Oracle (PL/SQL)

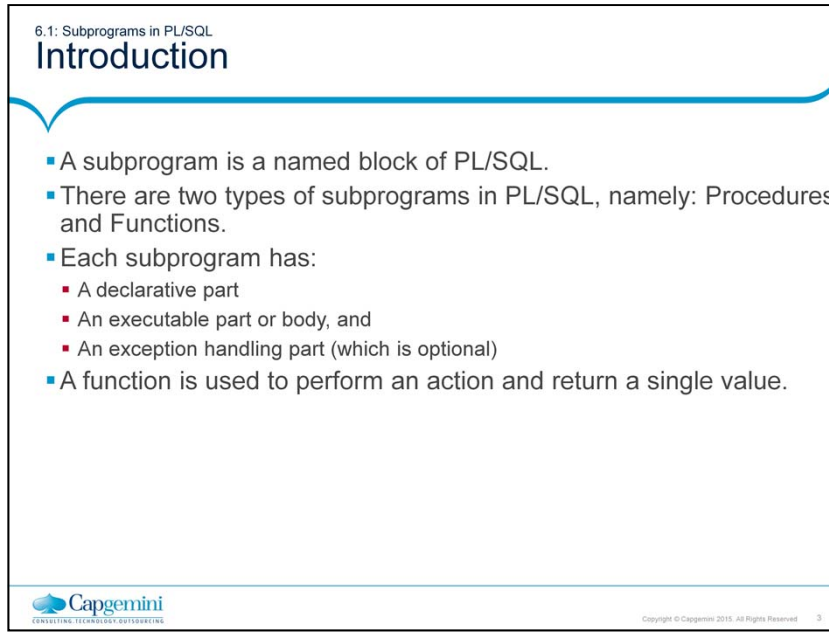
Procedures, Functions, and Packages

Lesson Objectives

- To understand the following topics:
  - Subprograms in PL/SQL
  - Anonymous blocks versus Stored Subprograms
  - Procedure
    - Subprogram Parameter modes
  - Functions
  - Packages
    - Package Specification and Package Body
  - Autonomous Transactions

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 2



Subprograms in PL/SQL:

- The subprograms are compiled and stored in the Oracle database as “stored programs”, and can be invoked whenever required. As the subprograms are stored in a compiled form, when called they only need to be executed. Hence this arrangement saves time needed for compilation.
- When a client executes a procedure or function, the processing is done in the server. This reduces the network traffic.
- Subprograms provide the following advantages:
  - They allow you to write a PL/SQL program that meets our need.
  - They allow you to break the program into manageable modules.
  - They provide reusability and maintainability for the code.

6.1: Subprograms in PL/SQL

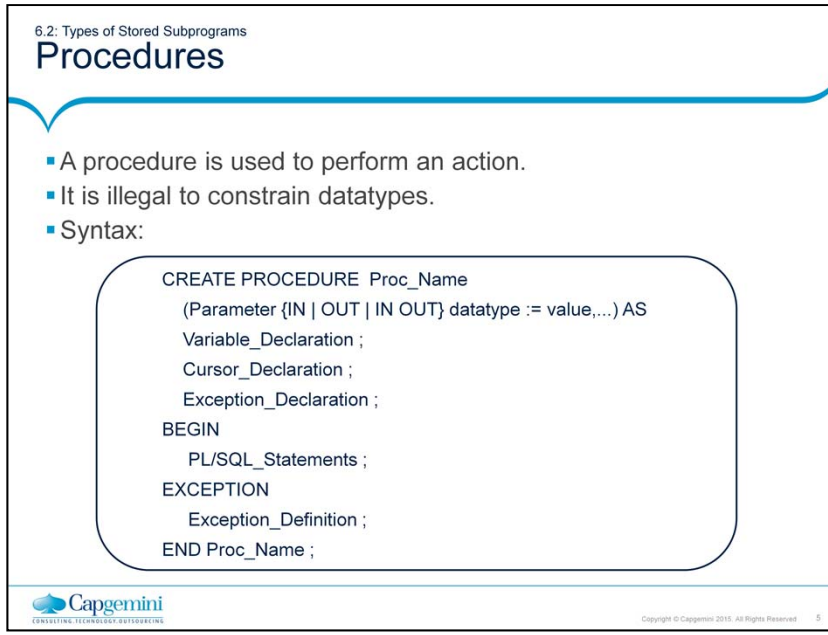
Comparison

- Anonymous Blocks & Stored Subprograms Comparison

Anonymous Blocks	Stored Subprograms/Named Blocks
1. Anonymous Blocks do not have names.	1. Stored subprograms are named PL/SQL blocks.
2. They are interactively executed. The block needs to be compiled every time it is run.	2. They are compiled at the time of creation and stored in the database itself. Source code is also stored in the database.
3. Only the user who created the block can use the block.	3. Necessary privileges are required to execute the block.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

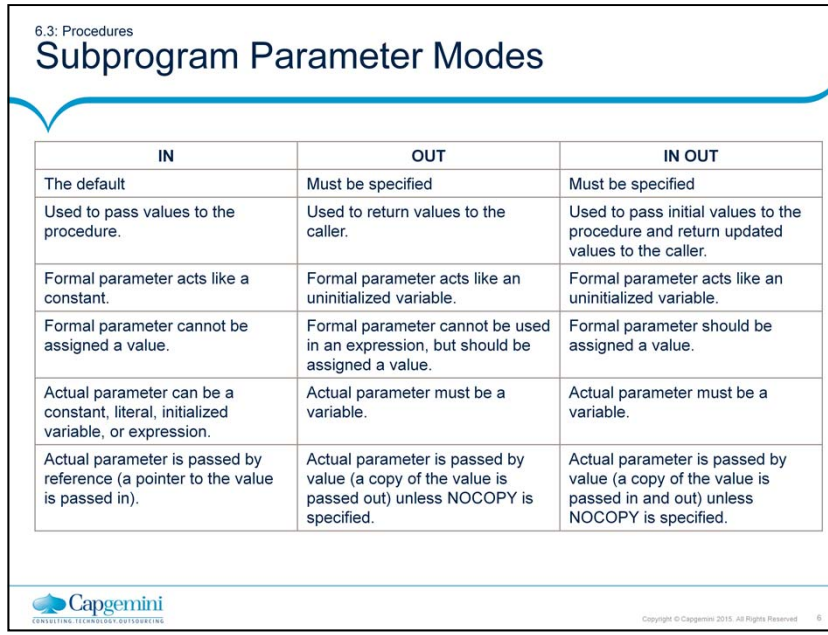
 4



Procedures:

- **A procedure is a subprogram used to perform a specific action.**
- **A procedure contains two parts:**
  - **the specification, and**
  - **the body**
- **The procedure specification begins with CREATE and ends with procedure name or parameters list. Procedures that do not take parameters are written without a parenthesis.**
- **The procedure body starts after the keyword IS or AS and ends with keyword END.**

contd.



Subprogram Parameter Modes:

- **You use "parameter modes" to define the behavior of "formal parameters". The three parameter modes are IN (the default), OUT, and INOUT. The characteristics of the three modes are shown in the slide.**
- **Any parameter mode can be used with any subprogram.**
- **Avoid using the OUT and INOUT modes with functions.**
- **To have a function return multiple values is a poor programming practice. Besides functions should be free from side effects, which change the values of variables that are not local to the subprogram.**
- **Example1:**

```
CREATE PROCEDURE split_name
(
  phrase IN VARCHAR2, first OUT VARCHAR2, last OUT
      VARCHAR2
)
IS
  first := SUBSTR(phrase, 1, INSTR(phrase, ' ')-1);
  last := SUBSTR(phrase, INSTR(phrase, ' ')+1);
  IF first = 'John' THEN
    DBMS_OUTPUT.PUT_LINE('That is a common first name.');
  END IF;
END;
```

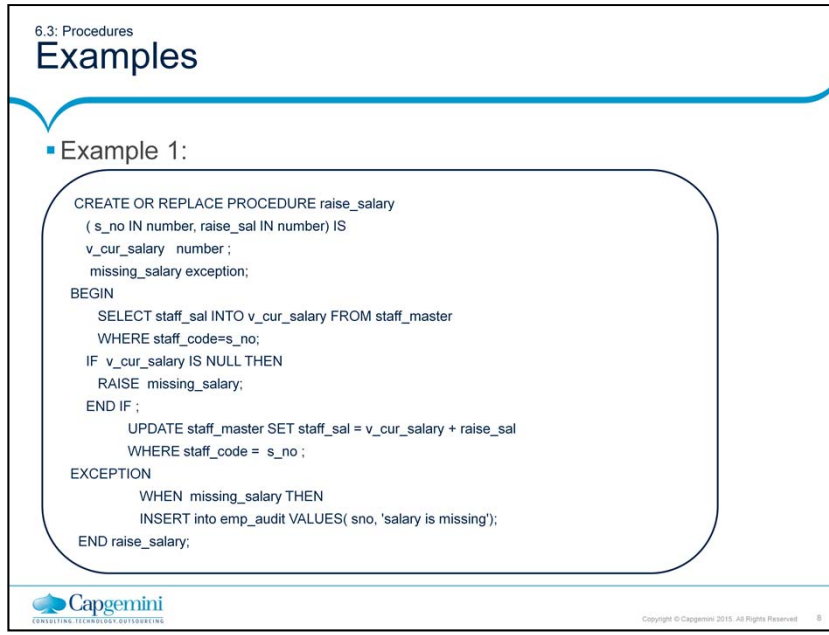
Subprogram Parameter Modes (contd.):

Examples:

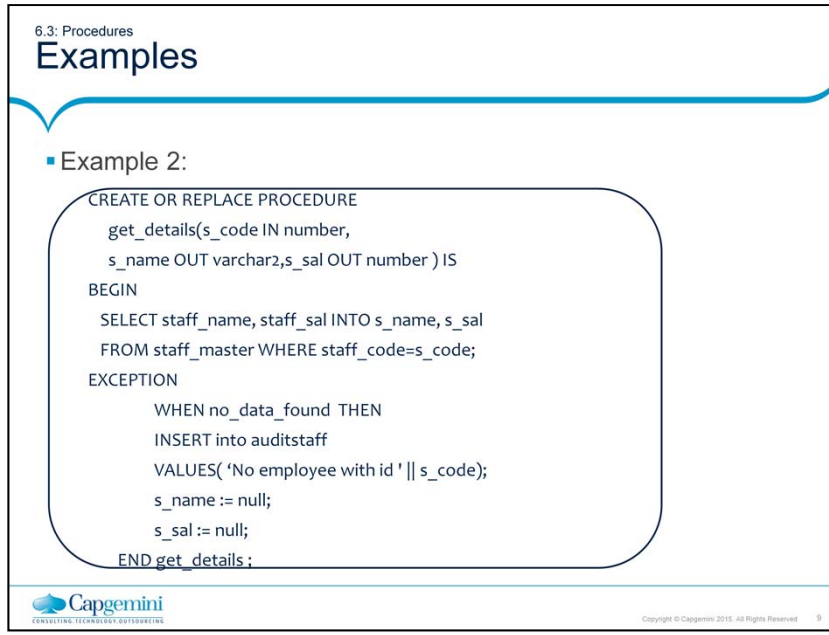
Example 2:

```
SQL > SET SERVEROUTPUT ON
SQL > CREATE OR REPLACE PROCEDRE PROC1 AS
    2   BEGIN
    3       DBMS_OUTPUT.PUT_LINE('Hello from procedure …');
    4   END;
    5   /
Procedure created.
SQL > EXECUTE PROC1
Hello from procedure …
PL/SQL procedure successfully created.
```

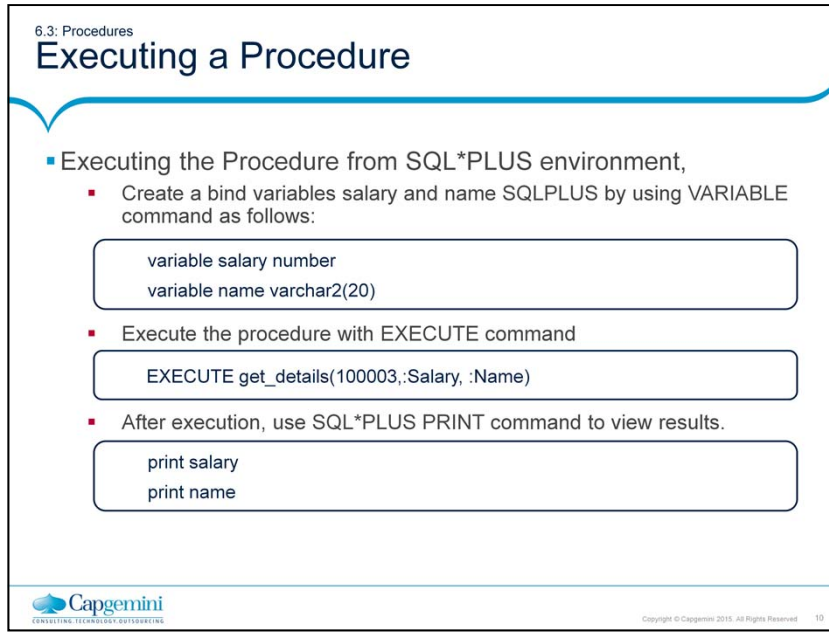
```
SQL > CREATE OR REPLACE      PROCEDURE PROC2
    2      (N1 IN NUMBER, N2 IN NUMBER, TOT OUT NUMBER) IS
    3   BEGIN
    4      TOT := N1 + N2;
    5   END;
    6   /

Procedure created.

SQL > VARIABLE T NUMBER
SQL > EXEC PROC2(33, 66, :T)

PL/SQL procedure successfully completed.

SQL > PRINT T

           T
----------
          99
```



The procedure example on the slide modifies the salary of staff member. It also handles exceptions appropriately. In addition to the above shown exception you can also handle "NO_DATA_FOUND" exception. The procedure accepts two parameters which is the staff_code and amount that has to be given as raise to the staff member.



The procedure on the slide accept three parameters, one is IN mode and other two are OUT mode. The procedure retrieves the name and salary of the staff member based on the staff_code passed to the procedure. The S_NAME and S_SAL are the OUT parameters that will return the values to the calling environment



Procedures can be executed through command line as shown on the slide or can be called from other procedures/functions/Anonymous PL/SQL blocks.

On the slide the first snippet declares two variables viz. salary and name. The second snippet calls the procedure and passes the actual parameters. The first is a literal string and the next two parameters are empty variables which will be assigned with values within the procedure.

Calling the procedure from an anonymous PL/SQL block

```
DECLARE
      s_no  number(10):=&sno;
      sname varchar2(10);
      sal  number(10,2);
BEGIN
      get_details(s_no,sname,sal);
      dbms_output.put_line('Name:'||sname||'Salary'||sal);
END;
```

Parameter default values:

- Like variable declarations, the formal parameters to a procedure or function can have default values.
- If a parameter has default values, it does not have to be passed from the calling environment.
  - If it is passed, actual parameter will be used instead of default.
- Only IN parameters can have default values.

Examples:

Example 1:

```
PROCEDURE Create_Dept( New_Deptno IN NUMBER,
   New_Dname IN  VARCHAR2  DEFAULT 'TEMP') IS
BEGIN
  INSERT INTO  department_master
            VALUES ( New_Deptno, New_Dname, New_Loc)
            ;
END ;
```

Example 2:

Now consider the following calls to Create_Dept.

```
BEGIN
Create_Dept( 50);
--  Actual  call will be  Create_Dept ( 50,
'TEMP', 'TEMP')

Create_Dept ( 50, 'FINANCE');
-- Actual call will be  Create_Dept ( 50,
'FINANCE' ,'TEMP')

Create_Dept( 50, 'FINANCE', 'BOMBAY') ;
-- Actual  call will be  Create_Dept(50,
'FINANCE', 'BOMBAY' )

END;
```

Procedures (contd.):

Using Positional, Named, or Mixed Notation for Subprogram Parameters:

- When calling a subprogram, you can write the actual parameters by using either Positional notation, Named notation, or Mixed notation.
  - **Positional notation:** You specify the same parameters in the same order as they are declared in the procedure. This notation is compact, but if you specify the parameters (especially literals) in the wrong order, the bug can be hard to detect. You must change your code if the procedure's parameter list changes.
  - **Named notation:** You specify the name of each parameter along with 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", and then switch to "Named notation" for the last parameters. You can use this notation to call procedures that have some "required parameters", followed by some "optional parameters".
- We have already seen a few examples of calling procedures with Positional notation.
- The example shows calling the Create_Dept procedure with named notations

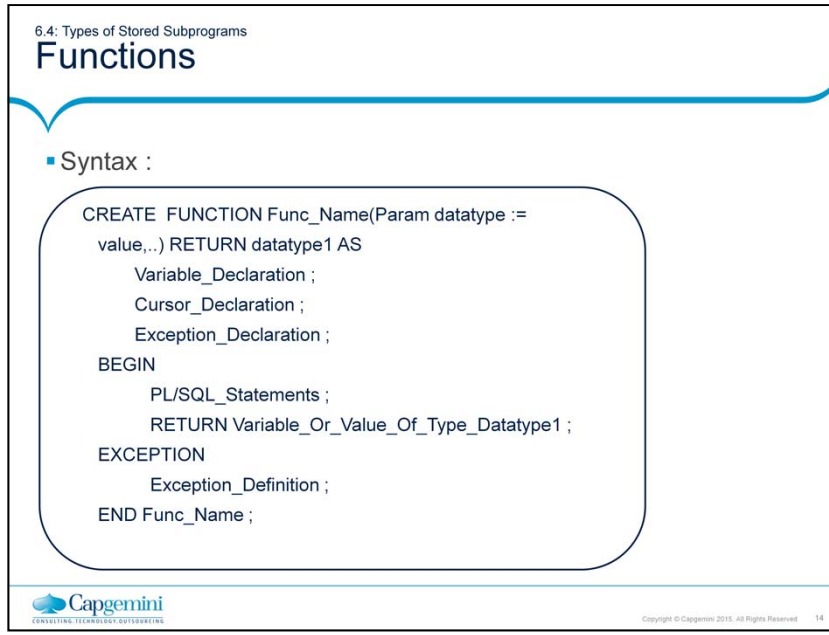
```
Create_Dept (New_Deptno=> 50,
New_Dname=>'FINANCE');
```

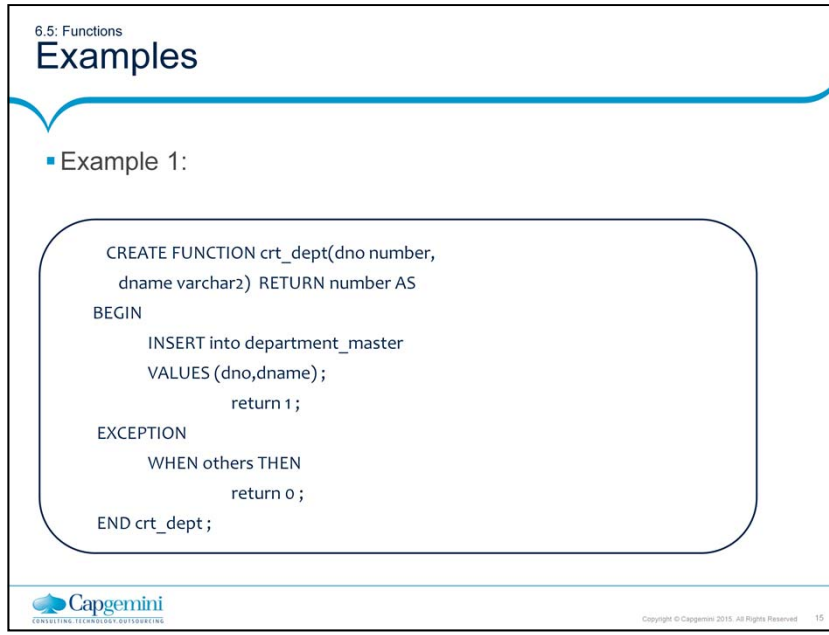
6.4: Types of Stored Subprograms

Functions

- A function is similar to a procedure.
- A function is used to compute a value.
  - A function accepts one or more parameters, and returns a single value by using a return value.
  - A function can return multiple values by using OUT parameters.
  - A function is used as part of an expression, and can be called as Lvalue = Function_Name(Param1, Param2, …….)
  - Functions returning a single value for a row can be used with SQL statements.

 13

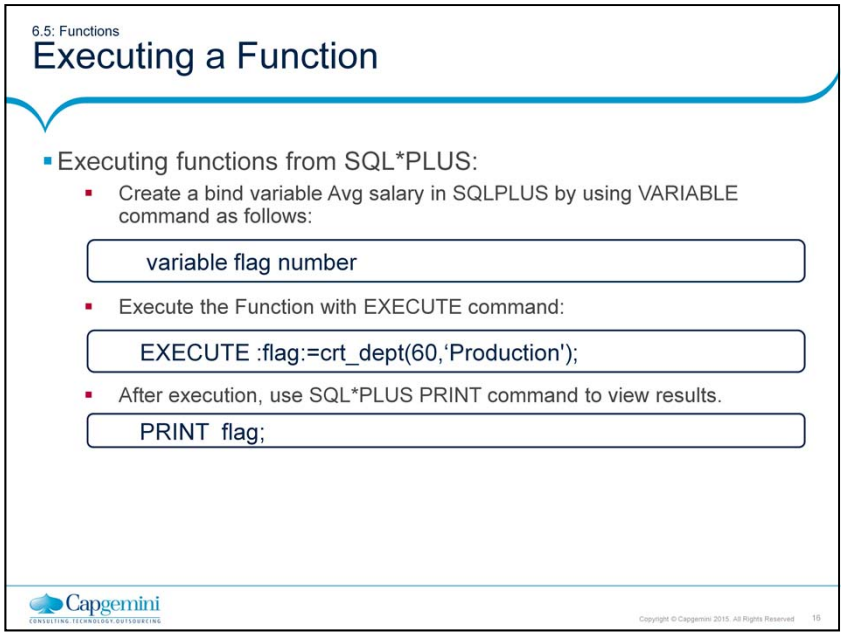

6.4: Types of Stored Subprograms
Functions
Syntax :
CREATE FUNCTION Func_Name(Param datatype :=
value,..) RETURN datatype1 AS
Variable_Declaration ;
Cursor_Declaration ;
Exception_Declaration ;
BEGIN
PL/SQL_Statements ;
RETURN Variable_Or_Value_Of_Type_Datatype1 ;
EXCEPTION
Exception_Definition ;
END Func_Name ;
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING
Copyright © Capgemini 2015. All Rights Reserved
14



Example 2:

- Function to calculate average salary of a department:
  - Function returns average salary of the department
  - Function returns -1, in case no employees are there in the department.
  - Function returns -2, in case of any other error.

```
CREATE OR REPLACE FUNCTION
get_avg_sal(p_deptno in number) RETURN number as
   V_Sal number;
BEGIN
            SELECT Trunc(Avg(staff_sal)) INTO V_Sal
            FROM staff_master
            WHERE deptno=P_Deptno;
            IF v_sal is null THEN
                        v_sal := -1 ;
         END IF;
                        return v_sal;
EXCEPTION
       WHEN others THEN
              return -2;  --signifies any other errors
END get_avg_sal;
```



Functions can also be executed through command line as shown on the slide or can be called from other procedures/functions/Anonymous PL/SQL blocks.

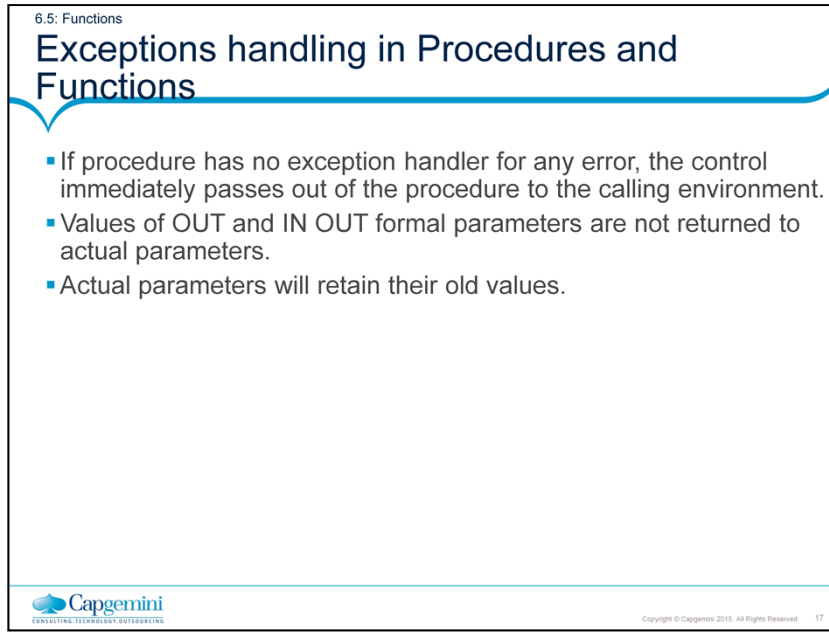
The second snippet calls the function and passes the actual parameters. The variable declared earlier is used for collecting the return value from the function

Calling the function from an anonymous PL/SQL block

```
DECLARE
   avgsalary  number;
BEGIN
      avgsalary:=_get_avg_sal(20);
      dbms_output.put_line('The average salary of Dept 20
is'||
      avgsalary);
END;
```

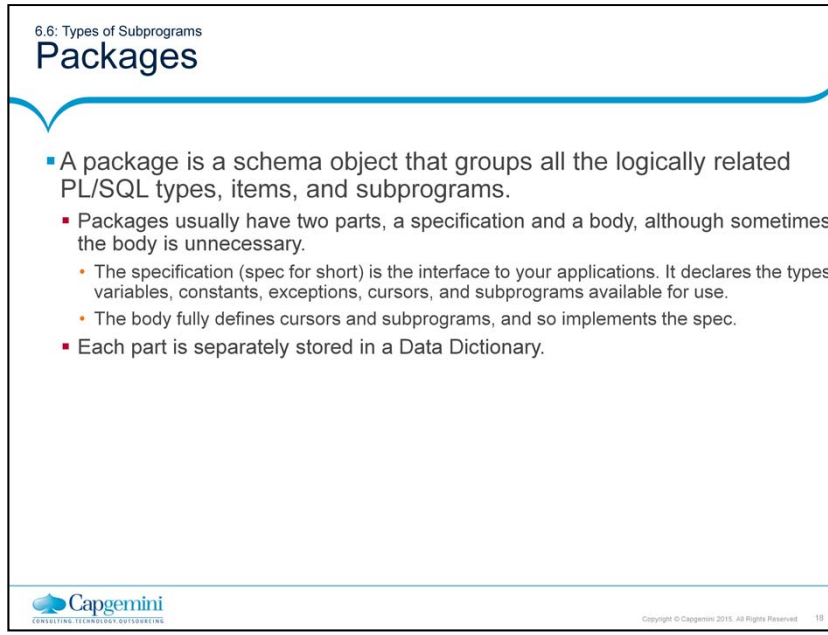
```
SELECT  get_avg_sal(30) FROM staff_master;
```

Calling function using a Select statement



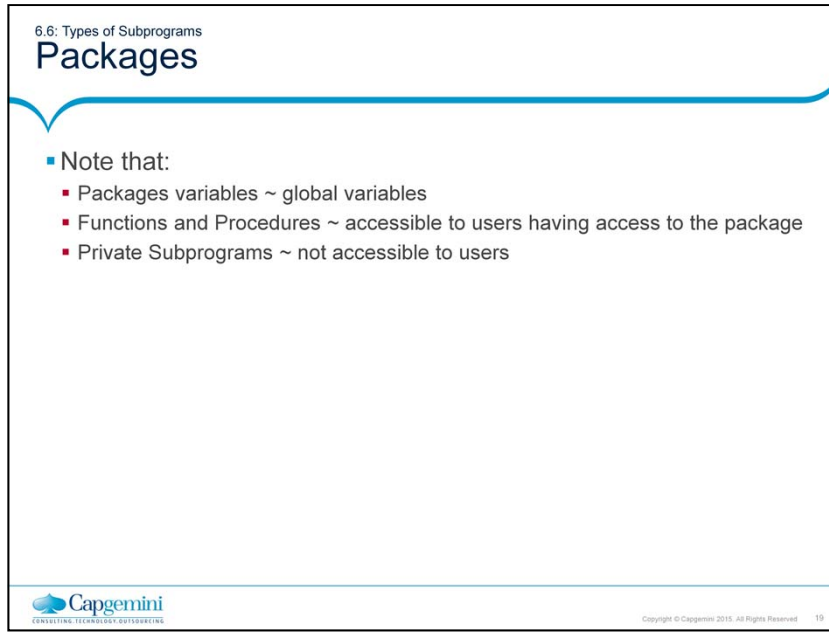
Exceptions raised inside Procedures and Functions:

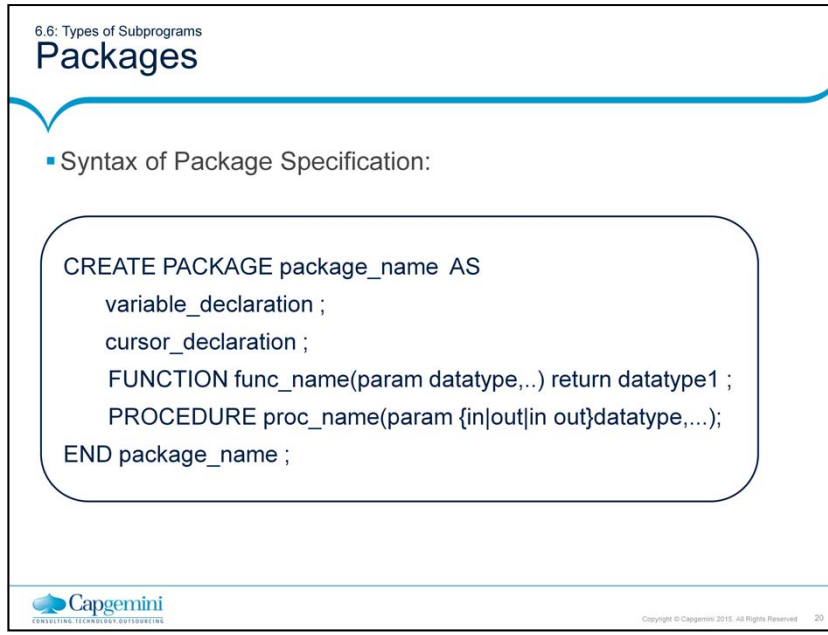
- If an error occurs inside a procedure, an exception (pre-defined or user-defined) is raised.



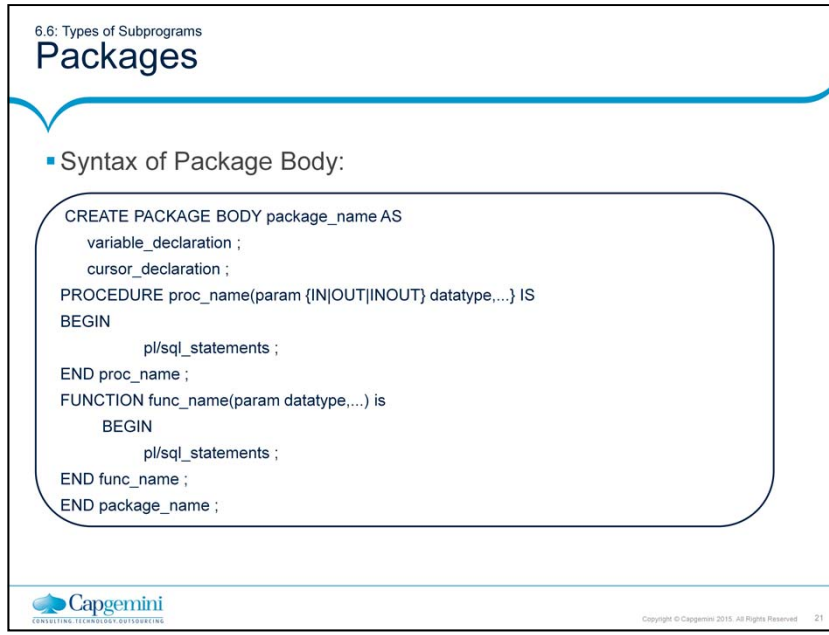
Packages:

- Packages are PL/SQL constructs that allow related objects to be stored together. A Package consists of two parts, namely “Package Specification” and “Package Body”. Each of them is stored separately in a “Data Dictionary”.
- **Package Specification:** It is used to declare functions and procedures that are part of the package. Package Specification also contains variable and cursor declarations, which are used by the functions and procedures. Any object declared in a Package Specification can be referenced from other PL/SQL blocks. So Packages provide global variables to PL/SQL.
- **Package Body:** It contains the function and procedure definitions, which are declared in the Package Specification. The Package Body is optional. If the Package Specification does not contain any procedures or functions and contains only variable and cursor declarations then the body need not be present.
- All functions and procedures declared in the Package Specification are accessible to all users who have permissions to access the Package. Users cannot access subprograms, which are defined in the Package Body but not declared in the Package Specification. They can only be accessed by the subprograms within the Package Body. This facility is used to hide unwanted or sensitive information from users.
- A Package generally consists of functions and procedures, which are required by a specific application or a particular module of an application.

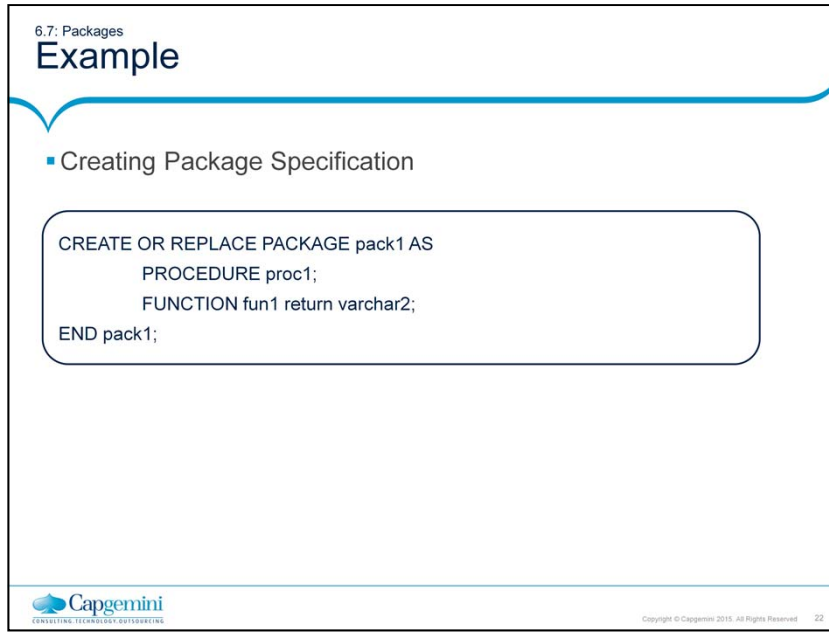

6.6: Types of Subprograms
Packages
Note that:
Packages variables ~ global variables
Functions and Procedures ~ accessible to users having access to the package
Private Subprograms ~ not accessible to users
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING
Copyright © Capgemini 2015. All Rights Reserved
19



The package specification can contain variables,cursors, procedure and functions. Whatever is specified within the packages are global by default and are accessible to users who have the privileges on the package



The package body should contain all the procedures and function declared in the package specification. Any variables and cursors declared within the package body are local to the package body and are accessible only within the package. The package body can contain additional procedures and functions apart from the ones declared in package body. The procedures/functions are local to the package and cannot be accessed by any user outside the package.


6.7: Packages
Example
Creating Package Specification
CREATE OR REPLACE PACKAGE pack1 AS
PROCEDURE proc1;
FUNCTION fun1 return varchar2;
END pack1;
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING
Copyright © Capgemini 2015. All Rights Reserved
22

6.7: Packages

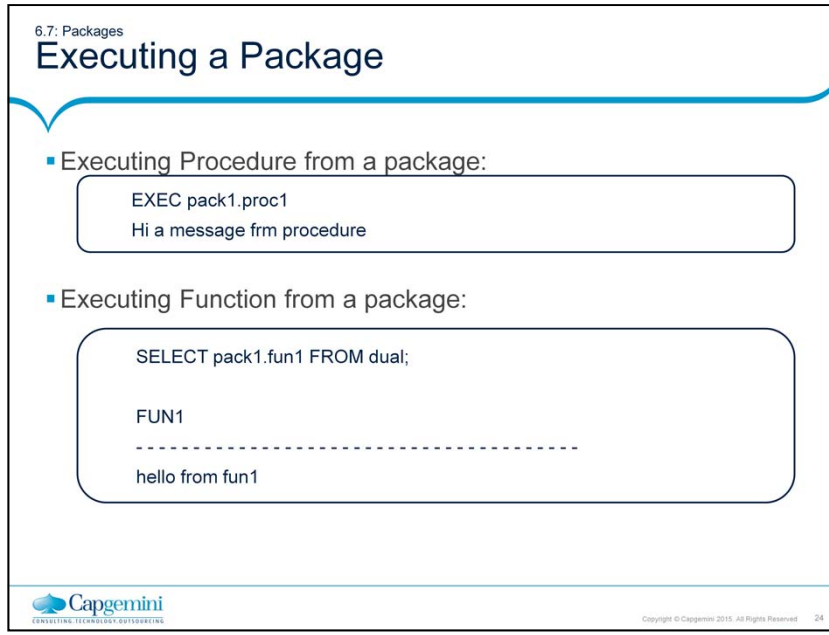
Example

- Creating Package Body

```
CREATE OR REPLACE PACKAGE BODY pack1 AS
   PROCEDURE proc1 is
   BEGIN
         dbms_output.put_line('hi a message frm procedure');
   END proc1;
         function fun1 return varchar2 is
   BEGIN
         return ('hello from fun1');
   END fun1;
END pack1;
```

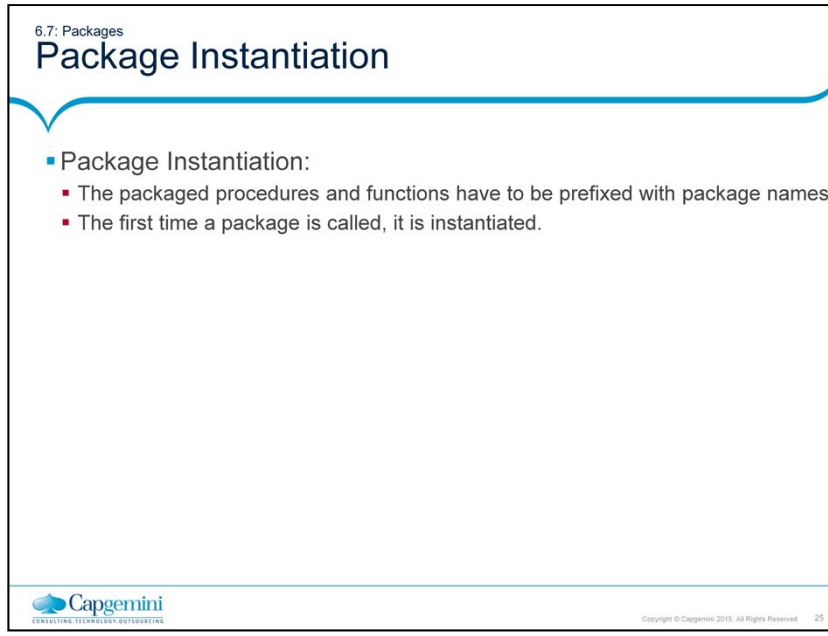
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 23



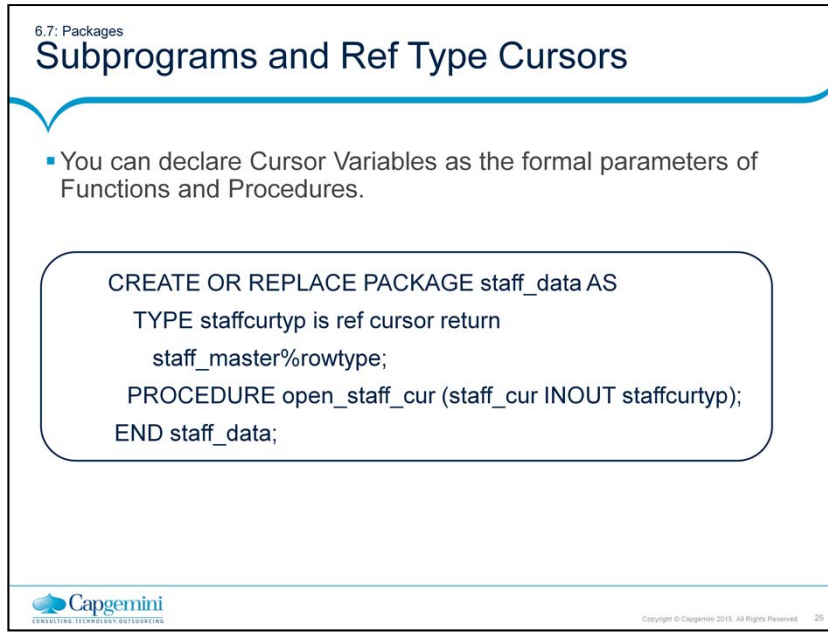
Note:

If the specification of the package declares only types, constants, variables, and exceptions, then the package body is not required there. This type of packages only contains global variables that will be used by subprograms or cursors.



Package Instantiation:

- The procedure and function calls are the same as in standalone subprograms.
- The packaged procedures and functions have to be prefixed with package names.
- The first time a package is called, it is instantiated.
  - ➢ This means that the package is read from disk into memory, and P-CODE is run.
  - ➢ At this point, the memory is allocated for any variables defined in the package.
    - Each session will have its own copy of packaged variables, so there is no problem of two simultaneous sessions accessing the same memory locations.



Subprograms and Ref Type Cursors:

- You can declare Cursor Variables as the formal parameters of Functions and Procedures.
- In the following example, you define the REF CURSOR type staffCurTyp, then declare a Cursor Variable of that type as the formal parameter of a procedure:

```
DECLARE
TYPE staffCurTyp IS REF CURSOR RETURN
staff_master%ROWTYPE;
PROCEDURE open_staff_cv (staff_cv IN OUT
staffCurTyp) IS
```

- Typically, you open a Cursor Variable by passing it to a stored procedure that declares a Cursor Variable as one of its formal parameters.
- The packaged procedure shown in the slide, for example, opens the cursor variable emp_cur.

6.7: Packages

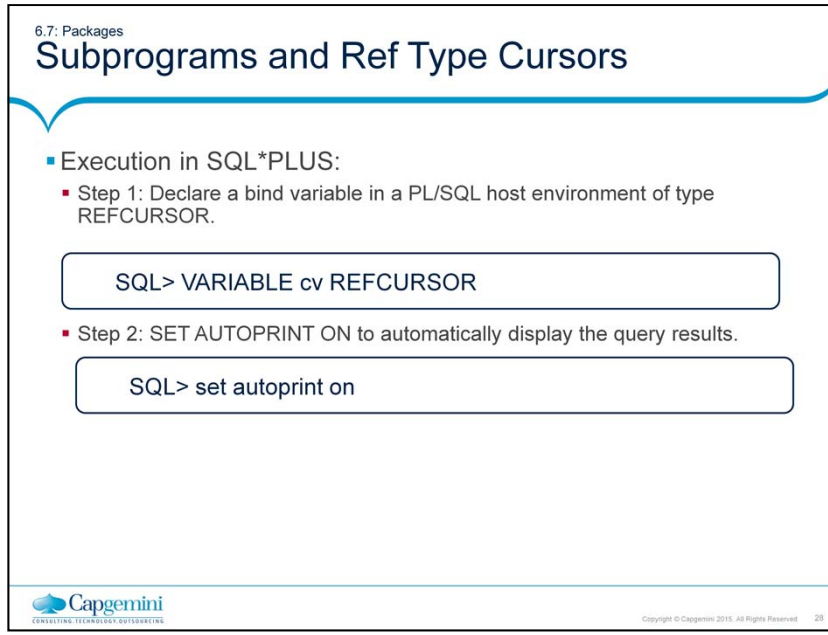
Subprograms and Ref Type Cursors

- Note: Cursor Variable as the formal parameter should be in IN OUT mode.

```
CREATE OR REPLACE PACKAGE BODY staff_data AS
PROCEDURE open_staff_cur (staff_cur INOUT staffcurtyp) IS
BEGIN
    OPEN staff_cur  for SELECT * FROM staff_master;
    end open_staff_cur;
END emp_data;
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 27



Subprograms and Ref Type Cursors: Execution in SQL*PLUS:

- When you declare a Cursor Variable as the formal parameter of a subprogram that opens the cursor variable, you must specify the IN OUT mode. That way, the subprogram can pass an open cursor back to the caller.
- To see the value of the Cursor Variable on the SQL prompt, you need to do following:
  - Declare a bind variable in a PL/SQL host environment of type REFCURSOR as shown below. The SQL*Plus datatype REFCURSOR lets you declare Cursor Variables, which you can use to return query results from stored subprograms.

    SQL> VARIABLE cv REFCURSOR
  - Use the SQL*Plus command SET AUTOPRINT ON to automatically display the query results.

    SQL> set autoprint on
  - Now execute the package with the specified procedure along with the cursor as follows :

    SQL> execute emp_data.open_emp_cur(:cv);

6.7: Packages

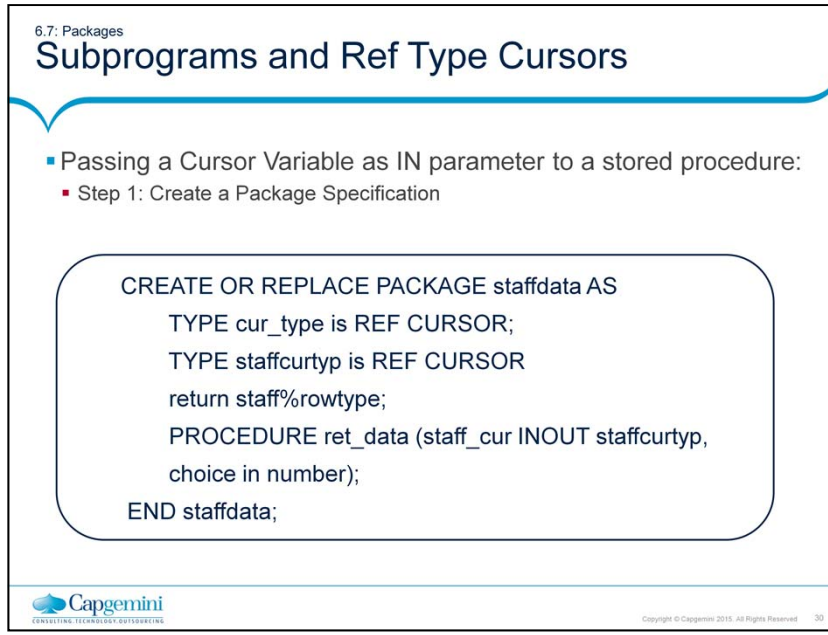
Subprograms and Ref Type Cursors

- Step 3: Execute the package with the specified procedure along with the cursor as follows:

```
SQL> execute staff_data.open_staff_cur(:cv);
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 29



Subprograms and Ref Type Cursors: Passing a Cursor Variable:

- You can pass a Cursor Variable and an IN parameter to a stored procedure, which will execute the queries with different return types.
- In the example shown in the slide, you are passing the cursor as well as the number variable as choice. Depending on the choice you can write multiple queries, and retrieve the output from the cursor.
- When called, the procedure opens the cursor variable emp_cur for the chosen query.

6.7: Packages

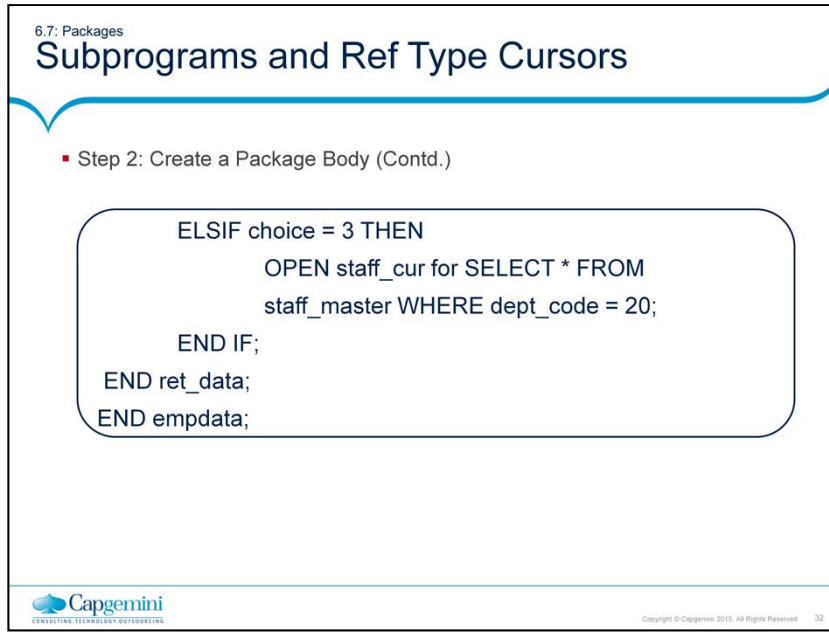
Subprograms and Ref Type Cursors (Contd.)

- Step 2: Create a Package Body:

```
CREATE OR REPLACE PACKAGE BODY staffdata AS
    PROCEDURE ret_data (staff_cur INOUT staffcurtyp,
        choice IN number) is
BEGIN
    IF choice = 1 THEN
            OPEN staff_cur for select * FROM staff_master
              WHERE staff_dob is not null;
    ELSIF choice = 2 THEN
             OPEN staff_cur for SELECT * FROM staff_master
              WHERE staff_sal > 2500;
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 31



Step 3: To retrieve the values from the cursor:

- Define a variable in SQL *PLUS environment using variable command.
- Set the autoprint command on the SQL prompt.
- Call the procedure with the package name and the relevant parameters.

```
SQL> variable cur refcursor
SQL> set autoprint on
SQL> execute staffdata.ret_data(:cur,1);
```

Subprograms and Ref Type Cursors: Passing a Cursor Variable (contd.):

- In a similar manner, you can pass the Cursor Variable as (: cur) and '2' number for second choice in the EmpData.ret_data procedure. This will give you the output for all the employees who have salary above 2500.
- To see the output of the third cursor, use the same package.procedure name with the ': cur' host variable, and choice value which shows all the employees having department number as 20.
- We can also create a package with the different REF CURSOR TYPES available (that is define the REF CURSOR type in a separate package), and then reference that type in the standalone procedure.
- Example 1: Create a package as shown below:

```
SQL> CREATE or replace PACKAGE cv_types AS
                                        TYPE
GenericCurTyp IS REF CURSOR;
TYPE staffCurTyp IS REF CURSOR RETURN
staff_master%ROWTYPE;
            TYPE deptCurTyp IS REF CURSOR
RETURN department_master%ROWTYPE;
 END cv_types;
 /
Package created.
```

```
SQL> CREATE or REPLACE PROCEDURE open_pro
(generic_cv IN OUT
      cv_types.GenericCurTyp,choice IN NUMBER) IS
 BEGIN
  IF choice = 1 THEN
OPEN generic_cv FOR SELECT * FROM staff_master;
  ELSIF choice = 2 THEN
      OPEN generic_cv FOR SELECT * FROM
department_master;
  ELSIF choice = 3 THEN
            OPEN generic_cv FOR SELECT * FROM item;
    END IF;
END open_pro;
/
Package created.
```

- Example 2: Create a standalone procedure that references the REF CURSOR type GenericCurTyp, which is defined in the package cv_types. Hence create a procedure as shown below:

contd.

Subprograms and Ref Type Cursors: Passing a Cursor Variable (contd.):

- Open_ procedure, which has a cursor parameter generic pro is an independent _cv, which refers to type REF CURSOR defined in the cv_types package. You can pass a Cursor Variable and Selector to a stored procedure that executes queries with different return types (that is what you have done in the Open_pro procedure). When you call this procedure with the Generic_cv cursor along with the Selector value, the generic_cv cursor gets open and it retrieves the values from the different tables.
- To execute this procedure you need to create the variable of type REFCURSOR, and pass that variable in the Open_pro procedure to see the output.
- For example:

```
SQL> execute open_pro(:cv,2);
```

This output is that for the choice number 2, that is the Cursor Variable will show all the rows from the Dept table.

6.7: Packages

Autonomous transactions

- Autonomous transactions are useful for implementing:
  - transaction logging,
  - counters, and
  - other such actions, which needs to be performed independent of whether the calling transaction is committed or rolled-back
- Autonomous transactions:
  - are independent of the parent transaction.
  - do not inherit the characteristic of the parent (calling) transaction.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 35

6.7: Packages

Autonomous transactions

- Note that:
  - Any changes made cannot be seen by the calling transaction unless they are committed.
  - Rollback of the parent does not rollback the called transaction. There are no limits other than the resource limits on how many Autonomous transactions may be nested.
  - Autonomous transactions must be explicitly committed or rolled-back, otherwise an error is generated.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 36

6.7: Packages

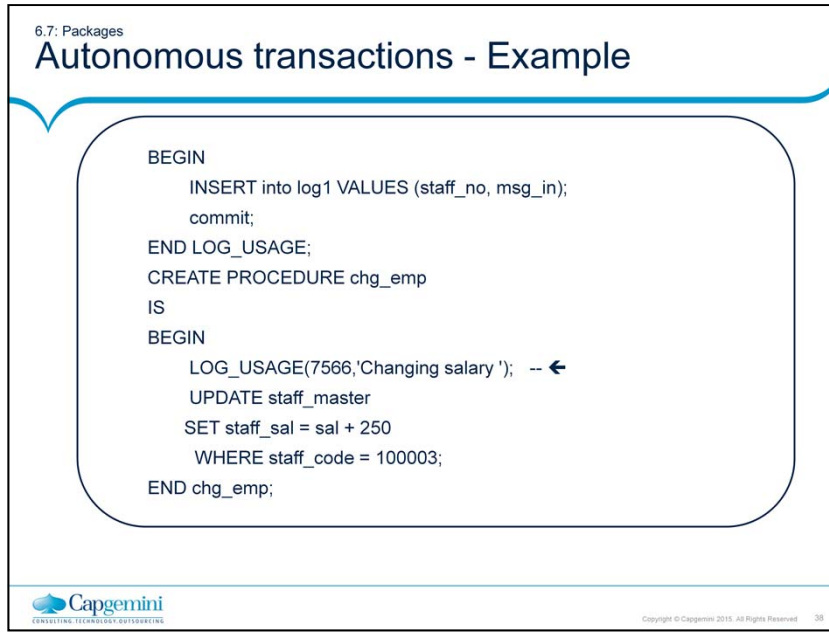
Autonomous transactions - Example

- The following example shows how to define an Autonomous block.

```
CREATE PROCEDURE  LOG_USAGE ( staff_no IN number,
                msg_in IN varchar2)
IS
PRAGMA AUTONOMOUS_TRANSACTION;
 contd.
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 37



Note:

- In the example shown in the slide, we are calling log_usage with the employee number and the appropriate message. Then we are updating the corresponding employee record.
- Irrespective of whether the update is successful or not, the insert in the log_usage procedure is always committed.

Definer's and Invoker's Rights Model:

- In case of stored procedures, functions and packages (stored subprograms), there are always two situations.
  - First situation is where a stored subprogram is created by a user.
  - Second case is when an already created stored subprogram is invoked by a privileged user of the database.

  By default whenever a subprogram is invoked, it is executed with the privileges of the creating user. This mechanism is called "**definer's rights model**".
- In "definer's rights model", if the stored subprogram (based on EMP table) is created by the user Scott, and another user (say TRG1) executes the stored subprogram, then the privileges of Scott (owner) is used in the context. In this case, even if TRG1 does not have any privileges on the table EMP (owned by Scott), he can still execute the stored subprogram and perform DML operations on the table EMP. This is because the subprogram is executed in the "definer's rights model". This model available as the default model.
- After the Oracle 8i release, the "**invoker's rights model**" can be used. In this model, the procedure executes under the privileges of the user executing the subprogram.
- In the "invokers rights model", if the user Scott creates a stored subprogram, and another user (say TRG1) executes the subprogram, then the privileges of TRG1 (the invoker) will be used in the context of the subprogram rather than the owner of subprogram (Scott).
- In this case, if the user Vivek does not have sufficient rights on the table EMP (owned by Scott), then the invocation of the subprogram will result in an appropriate error message to the invoker.
- Example: As user Scott:

```
CREATE PROCEDURE NAME_COUNT
AUTHID CURRENT_USER
IS
BEGIN
  DECLARE
      N NUMBER;
  BEGIN
  SELECT COUNT(*) INTO N FROM
SCOTT.STAFF_MASTER;
  INSERT INTO STAFFCOUNT VALUES
(SYSDATE, N);
  END;
END;
```

  - Explanation:
    - In line 2, we have defined invoker's rights.
    - In line 8, we are referring to the table EMP from Scott's schema. (This is needed, otherwise Oracle will look for EMP table in the invokers schema)
    - In line 9, the number of employees is inserted into a table empcount which is present in the current users schema.

Summary

- In this lesson, you have learnt:
  - Subprograms in PL/SQL are named PL/SQL blocks.
    - There are two types of subprograms, namely: Procedures and Functions.
  - Procedure is used to perform an action.
    - Procedures have three subprogram parameter modes, namely: IN, OUT, and INOUT.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 40

Summary

- Functions are used to compute a value.
  - A function accepts one or more parameters, and returns a single value by using a return value.
  - A function can return multiple values by using OUT parameters.
- Packages are schema objects that groups all the logically related PL/SQL types, items, and subprograms.
  - Packages usually have two parts, a specification and a body,

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 41

Review Question

- Question 1: Anonymous Blocks do not have names.
  - True / False
- Question 2: A function can return multiple values by using OUT parameters
  - True / False
- Question 3: A Package consists of "Package Specification" and "Package Body", each of them is stored in a Data Dictionary named DBMS_package.

Review Question

- Question 4: An ___ parameter returns a value to the caller of a subprogram.
- Question 5: A procedure contains two parts: ___ and ___.
- Question 6: In ___ notation, the order of the parameters is not significant.