_actions package created in [Example 8-17](http://docs.oracle.com/cd/B25329_01/doc/appdev.102/b25108/xedev_programs.htm#BABEHEAB), you can execute the statements in [Example 8-18](http://docs.oracle.com/cd/B25329_01/doc/appdev.102/b25108/xedev_programs.htm#BABGEGJG). The subprograms can be executed in a BEGIN .. END block or from another subprogram.

[Example 8-18](http://docs.oracle.com/cd/B25329_01/doc/appdev.102/b25108/xedev_programs.htm" \l "BABGEGJG) Calling a Subprogram in a Package

-- the following calls the par\_sal\_ranking function in the works\_actions package

DECLARE

parid NUMBER := 5; -- use a test value for the participant id

BEGIN

DBMS\_OUTPUT.put\_line('The salary ranking for participant ' || parid || ' is: '

|| ROUND(works\_actions.par\_sal\_ranking(parid),2) );

END;

/

-- the following calls the add\_works subprogram in the works\_actions package -- with the associated parameter values

BEGIN

works\_actions.add\_works(9, 1, 'IT', 'VP', 55000, '01-APR-2006');

END;

/

-- the following calls the remove\_works subprogram in the works\_actions

-- package that we just added

BEGIN

works\_actions.remove\_works(9);

END;

/

Top of Form