PLSQL
Lesson 00:

IGATE is now a part of Capgemini

People matter, results count.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

©2016 Capgemini. All rights reserved.
The information contained in this document is proprietary and confidential.
For Capgemini only.

Document History

Date	Course Version No.	Software Version No.	Developer / SME	Change Record Remarks
19-July-2016	1.1		Shraddha Jadhav and Rahul Kulkarni	Integration refinements as per integrated RDBMS LOT Structure

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 2

Course Goals and Non Goals

- Course Goals
  - To understand basic PLSQL and Advanced PLSQL concepts.
- Course Non Goals
  - Nothing Specific.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 3

Intended Audience

- Software Programmers
- Software Analysts

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 4

Day Wise Schedule

- Day 1
  - Lesson 1: Need for PLSQL, Data types, Scalar and composite.
  - Lesson 2: Loop and conditional constructs
- Day 2
  - Lesson 3: Cursors, Exceptions
- Day 3
  - Lesson 3: Procedures, Functions, Packages
- Day 4
  - Lesson 3: Adv Package Concepts
  - Lesson 4: Triggers and its types
- Day 5
  - Lesson 5: Oracle Supplied packages
  - Lesson 6: Design Considerations

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 5

Day Wise Schedule

- Day 6
  - Lesson 6: Design Considerations (continued..)
  - Lesson 7: Different types of pragma
- Day 7
  - Lesson 8: Collection elements
  - Lesson 9: Oracle 11g features
- Day 8
  - Lesson 10: Best practices of PL SQL code, Dynamic SQL

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 6

Table of Contents

- Lesson 1: Need for PLSQL, Data types, Scalar and composite.
  - 1.1:Need for PLSQL
  - 1.3:Datatypes
  - 1.4:Scalar
  - 1.5:Composite
- Lesson 2: Loops and Conditional Constructs
  - 2.1. If
  - 2.2. Simple Loop
  - 2.3. For Loop
  - 2.4. While Loop

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 7

Table of Contents

- Lesson 3:Cursors, Exceptions, Procedure, Functions, Packages, Adv Packages concepts.
  - 3.1:Cursors
  - 3.2: Exceptions
  - 3.3:Procedure
  - 3.4:Functions
  - 3.5: Packages
  - 3.6: Adv Packages Concepts
- Lesson 4: Triggers and it's types
  - 4.1. Triggers
  - 4.2. Trigger types

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 8

Table of Contents

- Lesson 5: Supplied Packages
  - 5.1:DBMS_OUTPUT
  - 5.2: UTL_FILE
  - 5.3:UTL_MAIL
- Lesson 6: Design Considerations
  - 6.1. Bulk Binding concepts
  - 6.2. Ref cursors
  - 6.3. Using NOCOPY hint
  - 6.4. Using PARALLEL_ENABLE hint
  - 6.5. Using Cross-session PL/SQL function
  - 6.6. Using DETERMINISTIC clause
  - 6.7. Using returning clause

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 9

Table of Contents

- Lesson 7: Different types of pragma
  - 7.1: EXCEPTION_INIT
  - 7.2: PRAGMA AUTONOMOUS_TRANSACTION
  - 7.3: PRAGMA SERIALLY_REUSABLE
  - 7.4: PRAGMA RESTRICT_REFRENCES
- Lesson 8: PLSQL collection elements
  - 8.1. PLSQL Tables
  - 8.2. Nested tables
  - 8.3:Varrays
  - 8.4: Associative Arrays

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 10

Table of Contents

- Lesson 9: Oracle 11g features
  - 7.1: Use of sequence
  - 7.2: Compound Triggers
  - 7.3: Subprogram Inlining
  - 7.4: PL/SQL Continue statement
  - 7.5: PL/SQL Compiler Warnings
- Lesson 10: Best Practices of PLSQL and Dynamic SQL
  - 8.1. Best Practices of PLSQL code
  - 8.1.1. Conditional Compilation
  - 8.1.2: Using Selection Directives
  - 8.1.3: Obfuscation
  - 8.2: Dynamic SQL

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 11

References

- Oracle Press-Oracle Database 11g New Features by Robert G. Freeman
- Oracle Database 11g New Features for DBAs and Developers
- Oracle 11g Online documentation.
  - http://www.oracle-base.com

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 12

PLSQL

Lesson 01: PLSQL Basics, Datatypes

Lesson Objectives

- To understand the following topics:
  - Introduction and Need for PLSQL
  - Datatypes
  - Scalar
  - Composite Variables

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 2

1.1: Need for PL/SQL

Overview

- PL/SQL is a procedural extension to SQL.
- The “data manipulation” capabilities of “SQL” are combined with the “processing capabilities” of a “procedural language”.
- PL/SQL provides features like conditional execution, looping and branching.
- PL/SQL supports subroutines, as well.
- PL/SQL program is of block type, which can be “sequential” or “nested” (one inside the other).

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 3

Introduction to PL/SQL:
PL/SQL stands for Procedural Language/SQL. PL/SQL extends SQL by adding constructs found in procedural languages, resulting in a structural language that is “more powerful than SQL”.

With PL/SQL, you can use SQL statements to manipulate Oracle data and flow-of-control statements to process the data.
Moreover, you can declare constants and variables, define procedures and functions, and trap runtime errors.
Thus PL/SQL combines the “data manipulating power” of SQL with the “data processing power” of procedural languages.

PL/SQL is an “embedded language”. It was not designed to be used as a “standalone” language but instead to be invoked from within a “host” environment.

You cannot create a PL/SQL “executable” that runs all by itself.
It can run from within the database through SQL*Plus interface or from within an Oracle Developer Form (called client-side PL/SQL).

Features of PL/SQL

Tight Integration with SQL:

This integration saves both, your learning time as well as your processing time.

PL/SQL supports SQL data types, reducing the need to convert data passed between your application and database.

PL/SQL lets you use all the SQL data manipulation, cursor control, transaction control commands, as well as SQL functions, operators, and pseudo columns.

Better Performance:

Several SQL statements can be bundled together into one PL/SQL block, and sent to the server as a single unit.

This results in less network traffic and a faster application. Even when the client and the server are both running on the same machine, the performance is increased. This is because packaging SQL statements results in a simpler program that makes fewer calls to the database.

Portable:

PL/SQL is a standard and portable language.

A PL/SQL function or procedure written from within the Personal Oracle database on your laptop will run without any modification on your corporate network database. It is “Write once, run everywhere” with the only restriction being “everywhere” there is an Oracle Database.

Efficient:

Although there are a number of alternatives when it comes to writing software to run against the Oracle Database, it is easier to run highly efficient code in PL/SQL, to access the Oracle Database, than in any other language.

PL/SQL Block Structure:
PL/SQL is a block-structured language. Each basic programming unit that is written to build your application is (or should be) a “logical unit of work”. The PL/SQL block allows you to reflect that logical structure in the physical design of your programs. Each PL/SQL block has up to four different sections (some are optional under certain circumstances).

contd.

Block Types:
The basic units (procedures and functions, also known as subprograms, and anonymous blocks) that make up a PL/SQL program are “logical blocks”, which can contain any number of nested sub-blocks.
Therefore one block can represent a small part of another block, which in turn can be part of the whole unit of code.

Anonymous Blocks

Anonymous blocks are unnamed blocks. They are declared at the point in an application where they are to be executed and are passed to the PL/SQL engine for execution at runtime.

Named :

Subprograms

Subprograms are named PL/SQL blocks that can take parameters and can be invoked. You can declare them either as “procedures” or as “functions”.

Generally, you use a “procedure” to perform an “action” and a “function” to compute a “value”.

1.2: Data Types

Declaring PL/SQL variables

- Syntax

```
identifier [CONSTANT] datatype [NOT NULL]
                [:= | DEFAULT expr];
```

- Example

```
DECLARE
    v_hiredate      DATE;
    v_deptno        NUMBER(2) NOT NULL := 10;
    v_location      VARCHAR2(13) := 'Atlanta';
    c_comm CONSTANT NUMBER := 1400;
```

Capgemini CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 7

Declaring PL/SQL Variables:
You need to declare all PL/SQL identifiers within the “declaration section” before referencing them within the PL/SQL block.
You have the option to assign an initial value.

You do not need to assign a value to a variable in order to declare it.
If you refer to other variables in a declaration, you must separately declare them in a previous statement.
Syntax:

```
identifier [CONSTANT] datatype [NOT NULL]
                [:= | DEFAULT expr];
```

In the syntax given above:

identifier is the name of the variable.
CONSTANT constrains the variable so that its value cannot change. Constants must be initialized.
datatype is a scalar, composite, reference, or LOB datatype.
NOT NULL constrains the variable so that it must contain a value. NOT NULL variables must be initialized.
expr is any PL/SQL expression that can be a literal, another variable, or an expression involving operators and functions.
contd.

Base Scalar Datatypes:

NUMBER

This can hold a numeric value, either integer or floating point. It is same as the number database type.

BINARY_INTEGER

If a numeric value is not to be stored in the database, the BINARY_INTEGER datatype can be used. It can only store integers from -2147483647 to + 2147483647. It is mostly used for counter variables.

V_Counter BINARY_INTEGER DEFAULT 0;

VARCHAR2 (L)

L is necessary and is max length of the variable. This behaves like VARCHAR2 database type. The maximum length in PL/SQL is 32,767 bytes whereas VARCHAR2 database type can hold max 2000 bytes. If a VARCHAR2 PL/SQL column is more than 2000 bytes, it can only be inserted into a database column of type LONG.

CHAR (L)

Here L is the maximum length. Specifying length is optional. If not specified, the length defaults to 1. The maximum length of CHAR PL/SQL variable is 32,767 bytes, whereas the maximum length of the database CHAR column is 255 bytes. Therefore a CHAR variable of more than 255 bytes can be inserted in the database column of VARCHAR2 or LONG type.

contd.

1.2: Data Types

Base Scalar Data Types - Example

- Here are a few examples of Base Scalar Datatypes:

```
v_job          VARCHAR2(9);
v_count        BINARY_INTEGER := 0;
v_total_sal    NUMBER(9,2) := 0;
v_orderdate    DATE := SYSDATE + 7;
c_tax_rate     CONSTANT NUMBER(3,2) := 8.25;
v_valid        BOOLEAN NOT NULL := TRUE;
```

Capgemini CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 9

Base Scalar Datatypes (contd.):

LONG

PL/SQL LONG type is just 32,767 bytes. It behaves similar to LONG DATABASE type.

DATE

The DATE PL/SQL type behaves the same way as the date database type. The DATE type is used to store both date and time. A DATE variable is 7 bytes in PL/SQL.

BOOLEAN

A Boolean type variable can only have one of the two values, i.e. either TRUE or FALSE. They are mostly used in control structures.

V_Does_Dept_Exist BOOLEAN := TRUE;

V_ Flag BOOLEAN := 0; -- illegal

One more example

```
declare
        pie constant number := 7.18;
        radius number := &radius;
begin
        dbms_output.put_line('Area:
'||pie*power(radius,2));
        dbms_output.put_line('Diameter:
'||2*pie*radius);
end;
/
```

1.2: Data Types

Declaring Datatype with %TYPE Attribute

- While using the %TYPE Attribute:
  - Declare a variable according to:
  - a database column definition
  - another previously declared variable
  - Prefix %TYPE with:
  - the database table and column
  - the previously declared variable name

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 10

Reference types:
A “reference type” in PL/SQL is the same as a “pointer” in C. A “reference type” variable can point to different storage locations over the life of the program.
Using %TYPE
%TYPE is used to declare a variable with the same datatype as a column of a specific table. This datatype is particularly used when declaring variables that will hold database values.
Advantage:

You need not know the exact datatype of a column in the table in the database.
If you change database definition of a column, it changes accordingly in the PL/SQL block at run time.
Syntax:

Note: Datatype of V_Empno is same as datatype of Empno column of the EMP table.

```
Var_Name table_name.col_name%TYPE;
V_Empno   emp.empno%TYPE;
```

1.2: Data Types

Declaring Datatype with %TYPE Attribute (Contd…)

- Example:

```
...
v_name          staff_master.staff_name%TYPE;
v_balance       NUMBER(7,2);
v_min_balance   v_balance%TYPE := 10;
...
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 11

Using %TYPE (contd.)
Example

```
declare
      nSalary employee.salary%type;
begin
       select salary into nsalary
      from employee
      where emp_code = 11;
              update employee set salary = salary + 101 where
emp_code = 11;
end;
```

1.2: Data Types

Declaring Datatype by using %ROWTYPE

- Example:

```
DECLARE
    nRecord staff_master%rowtype;
BEGIN
    SELECT * into nrecord
            FROM staff_master
            WHERE staff_code = 100001;

            UPDATE staff_master
            SET staff_sal = staff_sal + 101
            WHERE emp_code = 100001;
END;
```

Capgemini CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 12

Using %ROWTYPE

%ROWTYPE is used to declare a compound variable, whose type is same as that of a row of a table.

Columns in a row and corresponding fields in record should have same names and same datatypes. However, fields in a %ROWTYPE record do not inherit constraints, such as the NOT NULL, CHECK constraints, or default values.

Syntax:

```
Var_Name table_name%ROWTYPE;
V_Emprec emp%ROWTYPE;
```

where V_Emprec is a variable, which contains within itself as many variables, whose names and datatypes match those of the EMP table. To access the Empno element of V_Emprec, use V_Emprec.empno;

For example:

```
DECLARE emprec emp%rowtype;
BEGIN
emprec.empno :=null;
emprec.deptno :=50;
dbms_output.put_line ('emprec.employee's
number'||emprec.empno);
END;
/
```

1.2: Data Types

Inserting and Updating using records

- Example:

```
DECLARE
      dept_info department_master%ROWTYPE;
BEGIN
-- dept_code, dept_name are the table columns.
-- The record picks up these names from the %ROWTYPE.
      dept_info.dept_code := 70;
      dept_info.dept_name := 'PERSONNEL';
/*Using the %ROWTYPE means we can leave out the column list (deptno,
dname) from the INSERT statement. */
       INSERT into department_master VALUES dept_info;
END;
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 13

1.2: Data Types

User-defined SUBTYPES

- User-defined SUBTYPES:
- User-defined SUBTYPES are subtypes based on an existing type.
- They can be used to give an alternate name to a type.
- Syntax:

```
SUBTYPE New_Type IS original_type;
```

```
SUBTYPE T_Counter IS NUMBER;
V_Counter T_Counter;
SUBTYPE T_Emp_Record IS  EMP%ROWTYPE;
```

- It can be a predefined type, subtype, or %type reference.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 14

User-defined SUBTYPES:
A SUBTYPE is a PL/SQL type based on an existing type. A subtype can be used to give an alternate name to a type to indicate its purpose.
A new sub_type base type can be a predefined type, subtype, or %type reference.
You can declare a dummy variable of the desired type with the constraint and use %TYPE in the SUBTYPE definition.

```
V_Dummy      NUMBER(4);
SUBTYPE  T_Counter  IS  V_Dummy%TYPE;

V_Counter          T_Counter ;
SUBTYPE  T_Numeric  IS NUMBER;

V_Counter  IS  T_Numeric(5);
```

1.2: Data Types

User-defined SUBTYPES (Contd…)

- It is illegal to constrain a subtype.

```
SUBTYPE  T_Counter  IS NUMBER(4)    -- Illegal
```

- Possible solutions:

```
V_Dummy   NUMBER(4);
SUBTYPE T_Counter   IS        V_Dummy%TYPE;
V_Counter    T_Counter ;
SUBTYPE    T_Numeric  IS      NUMBER;
V_Counter  IS  T_Numeric(5);
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 15

1.2: Data Types

Composite Data Types

- Composite Datatypes in PL/SQL:
  - Two composite datatypes are available in PL/SQL:
  - records
  - tables
- A composite type contains components within it. A variable of a composite type contains one or more scalar variables.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 16

1.2: Data Types

Record Data Types

- Record Datatype:
  - A record is a collection of individual fields that represents a row in the table.
  - They are unique and each has its own name and datatype.
  - The record as a whole does not have value.
- Defining and declaring records:
  - Define a RECORD type, then declare records of that type.
  - Define in the declarative part of any block, subprogram, or package.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 17

Record Datatype:
A record is a collection of individual fields that represents a row in the table. They are unique and each has its own name and datatype. The record as a whole does not have value. By using records you can group the data into one structure and then manipulate this structure into one “entity” or “logical unit”. This helps to reduce coding and keeps the code easier to maintain and understand.

1.2: Data Types

Record Data Types (Contd…)

- Syntax:

```
TYPE type_name IS RECORD (field_declaration [,field_
declaration] ...);
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 18

Defining and Declaring Records
To create records, you define a RECORD type, then declare records of that type. You can define RECORD types in the declarative part of any PL/SQL block, subprogram, or package by using the syntax.
where field_declaration stands for:

field_name field_type [[NOT NULL] {:= | DEFAULT} expression]
type_name is a type specifier used later to declare records. You can use %TYPE and %ROWTYPE to specify field types.

1.2: Data Types

Record Data Types - Example

- Here is an example for declaring Record datatype:

```
DECLARE
TYPE DeptRec IS RECORD (
Dept_id        department_master.dept_code%TYPE,
Dept_name              varchar2(15),
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 19

Record Datatype (contd.):
Field declarations are like variable declarations.
Each field has a unique name and specific datatype.
Record members can be accessed by using "." (Dot) notation.
The value of a record is actually a collection of values, each of which is of some simpler type. The attribute %ROWTYPE lets you declare a record that represents a row in a database table.
After a record is declared, you can reference the record members directly by using the "." (Dot) notation. You can reference the fields in a record by indicating both the record and field names.

For example: To reference an individual field, you use the dot notation DeptRec.deptno;

You can assign expressions to a record.

For example: DeptRec.deptno := 50;

You can also pass a record type variable to a procedure as shown below:

get_dept(DeptRec);

1.2: Data Types

Record Data Types - Example (Contd…)

- Here is an example for declaring and using Record datatype:

```
DECLARE
      TYPE recname is RECORD
      (customer_id number,
       customer_name varchar2(20));
      var_rec recname;
BEGIN
      var_rec.customer_id:=20;
      var_rec.customer_name:='Smith';
      dbms_output.put_line(var_rec.customer_id||'
'||var_rec.customer_name);
END;
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 20

1.2: Data Types

Table Data Type

- A PL/SQL table is:
  - a one-dimensional, unbounded, sparse collection of homogeneous elements
  - indexed by integers
  - In technical terms, a PL/SQL table:
    - is like an array
    - is like a SQL table; yet it is not precisely the same as either of those data structures
    - is one type of collection structure
    - is PL/SQL's way of providing arrays

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 21

Table Datatype

Like PL/SQL records, the table is another composite datatype. PL/SQL tables are objects of type TABLE, and look similar to database tables but with slight difference. PL/SQL tables use a primary key to give you array-like access to rows.

Like the size of the database table, the size of a PL/SQL table is unconstrained. That is, the number of rows in a PL/SQL table can dynamically increase. So your PL/QSL table grows as new rows are added. PL/SQL table can have one column and a primary key, neither of which can be named.

The column can have any datatype, but the primary key must be of the type BINARY_INTEGER.

Arrays are like temporary tables in memory. Thus they are processed very quickly. Like the size of the database table, the size of a PL/SQL table is unconstrained. The “column” can have any datatype. However, the “primary key” must be of the type BINARY_INTEGER.

1.2: Data Types

Table Data Type (Contd…)

- Declaring a PL/SQL table:
  - There are two steps to declare a PL/SQL table:
    - Declare a TABLE type.
    - Declare PL/SQL tables of that type.

```
TYPE type_name is TABLE OF
{Column_type | table.column%type} [NOT NULL]
INDEX BY BINARY_INTEGER;
```

  - If the column is defined as NOT NULL, then PL/SQL table will reject NULLs.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 22

Declaring a PL/SQL table

PL/SQL tables must be declared in two steps. First you declare a TABLE type, then declare PL/SQL tables of that type. You can declare TABLE type in the declarative part of any block, subprogram or package.

In the syntax on the above slide:

Type_name is type specifier used in subsequent declarations to define PL/SQL tables and column_name is any datatype.

You can use %TYPE attribute to specify a column datatype. If the column to which table.column refers is defined as NOT NULL, the PL/SQL table will reject NULLs.

1.2: Data Types

Table Data Type - Examples

- Example 1:
  - To create a PL/SQL table named as “student_table” of char column.

```
DECLARE
TYPE student_table is table of char(10)
INDEX BY BINARY_INTEGER;
```

- Example 2:
  - To create “student_table” based on the existing column of “student_name” of EMP table.

```
DECLARE
TYPE student_table is table of student_master.student_name%type
INDEX BY BINARY_INTEGER;
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 23

Declaring a PL/SQL table (contd.):
Example 3:
To declare a NOT NULL constraint

Note: INDEX BY BINARY INTERGER is a mandatory feature of the PL/SQL table declaration.

```
DECLARE
TYPE student_table is table of
student_master.student_name%TYPE NOT NULL
INDEX BY BINARY_INTEGER;
```

1.2: Data Types

Table Data Type - Examples (Contd…)

- After defining type emp_table, define the PL/SQL tables of that type.
  - For example:

```
Student_tab  student_table;
```

  - These tables are unconstrained tables.
  - You cannot initialize a PL/SQL table in its declaration.
    - For example:

```
Student_tab :=('SMITH','JONES','BLAKE');        --Illegal
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 24

Note:
The PL/SQL tables are unconstrained tables, because its primary key can assume any value in the range of values defined by BINARY_INTEGER.
You cannot initialize a PL/SQL table in its declaration.

1.2: Data Types

Referencing PL/SQL Tables

- Here is an example of referencing PL/SQL tables:

```
DECLARE
   TYPE staff_table is table of
    staff_master.staff_name%type
    INDEX BY BINARY_INTEGER;
staff_tab staff_table;
BEGIN
    staff_tab(1) := 'Smith'; --update Smith's salary
    UPDATE staff_master
    SET staff_sal = 1.1 * staff_sal
    WHERE staff_name = staff_tab(1);
END;
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 25

Referencing PL/SQL tables:
To reference rows in a PL/SQL table, you specify the PRIMARY KEY value using the array-like syntax as shown below:

When primary key value belongs to type BINARY_INTEGER you can reference the first row in PL/SQL table named emp_tab as shown in the slide.

PL/SQL table_name (primary key value)

1.2: Data Types

Referencing PL/SQL Tables - Examples

- To assign values to specific rows, the following syntax is used:

```
PLSQL_table_name(primary_key_value) := PLSQL expression;
```

- From ORACLE 7.3, the PL/SQL tables allow records as their columns.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 26

Referencing PL/SQL tables:
Examples:

```
type staff_rectype is record (
staff_id integer,
staff_sname varchar2(60)) ;

type staff_table is table of staff_rectype
index by binary_integer;

staff_tab      staff_table;
```

Referencing fields of record elements in PL SQL tables:

```
staff_tab(375).staff_sname := 'SMITH';
```

Summary

- In this lesson, you have learnt:
  - Introduction and Need for PLSQL
  - Datatypes
  - Scalar
  - Composite Variables

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 27

Add the notes here.

Review Question

- Question 1: User-defined SUBTYPES are subtypes based on an existing type.
  - True / False
- Question 2: A record is a collection of individual fields that represents a row in the table.
  - True/ False

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 28

Add the notes here.

Review Question

- Question 3: %ROWTYPE is used to declare a variable with the same datatype as a column of a specific table.
  - True / False
- Question 4: PL/SQL tables use a primary key to give you array-like access to rows.
  - True / False

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 29

Add the notes here.

PLSQL

Lesson 02: Loops and Conditional constructs

Lesson Objectives

- To understand the following topics:
  - Loop and conditional constructs
    - If construct
    - Simple Loop
    - For
    - While

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 2

2.1: Loops and conditional constructs

Types of Programmatic Constructs

- Programmatic Constructs are of the following types:
  - Selection structure
  - Iteration structure
  - Sequence structure

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 3

Programming Constructs:
The selection structure tests a condition, then executes one sequence of statements instead of another, depending on whether the condition is TRUE or FALSE.

A condition is any variable or expression that returns a Boolean value (TRUE or FALSE).

The iteration structure executes a sequence of statements repeatedly as long as a condition holds true.
The sequence structure simply executes a sequence of statements in the order in which they occur.

2.2: If Construct

IF - Syntax

- Given below is a list of Programmatic Constructs which are used in PL/SQL:
  - Conditional Execution:
    - This construct is used to execute a set of statements only if a particular condition is TRUE or FALSE.
    - Syntax:

```
IF Condition_Expr
THEN
    PL/SQL_Statements
END IF;
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 4

Programmatic Constructs (contd.)
Conditional Execution:
Conditional execution is of the following type:

IF-THEN-END IF
IF-THEN-ELSE-END IF
IF-THEN-ELSIF-END IF

Conditional Execution construct is used to execute a set of statements only if a particular condition is TRUE or FALSE.

2.2: If Construct

IF Construct - Example

- For example:

```
IF v_staffno = 100003
THEN
    UPDATE staff_master
    SET staff_sal = staff_sal + 100
    WHERE staff_code = 100003 ;
END IF;
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 5

Programmatic Constructs (contd.)
Conditional Execution (contd.):
As shown in the example in the slide, when the condition evaluates to TRUE, the PL/SQL statements are executed, otherwise the statement following END IF is executed.
UPDATE statement is executed only if value of v_staffno variable equals 100003.
PL/SQL allows many variations for the IF – END IF construct.

2.2: If Construct

IF Construct - Example (Contd…)

- To take alternate action if condition is FALSE, use the following syntax:

```
IF Condition_Expr THEN
    PL/SQL_Statements_1 ;
ELSE
    PL/SQL_Statements_2 ;
END IF;
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 6

Programmatic Constructs (contd.)
Conditional Execution (contd.):
Note:
When the condition evaluates to TRUE, the PL/SQL_Statements_1 is executed, otherwise PL/SQL_Statements_2 is executed.
The above syntax checks only one condition, namely Condition_Expr.

2.2: If Construct

IF Construct - Example (Contd…)

- To check for multiple conditions, use the following syntax.

```
IF Condition_Expr_1
    THEN
            PL/SQL_Statements_1 ;
    ELSIF Condition_Expr_2
    THEN
            PL/SQL_Statements_2 ;
    ELSIF Condition_Expr_3
    THEN
            PL/SQL_Statements_3 ;
            ELSE
            PL/SQL_Statements_n ;
    END IF;
```

- Note: Conditions for NULL are checked through IS NULL and IS NOT NULL predicates.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 7

Programmatic Constructs (contd.)
Conditional Execution (contd.):

```
DECLARE
    D VARCHAR2(3): = TO_CHAR(SYSDATE, 'DY')
BEGIN
    IF  D= 'SAT' THEN
        DBMS_OUTPUT.PUT_LINE('ENJOY YOUR
WEEKEND');
    ELSIF D= 'SUN' THEN
        DBMS_OUTPUT.PUT_LINE('ENJOY YOUR
WEEKEND');
    ELSE
        DBMS_OUTPUT.PUT_LINE('HAVE A NICE DAY');
    END IF;
END;
```

```
IF Condition_Expr_1 THEN

PL/SQL_Statements_1 ;
ELSIF Condition_Expr_2 THEN

PL/SQL_Statements_2 ;
ELSIF Condition_Expr_3 THEN
        Null;
END IF;
```

Programmatic Constructs (contd.)
Conditional Execution (contd.):
As every condition must have at least one statement, NULL statement can be used as filler.
NULL command does nothing.
Sometimes NULL is used in a condition merely to indicate that such a condition has been taken into consideration, as well. So your code will resemble the code as given below:

Conditions for NULL are checked through IS NULL and IS NOT NULL predicates.

2.2: Loop

Simple Loop - Syntax

- Looping
  - A LOOP is used to execute a set of statements more than once.
  - Syntax:

```
LOOP
    PL/SQL_Statements;
END LOOP ;
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 9

2.2: Loop

Simple Loop (Contd…)

- For example:

```
DECLARE
    v_counter  number := 50 ;
BEGIN
LOOP
    INSERT INTO department_master
            VALUES(v_counter,'new dept');
    v_counter := v_counter + 10 ;
END LOOP;
    COMMIT ;
END ;
/
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 10

Programmatic Constructs (contd.)
Looping
The example shown in the slide is an endless loop.
When LOOP ENDLOOP is used in the above format, then an exit path must necessarily be provided. This is discussed in the following slide.

2.2: Loop

Simple Loop – EXIT statement

- EXIT
  - Exit path is provided by using EXIT or EXIT WHEN commands.
  - EXIT is an unconditional exit. Control is transferred to the statement following END LOOP, when the execution flow reaches the EXIT statement.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 11

2.2: Loop

Simple Loop – EXIT statement (Contd…)

- Syntax:

```
BEGIN
    .....
    .....
    LOOP                                    IF <Condition> THEN

            . . . . .
            EXIT ;              -- Exits loop immediately
        END IF ;
    END LOOP;
    LOOP
        ..........
        ..........
        EXIT WHEN <condition>
    END LOOP;

    . . . . .                   -- Control resumes here
    COMMIT ;
    END ;
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 12

Note:
EXIT WHEN is used for conditional exit out of the loop.

2.2: Loop

Simple Loop – EXIT statement (Contd…)

- For example:

```
DECLARE
      v_counter number := 50 ;
BEGIN
      LOOP
         INSERT INTO department_master
        VALUES(v_counter,'NEWDEPT');
      DELETE  FROM emp WHERE deptno = v_counter;
                  v_counter := v_counter + 10 ;
                                          EXIT WHEN v_counter
>100 ;
      END LOOP;
      COMMIT ;
END ;
```

- Note: As long as v_counter has a value less than or equal to 100, the loop continues.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 13

Note:
LOOP.. END LOOP can be used in conjunction with FOR and WHILE for better control on looping.

2.3: For Loop

For - Syntax

- FOR Loop:
- Syntax:

```
FOR Variable IN [REVERSE] Lower_Bound..Upper_Bound
LOOP
    PL/SQL_Statements
END LOOP ;
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 14

Programmatic Constructs (contd.)
FOR Loop:
FOR loop is used for executing the loop a fixed number of times. The number of times the loop will execute equals the following:

Upper_Bound - Lower_Bound + 1.

Upper_Bound and Lower_Bound must be integers.
Upper_Bound must be equal to or greater than Lower_Bound.
Variables in FOR loop need not be explicitly declared.

Variables take values starting at a Lower_Bound and ending at a Upper_Bound.
The variable value is incremented by 1, every time the loop reaches the bottom.
When the variable value becomes equal to the Upper_Bound, then the loop executes and exits.

When REVERSE is used, then the variable takes values starting at Upper_Bound and ending at Lower_Bound.
Value of the variable is decremented each time the loop reaches the bottom.

2.3: For Loop

For Loop - Example

- For Example:

```
DECLARE
v_counter number := 50 ;
BEGIN
      FOR  Loop_Counter  IN 2..5
      LOOP
      INSERT INTO dept
      VALUES(v_counter ,'NEW DEPT') ;
       v_counter := v_counter + 10 ;
      END LOOP;
      COMMIT ;
END ;
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 15

Programmatic Constructs (contd.)
In the example in the above slide, the loop will be executed (5 - 2 + 1) = 4 times.
A Loop_Counter variable can also be used inside the loop, if required.
Lower_Bound and/or Upper_Bound can be integer expressions, as well.

2.3: While Loop

While Loop - Syntax

- WHILE Loop
- The WHILE loop is used as shown below.
  - Syntax:

```
WHILE Condition
LOOP
    PL/SQL  Statements;
END LOOP;
```

- EXIT OR EXIT WHEN can be used inside the WHILE loop to prematurely exit the loop.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 16

Programmatic Constructs (contd.)
WHILE Loop:
Example:

```
DECLARE
    ctr number := 1;
BEGIN
    WHILE ctr <= 10
    LOOP
    dbms_output.put_line(ctr);
        ctr := ctr+1;
    END LOOP;
END;
/
```

2.4: Labeling Loops

Labeling of Loops

- Labeling of Loops:
  - The label can be used with the EXIT statement to exit out of a particular loop.

```
BEGIN
    <<Outer_Loop>>
    LOOP
            PL/SQL
            << Inner_Loop>>
            LOOP
                    PL/SQL  Statements ;
                    EXIT Outer_Loop WHEN <Condition Met>
            END LOOP  Inner_Loop
    END LOOP  Outer_Loop
END ;
```

Capgemini CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 17

Programmatic Constructs (contd.)
Labeling of Loops:
Loops themselves can be labeled as in the case of blocks.
The label can be used with the EXIT statement to exit out of a particular loop.

Summary

- In this lesson, you have learnt:
  - Different programmatic constructs in PL/SQL are
  - Selection structure,
  - Iteration structure,
  - Sequence structure

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 18

Add the notes here.

Review Question

- Question 1: While using FOR loop, Upper_Bound, and Lower_Bound must be integers.
  - True / False
- Question 2: ____ is used to exit out of loop.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 19

Add the notes here.

PLSQL

Lesson 03: Cursors, Exception handling, Procedures, Functions, Packages, Adv Packages concepts.

Lesson Objectives

- To understand the following topics:
  - Introduction to Cursors
  - Implicit and Explicit Cursors
  - Cursor attributes
  - Processing Implicit Cursors and Explicit Cursors
  - Error Handling
    - Predefined Exceptions
    - Numbered Exceptions
    - User Defined Exceptions
  - Raising Exceptions
  - Control passing to Exception Handlers

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 2

Lesson Objectives

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

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 3

3.1: Cursors

Concept

- A cursor is a “handle” or “name” for a private SQL area.
- An SQL area (context area) is an area in the memory in which a parsed statement and other information for processing the statement are kept.
- PL/SQL implicitly declares a cursor for all SQL data manipulation statements, including queries that return “only one row”.
- For queries that return “more than one row”, you must declare an explicit cursor.
  - Thus the two types of cursors are:
    - implicit
    - explicit

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 4

Introduction to Cursors:
ORACLE allocates memory on the Oracle server to process SQL statements. It is called as “context area”. Context area stores information like number of rows processed, the set of rows returned by a query, etc.
A Cursor is a “handle” or “pointer” to the context area. Using this cursor the PL/SQL program can control the context area, and thereby access the information stored in it.
There are two types of cursors - “explicit” and “implicit”.

In an explicit cursor, a cursor name is explicitly assigned to a SELECT statement through CURSOR IS statement.

An implicit cursor is used for all other SQL statements.

Processing an explicit cursor involves four steps. In case of implicit cursors, the PL/SQL engine automatically takes care of these four steps.

3.1: Cursors

Concept

- Implicit Cursor:
  - The PL/SQL engine takes care of automatic processing.
  - PL/SQL implicitly declares cursors for all DML statements.
  - They are simple SELECT statements and are written in the BEGIN block (executable section) of the PL/SQL.
  - They are easy to code, and they retrieve exactly one row

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 5

3.1: Cursors

Implicit Cursors

- Processing Implicit Cursors:
  - Oracle implicitly opens a cursor to process each SQL statement that is not associated with an explicitly declared cursor.
  - This implicit cursor is known as SQL cursor.
    - Program cannot use the OPEN, FETCH, and CLOSE statements to control the SQL cursor. PL/SQL implicitly does those operations .
    - You can use cursor attributes to get information about the most recently executed SQL statement.
    - Implicit Cursor is used to process INSERT, UPDATE, DELETE, and single row SELECT INTO statements.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 6

Processing Implicit Cursors:
All SQL statements are executed inside a context area and have a cursor, which points to that context area. This implicit cursor is known as SQL cursor.
Implicit SQL cursor is not opened or closed by the program. PL/SQL implicitly opens the cursor, processes the SQL statement, and closes the cursor.
Implicit cursor is used to process INSERT, UPDATE, DELETE, and single row SELECT INTO statements.
The cursor attributes can be applied to the SQL cursor.

3.1: Cursors

Implicit Cursors - Example

```
BEGIN
   UPDATE dept SET dname ='Production' WHERE deptno= 50;
   IF SQL%NOTFOUND THEN
     INSERT into department_master VALUES ( 50, 'Production');
   END IF;
END;
```

```
BEGIN
  UPDATE dept SET dname ='Production' WHERE deptno = 50;
  IF SQL%ROWCOUNT = 0 THEN
    INSERT into  department_master VALUES ( 50, 'Production');
END IF;
END;
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 7

Processing Implicit Cursors:
Note:
SQL%NOTFOUND should not be used with SELECT INTO …. Statement.
This is because SELECT INTO…. Statement will raise an ORACLE error if no rows are selected, and

control will pass to exception * section (discussed later), and
SQL%NOTFOUND statement will not be executed at all

The slide shows two code snippets using Cursor attributes SQL%NOTFOUND and SQL%ROWCOUNT respectively.

3.1: Cursors

Explicit Cursors

- Explicit Cursor:
  - The set of rows returned by a query can consist of zero, one, or multiple rows, depending on how many rows meet your search criteria.
  - When a query returns multiple rows, you can explicitly declare a cursor to process the rows.
  - You can declare a cursor in the declarative part of any PL/SQL block, subprogram, or package.
  - Processing has to be done by the user.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 8

Explicit Cursor:
When you need precise control over query processing, you can explicitly declare a cursor in the declarative part of any PL/SQL block, subprogram, or package.
This technique requires more code than other techniques such as the implicit cursor FOR loop. But it is beneficial in terms of flexibility. You can:

Process several queries in parallel by declaring and opening multiple cursors.

Process multiple rows in a single loop iteration, skip rows, or split the processing into more than one loop.

3.1: Cursors

Processing Explicit Cursors

- While processing Explicit Cursors you have to perform the following four steps:
  - Declare the cursor
  - Open the cursor for a query
  - Fetch the results into PL/SQL variables
  - Close the cursor

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 9

3.1: Cursors

Processing Explicit Cursors

- Declaring a Cursor:
- Syntax:

```
CURSOR Cursor_Name IS Select_Statement;
```

- Any SELECT statements are legal including JOINS, UNION, and MINUS clauses.
- SELECT statement should not have an INTO clause.
- Cursor declaration can reference PL/SQL variables in the WHERE clause.
- The variables (bind variables) used in the WHERE clause must be visible at the point of the cursor.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 10

Processing Explicit Cursors: Declaring a Cursor:
The code snippets on the slide show the usage of variables in cursor declaration. You cannot use a variable before it has been declared. It will be illegal.

3.1: Cursors

Processing Explicit Cursors

- Opening a Cursor
  - Syntax:

```
OPEN  Cursor_Name;
```

  - When a cursor is opened, the following events occur:
  - The values of bind variables are examined.
  - The active result set is determined.
  - The active result set pointer is set to the first row.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 12

Processing Explicit Cursors: Opening a Cursor:

When a Cursor is opened, the following events occur:

The values of “bind variables” are examined.

Based on the values of bind variables , the “active result set” is determined.

The active result set pointer is set to the first row.

“Bind variables” are evaluated only once at Cursor open time.

Changing the value of Bind variables after the Cursor is opened will not make any changes to the active result set.

The query will see changes made to the database that have been committed prior to the OPEN statement.

You can open more than one Cursor at a time.

3.1: Cursors

Processing Explicit Cursors

- Fetching from a Cursor
  - Syntax:

```
FETCH Cursor_Name  INTO List_Of_Variables;
FETCH  Cursor_Name INTO PL/SQL _Record;
```

- The “list of variables” in the INTO clause should match the “column names list” in the SELECT clause of the CURSOR declaration, both in terms of count as well as in datatype.
- After each FETCH, the active set pointer is increased to point to the next row.
- The end of the active set can be found out by using %NOTFOUND attribute of the cursor.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 13

Processing Explicit Cursors: Fetching from Cursor:

Processing Explicit Cursor

- Fetching Data

```
DECLARE
 v_deptno    department_master.dept_code%type;
 v_dept      department_master%rowtype;
    CURSOR c_alldept IS  SELECT * FROM
    department_master;
BEGIN
    OPEN    c_alldept;
    FETCH  c_Alldept INTO V_Dept;
```

Legal Fetch

```
    FETCH c_alldept INTO V_Deptno;
    END;
```

Illegal Fetch

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 14

Processing Explicit Cursors: Fetching from Cursor:
The code snippets on the slide shows example of fetching data from cursor. The second snippet FETCH is illegal since SELECT * selects all columns of the table, and there is only one variable in INTO list.
For each column value returned by the query associated with the cursor, there must be a corresponding, type-compatible variable in the INTO list.

To change the result set or the values of variables in the query, you must close and reopen the cursor with the input variables set to their new values.

3.1: Cursors

Processing Explicit Cursors

- Closing a Cursor
  - Syntax

```
CLOSE  Cursor_Name;
```

  - Closing a Cursor frees the resources associated with the Cursor.
  - You cannot FETCH from a closed Cursor.
  - You cannot close an already closed Cursor.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 15

3.1: Cursors

Attributes

- Cursor Attributes:
  - Explicit cursor attributes return information about the execution of a multi-row query.
  - When an “Explicit cursor” or a “cursor variable” is opened, the rows that satisfy the associated query are identified and form the result set.
  - Rows are fetched from the result set.
  - Examples: %ISOPEN, %FOUND, %NOTFOUND, %ROWCOUNT, etc.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 16

3.1: Cursors

Types of Cursor Attributes

- The different types of cursor attributes are described in brief, as follows:
- %ISOPEN
- %ISOPEN returns TRUE if its cursor or cursor variable is open. Otherwise it returns FALSE.
- Syntax:

```
Cur_Name%ISOPEN
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 17

Cursor Attributes: %ISOPEN
This attribute is used to check the open/close status of a Cursor.
If the Cursor is already open, the attribute returns TRUE.
Oracle closes the SQL cursor automatically after executing its associated SQL statement. As a result, %ISOPEN always yields FALSE for Implicit cursor.

3.1: Cursors

Types of Cursor Attributes

- Example:

```
DECLARE
            cursor c1 is
        select_statement ;
BEGIN
    IF c1%ISOPEN THEN
            pl/sql_statements ;
    END IF;
END ;
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 18

Cursor Attributes: %ISOPEN (contd.)
Note:
In the example shown in the slide, C1%ISOPEN returns FALSE as the cursor is yet to be opened.
Hence, the PL/SQL statements within the IF...END IF are not executed.

3.1: Cursors

Types of Cursor Attributes

- %FOUND
- %FOUND yields NULL after a cursor or cursor variable is opened but before the first fetch.
- Thereafter, it yields:
- TRUE if the last fetch has returned a row, or
- FALSE if the last fetch has failed to return a row
- Syntax:

```
cur_Name%FOUND
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 19

Cursor Attributes: %FOUND:
Until a SQL data manipulation statement is executed, %FOUND yields NULL. Thereafter, %FOUND yields TRUE if an INSERT, UPDATE, or DELETE statement affected one or more rows, or a SELECT INTO statement returned one or more rows. Otherwise, %FOUND yields FALSE

3.1: Cursors

Types of Cursor Attributes

- Example:

```
DECLARE section;
   open c1 ;
   fetch c1 into var_list ;
IF c1%FOUND THEN
      pl/sql_statements ;
END IF ;
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 20

3.1: Cursors

Types of Cursor Attributes

- %NOTFOUND
  - %NOTFOUND is the logical opposite of %FOUND.
  - %NOTFOUND yields:
    - FALSE if the last fetch has returned a row, or
    - TRUE if the last fetch has failed to return a row
  - It is mostly used as an exit condition.
  - Syntax:

```
cur_Name%NOTFOUND
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 21

Cursor Attributes: %NOTFOUND:
%NOTFOUND is the logical opposite of %FOUND. %NOTFOUND yields TRUE if an INSERT, UPDATE, or DELETE statement affected no rows, or a SELECT INTO statement returned no rows. Otherwise, %NOTFOUND yields FALSE.

3.1: Cursors

Types of Cursor Attributes

- %ROWCOUNT
  - %ROWCOUNT returns number of rows fetched from the cursor area using FETCH command.
  - %ROWCOUNT is zeroed when its cursor or cursor variable is opened.
    - Before the first fetch, %ROWCOUNT yields 0.
    - Thereafter, it yields the number of rows fetched at that point of time.
  - The number is incremented if the last FETCH has returned a row.
  - Syntax:

```
cur_Name%NOTFOUND
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 22

Cursor Attributes: %ROWCOUNT
For example: To give a 10% raise to all employees earning less than Rs. 2500.

```
DECLARE
        V_Salary   emp.sal%TYPE;
        V_Empno  emp.empno%TYPE;
        CURSOR C_Empsal  IS
        SELECT empno, sal FROM emp WHERE sal
        <2500;
BEGIN
        IF  NOT C_Empsal%ISOPEN THEN
        OPEN  C_Empsal  ;
        END  IF ;
        LOOP
        FETCH  C_Empsal  INTO
        V_Empno,V_Salary;
        EXIT WHEN  C_Empsal %NOTFOUND ;
                --Exit out of block when no rows
        UPDATE emp SET sal = 1.1 * V_Salary
        WHERE empno = V_Empno;
        END LOOP ;
        CLOSE  C_Empsal ;
        COMMIT ;
END ;
```

3.1: Cursors

Cursor FETCH loops

- They are examples of simple loop statements.
- The FETCH statement should be followed by the EXIT condition to avoid infinite looping.
- Condition to be checked is cursor%NOTFOUND.
- Examples: LOOP .. END LOOP, WHILE LOOP, etc

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 23

3.1: Cursors

Cursor using LOOP … END LOOP:

```
DECLARE
     cursor c1 is ……..
BEGIN
    open cursor c1;  /* open the cursor and identify the active result set.*/
LOOP
    fetch c1 into  variable_list ;
    -- exit out of the loop when there are no more rows.
    /* exit is done before processing to prevent handling of null rows.*/
    EXIT WHEN C1%NOTFOUND ;
    /* Process the fetched rows using variables and  PL/SQLstatements */
END LOOP;
    -- Free resources used by the cursor
    close  c1;
    -- commit
    commit;
END;
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 24

3.1: Cursors

Cursor using LOOP … END LOOP:

- There should be a FETCH statement before the WHILE statement to enter into the loop.
- The EXIT condition should be checked as cursorname%FOUND.
- Syntax:

```
DECLARE
    cursor c1  is ……..
BEGIN
    open cursor  c1 ; /* open the cursor and identify the active
result set.*/
    -- retrieve the first row to set  up the  while loop
    FETCH C1 INTO  VARIABLE_LIST ;
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 25

Processing Explicit Cursors: Cursor using WHILE loops:
Note:
FETCH statement appears twice, once before the loop and once after the loop processing.
This is necessary so that the loop condition will be evaluated for each iteration of the loop.

Cursor using WHILE loops…contd

```
/*Continue  looping ,  processing & fetching till last row is
retrieved.*/
WHILE C1%FOUND
LOOP
      fetch c1 into variable_list;
END LOOP;
   CLOSE C1; -- Free resources used by the cursor.
   commit;   -- commit
END;
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 26

3.1: Cursors

FOR Cursor LOOP

- FOR Cursor Loop

```
FOR Variable in Cursor_Name
    LOOP
            Process the variables
    END LOOP;
```

- You can pass parameters to the cursor in a CURSOR FOR loop.

```
FOR Variable in Cursor_Name ( PARAM1 , PARAM 2 ….)
    LOOP
            Process the variables
    END LOOP;
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 27

Processing Explicit Cursors: FOR CURSOR Loop:
For all other loops we had to go through all the steps of OPEN, FETCH, AND CLOSE statements for a cursor.
PL/SQL provides a shortcut via a CURSOR FOR Loop. The CURSOR FOR Loop implicitly handles the cursor processing.

```
DECLARE
CURSOR  C1  IS ……..
BEGIN
-- An implict Open  of the  cursor C1 is done here.
-- Record variable should not be declared in
declaration section
FOR  Record_Variable   IN  C1 LOOP
-- An implicit Fetch is done here
-- Process the fetched rows using variables  and
PL/SQL statements
-- Before the loop is continued an implicit
C1%NOTFOUND test is done by PL/SQL
END LOOP;
-- An automatic close  C1 is issued after termination of
the loop
-- Commit
COMMIT ;
END;
/
```

In this snippet, the record variable is implicitly declared by PL/SQL and is of the type C1%ROWTYPE and the scope of Record_Variable is only for the cursor FOR LOOP.

Explicit Cursor - Examples

- Example 1: To add 10 marks in subject3 for all students

```
DECLARE
      v_sno      student_marks.student_code%type;
      cursor c_stud_marks is
     select student_code from student_marks;
BEGIN
      OPEN c_stud_marks;
      FETCH c_stud_marks into v_sno;
      WHILE c_stud_marks%found
        LOOP
          UPDATE student_marks SET subject3 =subject3+10
         WHERE student_code = v_sno ;
   FETCH c_stud_marks into v_empno;
   END LOOP ;
   CLOSE c_stud_marks;
END;
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 28

Explicit Cursor - Examples

- Example 2: The block calculates the total marks of each student for all the subjects. If total marks are greater than 220 then it will insert that student detail in “Performance” table

```
DECLARE
    cursor c_stud_marks is select * from student_marks;
    total_marks number(4);
BEGIN
   FOR mks in c_stud_marks
LOOP
   total_marks:=mks.subject1+mks.subject2+mks.subject3;
IF (total_marks >220) THEN
   INSERT into performance
     VALUES (mks.student_code,total_marks);
END IF;
END LOOP;
END;
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 29

In the above example “Performance” is a user defined table.

3.1: Cursors

SELECT… FOR UPDATE

- SELECT … FOR UPDATE cursor:
  - The method of locking records which are selected for modification, consists of two parts:
  - The FOR UPDATE clause in CURSOR declaration.
  - The WHERE CURRENT OF clause in an UPDATE or DELETE statement.
  - Syntax: FOR UPDATE

```
CURSOR   Cursor_Name IS  SELECT  ….. FROM … WHERE   .. ORDER BY
FOR UPDATE  [OF column names ] [ NOWAIT]
```

- where column names are the names of the columns in the table
- against which the query is fired. The column names are optional.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 30

Processing Explicit Cursors: Using FOR UPDATE

The FOR UPDATE clause is part of the SELECT statement of the cursor. It is the last clause of the SELECT statement after the ORDER BY clause (if present). Normally, a SELECT operation does not lock the rows being accessed. This allows others to change the data selected by the SELECT statement. Besides, at OPEN time the active set consists of changes which were committed. Any changes made after OPEN even if they are committed, are not reflected in the active result set unless the cursor is reopened. This results in Data Inconsistency.

If the FOR UPDATE clause is present, exclusive “row locks” are taken on the rows in the active set.

These locks prevent other sessions / users from changing these rows unless the changes are committed.

If another session / user already has locks on the rows in the active set, then the SELECT FOR UPDATE will wait indefinitely for these locks to be released. This statement will hang the system till the locks are released.

To avoid this NOWAIT clause is used.

With the NOWAIT clause SELECT FOR UPDATE will not wait for the locks acquired by previous sessions to be released and will return immediately with an ORACLE error.

3.1: Cursors

SELECT… FOR UPDATE

- If the cursor is declared with a FOR UPDATE clause, the WHERE CURRENT OF clause can be used in an UPDATE or DELETE statement.
- Syntax: WHERE CURRENT OF

```
WHERE CURRENT OF Cursor_Name
```

- The WHERE CURRENT OF clause evaluates up to the row that was just retrieved by the cursor.
- When querying multiple tables Rows in a table are locked only if the FOR UPDATE OF clause refers to a column in that table.

contd.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 31

Processing Cursors: Using WHERE CURRENT OF:
Note:
When querying multiple tables you can use the FOR UPDATE OF column to confine row locking for a particular table.

3.1: Cursors

SELECT… FOR UPDATE

- For example: Following query locks the staff_master table but not the department_master table.

```
CURSOR C1 is SELECT staff_code, job, dname from emp,
dept WHERE emp.deptno=dept.deptno FOR UPDATE OF sal;
```

- Using primary key simulates the WHERE CURRENT OF clause but does not create any locks.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 32

Processing Cursors: Using WHERE CURRENT OF (contd.):
Note:
If you are using NOWAIT clause, then OF Column_List is essential for syntax purpose.
Any COMMIT should be done after the processing is over in a CURSOR LOOP which has FOR UPDATE clause. This is because after commit locks will be released, the cursor will become invalid, and further FETCH will result in an error. This problem can be solved by using primary key. Using primary key simulates the WHERE CURRENT of clause, however it does not create any locks.

3.1: Cursors

SELECT… FOR UPDATE - Examples

- To promote professors who earn more than 20000

```
DECLARE
CURSOR c_staff  is SELECT staff_code, staff_master.design_code
 FROM staff_master,designation_master
 WHERE design_name = 'Professor' and staff_sal > 20000
and staff_master.design_code =designation_master.design_code
FOR UPDATE OF design_code NOWAIT;
d_code  designation_master.design_code%type;
BEGIN
    SELECT design_code into d_code FROM designation_master
        WHERE design_name='Director';
      FOR v_rec in c_staff
      LOOP
          UPDATE staff_master SET design_code = d_code
           WHERE current of c_staff;
      END LOOP;
 END;
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 33

3.2: Exception Handling

Understanding Exception Handling in PL/SQL

- Error Handling:
  - In PL/SQL, a warning or error condition is called an “exception”.
    - Exceptions can be internally defined (by the run-time system) or user defined.
    - Examples of internally defined exceptions:
  - division by zero
  - out of memory
    - Some common internal exceptions have predefined names, namely:
  - ZERO_DIVIDE
  - STORAGE_ERROR

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 34

Error Handling:
A good programming language should provide capabilities of handling errors and recovering from them if possible.
PL/SQL implements Error Handling via “exceptions” and “exception handlers”.
Types of Errors in PL/SQL
Compile Time errors: They are reported by the PL/SQL compiler, and you have to correct them before recompiling.
Run Time errors: They are reported by the run-time engine. They are handled programmatically by raising an exception, and catching it in the Exception section.

3.2: Exception Handling

Understanding Exception Handling in PL/SQL

- The other exceptions can be given user-defined names.
- Exceptions can be defined in the declarative part of any PL/SQL block, subprogram, or package. These are user-defined exceptions.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 35

3.2: Exception Handling

Declaring Exception

- Exception is an error that is defined by the program.
  - It could be an error with the data, as well.
- There are three types of exceptions in Oracle:
  - Predefined exceptions
  - Numbered exceptions
  - User defined exceptions

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 36

Declaring Exceptions:
Exceptions are declared in the Declaration section, raised in the Executable section, and handled in the Exception section.

3.2: Exception Handling

Predefined Exception

- Predefined Exceptions correspond to the most common Oracle errors.
  - They are always available to the program. Hence there is no need to declare them.
  - They are automatically raised by ORACLE whenever that particular error condition occurs.
  - Examples: NO_DATA_FOUND,CURSOR_ALREADY_OPEN, PROGRAM_ERROR

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 37

Predefined Exceptions:
An internal exception is raised implicitly whenever your PL/SQL program violates an Oracle rule or exceeds a system-dependent limit. Every Oracle error has a number, but exceptions must be handled by name. So, PL/SQL predefines some common Oracle errors as exceptions. For example, PL/SQL raises the predefined exception NO_DATA_FOUND if a SELECT INTO statement returns no rows.
Given below are some Predefined Exceptions:

NO_DATA_FOUND
This exception is raised when SELECT INTO …. statement does not return any rows.

TOO_MANY_ROWS
This exception is raised when SELECT INTO …. statement returns more than one row.

INVALID_CURSOR
This exception is raised when an illegal cursor operation is performed such as closing an already closed cursor.

VALUE_ERROR
This exception is raised when an arithmetic, conversion, truncation, or constraint error occurs in a procedural statement.

DUP_VAL_ON_INDEX
This exception is raised when the UNIQUE CONSTRAINT is violated.

3.2: Exception Handling

Predefined Exception - Example

- In the following example, the built in exception is handled.

```
DECLARE
        v_staffno   staff_master.staff_code%type;
        v_name      staff_master.staff_name%type;
BEGIN
        SELECT staff_name into v_name FROM staff_master
        WHERE staff_code=&v_staffno;
        dbms_output.put_line(v_name);
EXCEPTION
        WHEN  NO_DATA_FOUND THEN
        dbms_output.put_line('Not Found');
END;
/
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 38

Predefined Exceptions:
In the example shown on the slide, the NO_DATA_FOUND built in exception is handled. It is automatically raised if the SELECT statement does not fetch any value and populate the variable.

3.2: Exception Handling

Numbered Exception

- An exception name can be associated with an ORACLE error.
  - This gives us the ability to trap the error specifically to ORACLE errors
  - This is done with the help of “compiler directives” –
  - PRAGMA EXCEPTION_INIT

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 39

Numbered Exception:
The Numbered Exceptions are Oracle errors bound to a user defined exception name.

3.2: Exception Handling

Numbered Exception

- PRAGMA EXCEPTION_INIT:
  - A PRAGMA is a compiler directive that is processed at compile time, not at run time. It is used to name an exception.
  - In PL/SQL, the PRAGMA EXCEPTION_INIT tells the compiler to associate an exception name with an Oracle error number.
  - This arrangement lets you refer to any internal exception(error) by name, and to write a specific handler for it.
  - When you see an error stack, or sequence of error messages, the one on top is the one that you can trap and handle.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 40

3.2: Exception Handling

Numbered Exception (Contd.)

- User defined exceptions can be named with error number between -20000 and -20999.
- The naming is declared in Declaration section.
- It is valid within the PL/SQL blocks only.
- Syntax is:

```
PRAGMA EXCEPTION_INIT(Exception Name,Error_Number);
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 41

3.2: Exception Handling

Numbered Exception - Example

- A PL/SQL block to handle Numbered Exceptions

```
DECLARE
        v_bookno number := 10000008;
       child_rec_found EXCEPTION;
       PRAGMA EXCEPTION_INIT (child_rec_found, -2292);
BEGIN
            DELETE from book_master
          WHERE book_code = v_bookno;
EXCEPTION
            WHEN child_rec_found THEN
      INSERT into error_log
            VALUES ('Book entries exist for book:' || v_bookno);
END;
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 42

If a user tries to delete record from the parent table wherein child records exist an error is raised by Oracle. We would want to handle this error through the PL/SQL block which is deleting records from a parent table. The example on the slide demonstrates this. In the PL/SQL block we are binding the constraint exception raised by Oracle to user defined exception name.
All oracle errors are negative i.e prefixed with a minus symbol.
In the example we are mapping error-2292 which occurs when referential integrity rule is violated.

3.2: Exception Handling

User-defined Exception

- User-defined Exceptions are:
  - declared in the Declaration section,
  - raised in the Executable section, and
  - handled in the Exception section

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 43

User-Defined Exceptions:
These exception are entirely user defined based on the application. The programmer is responsible for declaring, raising and handling them.

3.2: Exception Handling

User-defined Exception - Example

- Here is an example of User Defined Exception:

```
DECLARE
    E_Balance_Not_Sufficient EXCEPTION;
    E_Comm_Too_Larage EXCEPTION;
    …
BEGIN
    NULL;
END;
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 44

3.2: Exception Handling

Raising Exceptions

- Raising Exceptions:
- Internal exceptions are raised implicitly by the run-time system, as are user-defined exceptions that are associated with an Oracle error number using EXCEPTION_INIT.
- Other user-defined exceptions must be raised explicitly by RAISE statements.
- The syntax is:

```
RAISE  Exception_Name;
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 45

Raising Exceptions:
When the error associated with an exception occurs, the exception is raised.
This is done through the RAISE command.

3.2: Exception Handling

Raising Exceptions - Example

- An exception is defined and raised as shown below:

```
DECLARE
    ...
    retired_emp EXCEPTION ;
BEGIN
    pl/sql_statements ;
    if error_condition then
    RAISE retired_emp ;
    pl/sql_statements ;
EXCEPTION
    WHEN retired_emp THEN
    pl/sql_statements ;
END ;
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 45

Control passing to Exception Handler

- Control passing to Exception Handler :
  - When an exception is raised, normal execution of your PL/SQL block or subprogram stops, and control passes to its exception-handling part.
  - To catch the raised exceptions, you write “exception handlers”.
  - Each exception handler consists of a WHEN clause, which specifies an exception, followed by a sequence of statements to be executed when that exception is raised.
  - These statements complete execution of the block or subprogram, however, the control does not return to where the exception was raised. In other words, you cannot resume processing where you left off.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 47

Control passing to Exception Handler:

```
DECLARE
Retired_Emp
EXCEPTION;
BEGIN
...
RAISE Retired_Emp;
...
EXCEPTION
WHEN Retired_Emp
THEN
...
END;
```

EXCEPTION Retired_Emp is declared here.

EXCEPTION Retired_Emp is raised here.

Any code here is not executed.

Control passes to the Exception Handler. Code here will be executed.

3.2: Exception Handling

User-defined Exception - Example

- User Defined Exception Handling:

```
DECLARE
    dup_deptno EXCEPTION;
    v_counter binary_integer;
    v_department number(2) := 50;
BEGIN
    SELECT count(*) into v_counter  FROM department_master
    WHERE dept_code=50;
  IF v_counter > 0 THEN
            RAISE dup_deptno ;
  END IF;
        INSERT into department_master
        VALUES (v_department ,'new name');
EXCEPTION
          WHEN dup_deptno THEN
            INSERT into error_log
            VALUES ('Dept: '|| v_department ||' already exists");
   END ;
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 48

The example on the slide demonstrates user-defined exceptions. It checks for department no value to be inserted in the table. If the value is duplicated it will raise an exception.

3.2: Exception Handling

Others Exception Handler

- OTHERS Exception Handler:
  - The optional OTHERS exception handler, which is always the last handler in a block or subprogram, acts as the handler for all exceptions that are not specifically named in the Exception section.
  - A block or subprogram can have only one OTHERS handler.
  - To handle a specific case within the OTHERS handler, predefined functions SQLCODE and SQLERRM are used.
  - SQLCODE returns the current error code. And SQLERRM returns the current error message text.
  - The values of SQLCODE and SQLERRM should be assigned to local variables before using it within a SQL statement.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 49

3.2: Exception Handling

Others Exception Handler - Example

```
DECLARE
  v_dummy varchar2(1);
  v_designation number(2) := 109;
BEGIN
    SELECT 'x' into v_dummy FROM designation_master
    WHERE design_code= v_designation;
     INSERT into error_log
    VALUES ('Designation: ' || v_designation || 'already exists');
EXCEPTION
    WHEN no_data_found THEN
        insert into designation_master  values (v_designation,'newdesig');
    WHEN OTHERS THEN
        Err_Num = SQLCODE;
        Err_Msg =SUBSTR( SQLERRM, 1, 100);
         INSERT into errors  VALUES( err_num, err_msg );
END ;
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 50

The example on the slide uses OTHERS Exception handler. If the exception that is raised by the code is not NO_DATA_FOUND, then it will go to the OTHERS exception handler since it will notice that there is no appropriate exception handler defined.
Also observe that the values of SQLCODE and SQLERRM are assigned to variables defined in the block.

3.2: Exception Handling

Raise_Application_Error

- RAISE_APPLICATION_ERROR:
- The procedure RAISE_APPLICATION_ERROR lets you issue user-defined ORA- error messages from stored subprograms.
- In this way, you can report errors to your application and avoid returning unhandled exceptions.
- Syntax:

```
RAISE_APPLICATION_ERROR( Error_Number, Error_Message);
```

- where:
- Error_Number is a parameter between -20000 and -20999
- Error_Message is the text associated with this error

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 51

Raise Application Error:
The built-in function RAISE_APPLICATION_ERROR is used to create our own error messages, which can be more descriptive and user friendly than Exception Names.

3.2: Exception Handling

Raise_Application_Error - Example

- Here is an example of Raise Application Error:

```
DECLARE
    /* VARIABLES */
BEGIN
    ........
    ........
EXCEPTION
    WHEN OTHERS THEN
    -- Will transfer the error to the calling environment
    RAISE_APPLICATION_ERROR( -20999 ,'Contact DBA');
END ;
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 52

3.3: Procedures

Introduction

- A subprogram is a named block of PL/SQL.
- There are two types of subprograms in PL/SQL, namely: Procedures and Functions.
- Each subprogram has:
  - A declarative part
  - An executable part or body, and
  - An exception handling part (which is optional)
- A function is used to perform an action and return a single value.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 53

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.

3.3: Procedures

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

Copyright © Capgemini 2015. All Rights Reserved 54

3.3: Procedures

Procedures

- A procedure is used to perform an action.
- It is illegal to constrain datatypes.
- Syntax:

```
CREATE PROCEDURE  Proc_Name
   (Parameter {IN | OUT | IN OUT} datatype := value,...) AS
   Variable_Declaration ;
   Cursor_Declaration ;
   Exception_Declaration ;
BEGIN
    PL/SQL_Statements ;
EXCEPTION
    Exception_Definition ;
END Proc_Name ;
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 55

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.

3.3: Procedures

Subprogram Parameter Modes

IN	OUT	IN OUT
The default	Must be specified	Must be specified
Used to pass values to the procedure.	Used to return values to the caller.	Used to pass initial values to the procedure and return updated values to the caller.
Formal parameter acts like a constant.	Formal parameter acts like an uninitialized variable.	Formal parameter acts like an uninitialized variable.
Formal parameter cannot be assigned a value.	Formal parameter cannot be used in an expression, but should be assigned a value.	Formal parameter should be assigned a value.
Actual parameter can be a constant, literal, initialized variable, or expression.	Actual parameter must be a variable.	Actual parameter must be a variable.
Actual parameter is passed by reference (a pointer to the value is passed in).	Actual parameter is passed by value (a copy of the value is passed out) unless NOCOPY is specified.	Actual parameter is passed by value (a copy of the value is passed in and out) unless NOCOPY is specified.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 56

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;
```

<u>Subprogram Parameter Modes (contd.)</u>:
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.
```

Example 3:

```
SQL > CREATE OR REPLACE                                    PROCEDURE PRO
    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
```

3.3: Procedures

Examples

- Example 1:

```
CREATE OR REPLACE PROCEDURE raise_salary
  ( s_no IN number, raise_sal IN number) IS
  v_cur_salary   number ;
   missing_salary exception;
BEGIN
     SELECT staff_sal INTO v_cur_salary FROM staff_master
     WHERE staff_code=s_no;
  IF  v_cur_salary IS NULL THEN
     RAISE  missing_salary;
  END IF ;
          UPDATE staff_master SET staff_sal = v_cur_salary + raise_sal
          WHERE staff_code =  s_no ;
EXCEPTION
            WHEN  missing_salary THEN
            INSERT into emp_audit VALUES( sno, 'salary is missing');
 END raise_salary;
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 58

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.

3.3: Procedures

Examples

- Example 2:

```
CREATE OR REPLACE PROCEDURE
    get_details(s_code IN number,
    s_name OUT varchar2,s_sal OUT number ) IS
BEGIN
  SELECT staff_name, staff_sal INTO s_name, s_sal
  FROM staff_master WHERE staff_code=s_code;
EXCEPTION
        WHEN no_data_found  THEN
        INSERT into auditstaff
        VALUES( 'No employee with id ' || s_code);
        s_name := null;
        s_sal := null;
    END get_details ;
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 59

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

3.3: Procedures

Executing a Procedure

- Executing the Procedure from SQL*PLUS environment,
- Create a bind variables salary and name SQLPLUS by using VARIABLE command as follows:

```
variable salary number
variable name varchar2(20)
```

- Execute the procedure with EXECUTE command

```
EXECUTE get_details(100003,:Salary, :Name)
```

- After execution, use SQL*PLUS PRINT command to view results.

```
print salary
print name
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 60

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;
```

```
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 ;
```

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:

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');
```

Example - 2

```
CREATE OR REPLACE PROCEDURE spEmp
   (nEmpno IN employee.empno%TYPE,
    nSal IN OUT NUMBER)
   AS
   nMinSal NUMBER;
   BEGIN
      SELECT min(sal) INTO nMinSal FROM employee;
      IF nSal<nMinSal
      THEN
            nSal:=nSal+nSal*.3;
     END IF;
  END spEmp;
  /
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 63

Example - 2

```
DECLARE
    salno NUMBER;
  BEGIN
    salno:=&salno;
    spEmp(&empno,salno);
    DBMS_OUTPUT.PUT_LINE(salno);
  END;
  /
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 64

3.4: Functions

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.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 65

3.4: Functions

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 66

3.4: Functions

Examples

- Example 1:

```
CREATE FUNCTION crt_dept(dno number,
  dname varchar2)  RETURN number AS
BEGIN
      INSERT into department_master
      VALUES (dno,dname) ;
                return 1 ;
EXCEPTION
      WHEN others THEN
                return 0 ;
END crt_dept ;
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 67

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;
```

3.4: Functions

Executing a Function

- Executing functions from SQL*PLUS:
- Create a bind variable Avg salary in SQLPLUS by using VARIABLE command as follows:

```
variable flag number
```

```
EXECUTE :flag:=crt_dept(60,'Production');
```

- Execute the Function with EXECUTE command:

```
PRINT  flag;
```

- After execution, use SQL*PLUS PRINT command to view results.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 68

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;
```

Calling function using a Select statement

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

3.4: Functions

Exceptions handling in Procedures and Functions

- If procedure has no exception handler for any error, the control immediately passes out of the procedure to the calling environment.
- Values of OUT and IN OUT formal parameters are not returned to actual parameters.
- Actual parameters will retain their old values.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 69

Exceptions raised inside Procedures and Functions:
If an error occurs inside a procedure, an exception (pre-defined or user-defined) is raised.

3.5: Packages

Packages

- A package is a schema object that groups all the logically related PL/SQL types, items, and subprograms.
  - Packages usually have two parts, a specification and a body, although sometimes the body is unnecessary.
  - The specification (spec for short) is the interface to your applications. It declares the types, variables, constants, exceptions, cursors, and subprograms available for use.
  - The body fully defines cursors and subprograms, and so implements the spec.
  - Each part is separately stored in a Data Dictionary.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 70

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.

3.5: Packages

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 71

3.5: Packages

Packages

- Syntax of Package Specification:

```
CREATE PACKAGE package_name  AS
      variable_declaration ;
      cursor_declaration ;
      FUNCTION func_name(param datatype,..) return datatype1 ;
      PROCEDURE proc_name(param {in|out|in out}datatype,...);
END package_name ;
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 72

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

3.5: Packages

Packages

- Syntax of Package Body:

```
CREATE PACKAGE BODY package_name AS
    variable_declaration ;
    cursor_declaration ;
PROCEDURE proc_name(param {IN|OUT|INOUT} datatype,...} IS
BEGIN
        pl/sql_statements ;
END proc_name ;
FUNCTION func_name(param datatype,...) is
    BEGIN
        pl/sql_statements ;
END func_name ;
END package_name ;
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 73

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.

3.5: 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 74

3.5: 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

Copyright © Capgemini 2015. All Rights Reserved 75

3.5: Packages

Executing a Package

- Executing Procedure from a package:

```
EXEC pack1.proc1
Hi a message frm procedure
```

- Executing Function from a package:

```
SELECT pack1.fun1 FROM dual;

FUN1
----------------------------------------
hello from fun1
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 76

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.

3.5: Packages

Package Instantiation

- Package Instantiation:
  - The packaged procedures and functions have to be prefixed with package names.
  - The first time a package is called, it is instantiated.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 77

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.

3.6: Adv Package Concepts

Subprograms and Ref Type Cursors

- You can declare Cursor Variables as the formal parameters of Functions and Procedures.

```
CREATE OR REPLACE PACKAGE staff_data AS
  TYPE staffcurtyp is ref cursor return
    staff_master%rowtype;
  PROCEDURE open_staff_cur (staff_cur INOUT staffcurtyp);
END staff_data;
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 78

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.

3.6: Adv Package Concepts

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

Copyright © Capgemini 2015. All Rights Reserved 79

3.6: Adv Package Concepts

Subprograms and Ref Type Cursors

- Execution in SQL*PLUS:
- Step 1: Declare a bind variable in a PL/SQL host environment of type REFCURSOR.

```
SQL> VARIABLE cv REFCURSOR
```

- Step 2: SET AUTOPRINT ON to automatically display the query results.

```
SQL> set autoprint on
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 80

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);
```

3.6: Adv Package Concepts

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

Copyright © Capgemini 2015. All Rights Reserved 81

3.6: Adv Package Concepts

Subprograms and Ref Type Cursors

- Passing a Cursor Variable as IN parameter to a stored procedure:
- Step 1: Create a Package Specification

```
CREATE OR REPLACE PACKAGE staffdata AS
      TYPE cur_type is REF CURSOR;
      TYPE staffcurtyp is REF CURSOR
      return staff%rowtype;
      PROCEDURE ret_data (staff_cur INOUT staffcurtyp,
      choice in number);
 END staffdata;
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 82

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.

3.6: Adv Package Concepts

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

Copyright © Capgemini 2015. All Rights Reserved 83

3.6: Adv Package Concepts

Subprograms and Ref Type Cursors

- Step 2: Create a Package Body (Contd.)

```
        ELSIF choice = 3 THEN
                OPEN staff_cur for SELECT * FROM
         staff_master WHERE dept_code = 20;
        END IF;
 END ret_data;
END empdata;
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING
Copyright © Capgemini 2015. All Rights Reserved

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);
```

```
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.
```

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:

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.

```
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.
```

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:

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

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

Summary

- n this lesson, you have learnt:
  - Cursor is a “handle” or “name” for a private SQL area.
  - Implicit cursors are declared for queries that return only one row.
  - Explicit cursors are declared for queries that return more than one row.
  - Exception Handling
  - User-defined Exceptions
  - Predefined Exceptions
  - Control passing to Exception Handler
  - OTHERS exception handler
    - Association of Exception name to Oracle errors
    - RAISE_APPLICATION_ERROR procedure
    - Procedures and functions
    - Packages and Adv Packages concepts.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 87

Add the notes here.

Review Question

- Question 1: %COUNT returns number of rows fetched from the cursor area by using FETCH command.
  - True / False
- Question 2: Implicit SQL cursor is opened or closed by the program.
  - True / False
- Question 3: PL/SQL provides a shortcut via a ___ Loop, which implicitly handles the cursor processing.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 88

Add the notes here.

Review Question

- Question 4: The procedure ___ lets you issue user-defined ORA-error messages from stored subprograms
- Question 5: The ___ tells the compiler to associate an exception name with an Oracle error number.
- Question 6: ___ returns the current error code. And ___ returns the current error message text.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 89

Add the notes here.

Review Question

- Question 7: Anonymous Blocks do not have names.
  - True / False
- Question 8: A function can return multiple values by using OUT parameters
  - True / False
- Question 9: A Package consists of "Package Specification" and "Package Body", each of them is stored in a Data Dictionary named DBMS_package.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 50

Add the notes here.

PLSQL

Lesson 04 :Triggers and its types

Lesson Objectives

- To understand the following topics:
  - Describe database triggers and their uses
  - Describe the different types of triggers
  - Create database triggers
  - Describe database trigger-firing rules
  - Remove database triggers
  - Display trigger information

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 2

4.1: Triggers

Triggers

- An event which leads to action
  - Types of Triggers
    - Application : triggers when an event occurs in application
    - Database
- Database triggers are stored procedures that are implicitly executed when an triggering event occurs
  - The triggering event could be (Database triggers):
    - DML statements on the table
    - DDL statements
    - System events such as startup, shutdown, and error messages
    - User events such as logon and logoff

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 3

Types of Triggers

Application triggers execute implicitly whenever a particular data manipulation language (DML) event occurs within an application. An example of an application that uses triggers extensively is an application developed with Oracle Forms Developer.

Database triggers execute implicitly when any of the following events occur:

- DML operations on a table
- DML operations on a view, with an INSTEAD OF trigger
- DDL statements, such as CREATE and ALTER

This is the case no matter which user is connected or which application is used. Database triggers also execute implicitly when some user actions or database system actions occur (for example, when a user logs on or the DBA shuts down the database).

Database triggers can be system triggers on a database or a schema (covered in the next lesson). For databases, triggers fire for each event for all users; for a schema, they fire for each event for that specific user. Oracle Forms can define, store, and run triggers of a different sort. However, do not confuse Oracle Forms triggers with the triggers discussed in this lesson.

Business Application Scenarios for Implementing Triggers

Develop database triggers in order to enhance features that cannot otherwise be implemented by the Oracle server or as alternatives to those provided by the Oracle server.

Security: The Oracle server allows table access to users or roles. Triggers allow table access according to data values.
Auditing: The Oracle server tracks data operations on tables. Triggers track values for data operations on tables.
Data integrity: The Oracle server enforces integrity constraints. Triggers implement complex integrity rules.
Referential integrity: The Oracle server enforces standard referential integrity rules. Triggers implement nonstandard functionality.
Table replication: The Oracle server copies tables asynchronously into snapshots. Triggers copy tables synchronously into replicas.
Derived data: The Oracle server computes derived data values manually. Triggers compute derived data values automatically.
Event logging: The Oracle server logs events explicitly. Triggers log events transparently.

4.1: Triggers

Available Trigger Types

- Simple DML triggers
  - BEFORE
  - AFTER
  - INSTEAD OF
- Compound triggers
  - Non-DML triggers
  - DDL event triggers
  - Database event triggers

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 5

Note

In this lesson, we will discuss the `BEFORE`, `AFTER`, and `INSTEAD OF` triggers. The other trigger types are discussed in the lesson titled “Creating Compound, DDL, and Event Database Triggers.”

4.1: Triggers

Parts of a Trigger

- Triggering event or statement
- Trigger restriction
- Trigger action

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 7

4.1: Triggers

Triggering Event or statement

- A triggering event or statement is the SQL statement, database event, or user event that causes a trigger to fire.
- A triggering event can be one or more of the following:
- An INSERT, UPDATE, or DELETE statement on a specific table/view
- A CREATE, ALTER, or DROP statement on any schema object
- A database startup or instance shutdown
- A specific error message or any error message
- A user logon or logoff

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 8

4.1: Triggers

Trigger Event Types and Body

- A trigger event type determines which DML statement causes the trigger to execute. The possible events are:
  - INSERT
  - UPDATE [OF column]
  - DELETE
- A trigger body determines what action is performed and is a PL/SQL block or a CALL to a procedure.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 9

Triggering Event Types

The triggering event or statement can be an INSERT, UPDATE, or DELETE statement on a table.

When the triggering event is an UPDATE statement, you can include a column list to identify which columns must be changed to fire the trigger. You cannot specify a column list for an INSERT or for a DELETE statement because it always affects entire rows.

. . . UPDATE OF salary . . .

The triggering event can contain one, two, or all three of these DML operations.

. . . INSERT or UPDATE or DELETE

. . . INSERT or UPDATE OF job_id . . .

The trigger body defines the action—that is, what needs to be done when the triggering event is issued. The PL/SQL block can contain SQL and PL/SQL statements, and can define PL/SQL constructs such as variables, cursors, exceptions, and so on. You can also call a PL/SQL procedure or a Java procedure.

Note: The size of a trigger cannot be greater than 32 KB.

4.1: Triggers

Statement-Level Triggers Versus Row-Level Triggers

Statement-Level Triggers	Row-Level Triggers
Is the default when creating a trigger	Use the FOR EACH ROW clause when creating a trigger.
Fires once for the triggering event	Fires once for each row affected by the triggering event
Fires once even if no rows are affected	Does not fire if the triggering event does not affect any rows

Capgemini CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 10

Types of DML Triggers

You can specify that the trigger will be executed once for every row affected by the triggering statement (such as a multiple row UPDATE) or once for the triggering statement, no matter how many rows it affects.

Statement Trigger

A statement trigger is fired once on behalf of the triggering event, even if no rows are affected at all. Statement triggers are useful if the trigger action does not depend on the data from rows that are affected or on data provided by the triggering event itself (for example, a trigger that performs a complex security check on the current user).

Row Trigger

A row trigger fires each time the table is affected by the triggering event. If the triggering event affects no rows, a row trigger is not executed. Row triggers are useful if the trigger action depends on data of the rows that are affected or on data provided by the triggering event itself.

Note: Row triggers use correlation names to access the old and new column values of the row being processed by the trigger.

4.1: Triggers

Trigger Restriction

- Specifies a Boolean expression that must evaluate to TRUE for the trigger to fire

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 11

4.1: Triggers

Trigger Action

- A trigger action is the procedure that contains either PL/SQL block, Java program or C code which contains SQL statements and code to be executed

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 12

Creating DML Triggers

The components of the trigger syntax are:

- *trigger_name* uniquely identifies the trigger.
- *timing* indicates when the trigger fires in relation to the triggering event. Values are BEFORE, AFTER, and INSTEAD OF.
- event identifies the DML operation causing the trigger to fire. Values are INSERT, UPDATE [OF column], and DELETE.
- *object_name* indicates the table or view associated with the trigger.

For row triggers, you can specify:

A REFERENCING clause to choose correlation names for referencing the old and new values of the current row (default values are OLD and NEW)

FOR EACH ROW to designate that the trigger is a row trigger

A WHEN clause to apply a conditional predicate, in parentheses, which is evaluated for each row to determine whether or not to execute the trigger body

The *trigger_body* is the action performed by the trigger, implemented as either of the following:

An anonymous block with a DECLARE or BEGIN, and an END

A CALL clause to invoke a stand-alone or packaged stored procedure, such as:

```
CALL my_procedure;
```

4.1: Triggers

Creating DML Triggers Using SQL Developer

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 14

Trigger-Firing Sequence: Single-Row Manipulation

Create a statement trigger or a row trigger based on the requirement that the trigger must fire once for each row affected by the triggering statement, or just once for the triggering statement, regardless of the number of rows affected.

When the triggering DML statement affects a single row, both the statement trigger and the row trigger fire exactly once.

Example

The SQL statement in the slide does not differentiate statement triggers from row triggers because exactly one row is inserted into the table using the syntax for the `INSERT` statement shown in the slide.

Trigger-Firing Sequence: Multirow Manipulation

When the triggering DML statement affects many rows, the statement trigger fires exactly once, and the row trigger fires once for every row affected by the statement.

Example

The SQL statement in the slide causes a row-level trigger to fire a number of times equal to the number of rows that satisfy the WHERE clause (that is, the number of employees reporting to department 30).

Creating a DML Statement Trigger

In the example in the slide, the `SECURE_EMP` database trigger is a `BEFORE` statement trigger that prevents the `INSERT` operation from succeeding if the business condition is violated. In this case, the trigger restricts inserts into the `EMPLOYEES` table during certain business hours, Monday through Friday.

If a user attempts to insert a row into the `EMPLOYEES` table on Saturday, then the user sees an error message, the trigger fails, and the triggering statement is rolled back. Remember that the `RAISE_APPLICATION_ERROR` is a server-side built-in procedure that returns an error to the user and causes the PL/SQL block to fail.

When a database trigger fails, the triggering statement is automatically rolled back by the Oracle server.

4.1: Triggers

Testing Trigger SECURE_EMP

```
INSERT INTO employees (employee_id, last_name,
first_name, email, hire_date, job_id, salary,
   department_id)
VALUES (300, 'Smith', 'Rob', 'RSMITH', SYSDATE,
'IT_PROG', 4500, 60);
```

Results | Script Output | Explain | Autotrace | DBMS Output | OWA Output

```
Error starting at line 1 in command:
INSERT INTO employees (employee_id, last_name, first_name, email, hire_date,
VALUES (300, 'Smith', 'Rob', 'RSMITH', SYSDATE, 'IT_PROG', 4500, 60)
Error report:
SQL Error: ORA-20500: You may insert into EMPLOYEES table only during  business hours.
ORA-06512: at "ORA42.SECURE_EMP", line 4
ORA-04088: error during execution of trigger 'ORA42.SECURE_EMP'
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 18

Testing SECURE_EMP

To test the trigger, insert a row into the EMPLOYEES table during nonbusiness hours. When the date and time are out of the business hours specified in the trigger, you receive the error message shown in the slide.

4.1: Triggers

Using Conditional Predicates

```
CREATE OR REPLACE TRIGGER secure_emp BEFORE
INSERT OR UPDATE OR DELETE ON employees
  BEGIN
    IF (TO_CHAR(SYSDATE,'DY') IN ('SAT','SUN')) OR
      (TO_CHAR(SYSDATE,'HH24')
       NOT BETWEEN '08' AND '18') THEN
      IF DELETING THEN RAISE_APPLICATION_ERROR(
        -20502,'You may delete from EMPLOYEES table'||
        'only during normal business hours.');
      ELSIF INSERTING THEN RAISE_APPLICATION_ERROR(
        -20500,'You may insert into EMPLOYEES table'||
        'only during normal business hours.');
      ELSIF UPDATING ('SALARY') THEN
        RAISE_APPLICATION_ERROR(-20503, 'You may '||
        'update SALARY only normal during business hours.');
      ELSE RAISE_APPLICATION_ERROR(-20504,'You may'||
        ' update EMPLOYEES table only during'||
        ' normal business hours.');
      END IF;
    END IF;
  END;
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 19

Detecting the DML Operation That Fired a Trigger

If more than one type of DML operation can fire a trigger (for example, `ON INSERT OR DELETE OR UPDATE OF Emp_tab`), the trigger body can use the conditional predicates `INSERTING`, `DELETING`, and `UPDATING` to check which type of statement fired the trigger.

You can combine several triggering events into one by taking advantage of the special conditional predicates `INSERTING`, `UPDATING`, and `DELETING` within the trigger body.

Example

Create one trigger to restrict all data manipulation events on the `EMPLOYEES` table to certain business hours, 8 AM to 6 PM, Monday through Friday.

4.1: Triggers

Creating a DML Row Trigger

```
CREATE OR REPLACE TRIGGER restrict_salary
BEFORE INSERT OR UPDATE OF salary ON employees
FOR EACH ROW
BEGIN
  IF NOT (:NEW.job_id IN ('AD_PRES', 'AD_VP'))
     AND :NEW.salary > 15000 THEN
    RAISE_APPLICATION_ERROR (-20202,
      'Employee cannot earn more than $15,000.');
  END IF;
END;
```

```
UPDATE employees
SET salary = 15500
WHERE last_name = 'Russell';
```

```
Error starting at line 1 in command:
UPDATE employees
SET salary = 15500
WHERE last_name = 'Russell'
Error report:
SQL Error: ORA-20202: Employee cannot earn more than $15,000.
ORA-06512: at "ORA62.RESTRICT_SALARY", line 4
ORA-04088: error during execution of trigger 'ORA62.RESTRICT_SALARY'
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 20

Creating a DML Row Trigger

You can create a BEFORE row trigger in order to prevent the triggering operation from succeeding if a certain condition is violated.

In the first example in the slide, a trigger is created to allow only employees whose job IDs are either AD_PRES or AD_VP to earn a salary of more than 15,000. If you try to update the salary of employee Russell whose employee ID is SA_MAN, the trigger raises the exception displayed in the slide.

Note: Before executing the first code example in the slide, make sure you disable the secure_emp and secure_employees triggers.

4.1: Triggers

Using OLD and NEW Qualifiers

- When a row-level trigger fires, the PL/SQL run-time engine creates and populates two data structures:
  - OLD: Stores the original values of the record processed by the trigger
  - NEW: Contains the new values
- NEW and OLD have the same structure as a record declared using the %ROWTYPE on the table to which the trigger is attached.

Data Operations	Old Value	New Value
INSERT	NULL	Inserted value
UPDATE	Value before update	Value after update
DELETE	Value before delete	NULL

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 21

Using OLD and NEW Qualifiers

Within a ROW trigger, you can reference the value of a column before and after the data change by prefixing it with the OLD and NEW qualifiers.

Note

The OLD and NEW qualifiers are available only in ROW triggers.
Prefix these qualifiers with a colon (:) in every SQL and PL/SQL statement.
There is no colon (:) prefix if the qualifiers are referenced in the WHEN restricting condition.
Row triggers can decrease the performance if you perform many updates on larger tables.

4.1: Triggers

New and Old Values

- New and old values of the DML statements can be processed with :NEW.column_name and :OLD.column_name in the trigger restriction and trigger action .
- Insert will have values in New variable
- Update will have values in New and Old variables
- Delete will have values in Old variable

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 22

4.1: Triggers

Database Triggers - DML

- Create or replace trigger <trigger_name>
- after/before
- insert/update of <column_list>/delete on <table_name/view_name>
- for each row
- When (<condition>)
- <pl_sql >

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 23

4.2: Trigger Types

Types of Trigger

- Row triggers and Statement triggers
- Before and After triggers
- Instead of triggers
- Triggers on system events and user events

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 24

4.2: Trigger Types

Row and Statement Triggers

- Row triggers fire once for every row affected by the triggering statement
- Statement triggers fire once on behalf of the triggering event

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 25

4.2: Trigger Types

Before and After triggers

- Specify the trigger timing
- Are fired by DML statements either before or after the execution of the DML statements
- Apply to row and statement triggers
- Cannot be specified on views
- Trigger type combinations :
- Before statement trigger
- Before row trigger
- After row trigger
- After statement trigger
- You can have multiple triggers of the same type for the same statement for any given table
-

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 26

4.2: Trigger Types

Firing Sequence

- Before Statement trigger
- Before Row trigger
- After Row trigger
- After Statement trigger

Fires for all the affected rows

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 27

4.2: Trigger Types

Trigger Firing Order

- Starting from 11g Oracle allows you to specify trigger firing order if more than one trigger is created .
- It is done using FOLLOWS keyword followed by trigger name after which current trigger is to be invoked.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 28

Contd.,

- create or replace trigger test_trg2 before insert on test
for each row
follows test_trg1
begin
insert into testlog values (‘From test_trig2′);
end;

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 29

4.2: Trigger Types

Instead of triggers

- Complex views which cannot be modified by DML statements can be modified by using Instead of trigger
- It provides a transparent way of modifying the base tables through the views
- Trigger is fired instead of executing the triggering statement

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 30

4.2: Trigger Types

Triggers on System Events and User Events

- Certain system events like database startup and shutdown and server error messages can be traced through triggers
- User events like user logon and logoff, DDL and DML can also be traced through triggers

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 31

4.2: Trigger Types

Example of DML trigger

```
create or replace trigger emp_del
before delete on emp
for each row
begin
        insert into emp_history values
        (:old.empno,:old.ename,:old.job,:old.mgr,:old.hiredate,
        :old.sal,:old.comm,:old.deptno);

end ;
/
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 32

4.2: Trigger Types

Example

```
create or replace trigger dept_tot_emp
after insert on emp
for each row
begin

        update dept set tot_emp = tot_emp + 1
       where deptno = :new.deptno;
end;
/
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 33

4.2: Trigger Types

Example of DML trigger

```
create or replace trigger  dept_tot_emp
after insert or delete or  update of deptno  on emp
for each row
begin

if inserting or updating then
            update dept set tot_emp = tot_emp + 1
            where deptno = :new.deptno;

end if;
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 34

4.2: Trigger Types

Example of DML trigger

```
if updating  or deleting then

        update dept set tot_emp = tot_emp - 1
        where deptno = :old.deptno;
end if;

end;
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 35

4.2: Trigger Types

Example of DML trigger

```
create or replace trigger  dept_tot_emp
after insert or delete or  update of deptno  on emp
for each row
when(old.deptno <> new.deptno)
begin
if inserting or updating then
            update dept set tot_emp = tot_emp + 1
           where deptno = :new.deptno;
end if;
if updating  or deleting then
         update dept set tot_emp = tot_emp - 1
         where deptno = :old.deptno;
end if;
end;
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 36

4.2: Trigger Types

Using OLD and NEW Qualifiers: Example

```
CREATE TABLE audit_emp (
  user_name     VARCHAR2(30),
  time_stamp    date,
  id            NUMBER(6),
  old_last_name VARCHAR2(25),
  new_last_name VARCHAR2(25),
  old_title     VARCHAR2(10),
  new_title     VARCHAR2(10),
  old_salary    NUMBER(8,2),
  new_salary    NUMBER(8,2) )
/
CREATE OR REPLACE TRIGGER audit_emp_values
AFTER DELETE OR INSERT OR UPDATE ON employees
FOR EACH ROW
BEGIN
  INSERT INTO audit_emp(user_name, time_stamp, id,
    old_last_name, new_last_name, old_title,
    new_title, old_salary, new_salary)
  VALUES (USER, SYSDATE, :OLD.employee_id,
    :OLD.last_name, :NEW.last_name, :OLD.job_id,
    :NEW.job_id, :OLD.salary, :NEW.salary);
END;
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 37

Using `OLD` and `NEW` Qualifiers: Example

In the example in the slide, the `AUDIT_EMP_VALUES` trigger is created on the `EMPLOYEES` table. The trigger adds rows to a user table, `AUDIT_EMP`, logging a user's activity against the `EMPLOYEES` table. The trigger records the values of several columns both before and after the data changes by using the `OLD` and `NEW` qualifiers with the respective column name.

4.2: Trigger Types

Using OLD and NEW Qualifiers:Example

```
INSERT INTO employees (employee_id, last_name, job_id, salary, email,
hire_date)
VALUES (999, 'Temp emp', 'SA_REP', 6000, 'TEMPEMP',
TRUNC(SYSDATE))
/
UPDATE employees
 SET salary = 7000, last_name = 'Smith'
 WHERE employee_id = 999
/
SELECT  *
FROM  audit_emp;
```

Results | Script Output | Explain | Autotrace | DBMS Output | OWA Output

Results:

	USER_NAME	TIME_STAMP	ID	OLD_LAST_NAME	NEW_LAST_NAME	OLD_TITLE	NEW_TITLE	OLD_SALARY	NEW_SALARY
1	ORA61	04-JUN-09	(null)	(null)	Temp emp	(null)	SA_REP	(null)	6000
2	ORA61	04-JUN-09	999	Temp emp	Smith	SA_REP	SA_REP	6000	7000

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 38

Using OLD and NEW Qualifiers: Example the Using AUDIT_EMP Table

Create a trigger on the EMPLOYEES table to add rows to a user table, AUDIT_EMP, logging a user's activity against the EMPLOYEES table. The trigger records the values of several columns both before and after the data changes by using the OLD and NEW qualifiers with the respective column name.

The following is the result of inserting the employee record into the EMPLOYEES table:

. . .

The following is the result of updating the salary for employee "Smith":

. . .

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
999	(null)	Smith	TEMPEMP	(null)	04-JUN-09	SA_REP	7000	(null)	(null)	(null)
300	Rob	Smith	RSMITH	(null)	04-JUN-09	IT_PROG	4500	(null)	(null)	60
206	William	Gietz	WGIETZ	515.123.8181	07-JUN-94	AC_ACC...	8300	(null)	205	110

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_...	MANAGER_ID	DEPARTMENT_ID
999	(null)	Smith	TEMPEMP	(null)	04-JUN-09	SA_REP	7000	(null)	(null)	(null)

4.2: Trigger Types

Using the WHEN Clause to Fire a Row Trigger Based on a Condition

```
CREATE OR REPLACE TRIGGER derive_commission_pct
BEFORE INSERT OR UPDATE OF salary ON employees
FOR EACH ROW
WHEN (NEW.job_id = 'SA_REP')
BEGIN
 IF INSERTING THEN
   :NEW.commission_pct := 0;
 ELSIF :OLD.commission_pct IS NULL THEN
   :NEW.commission_pct := 0;
 ELSE
   :NEW.commission_pct := :OLD.commission_pct+0.05;
 END IF;
END;
/
```

Capgemini CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 39

Restricting a Row Trigger: Example

Optionally, you can include a trigger restriction in the definition of a row trigger by specifying a Boolean SQL expression in a `WHEN` clause. If you include a `WHEN` clause in the trigger, then the expression in the `WHEN` clause is evaluated for each row that the trigger affects.

If the expression evaluates to `TRUE` for a row, then the trigger body executes on behalf of that row. However, if the expression evaluates to `FALSE` or `NOT TRUE` for a row (unknown, as with nulls), then the trigger body does not execute for that row. The evaluation of the `WHEN` clause does not have an effect on the execution of the triggering SQL statement (in other words, the triggering statement is not rolled back if the expression in a `WHEN` clause evaluates to `FALSE`).

Note: A `WHEN` clause cannot be included in the definition of a statement trigger.

In the example in the slide, a trigger is created on the `EMPLOYEES` table to calculate an employee's commission when a row is added to the `EMPLOYEES` table, or when an employee's salary is modified.

The `NEW` qualifier cannot be prefixed with a colon in the `WHEN` clause because the `WHEN` clause is outside the PL/SQL blocks.

4.2: Trigger Types

Summary of the Trigger Execution Model

1. Execute all BEFORE STATEMENT triggers.
2. Loop for each row affected by the SQL statement:
   a. Execute all BEFORE ROW triggers for that row.
   b. Execute the DML statement and perform integrity constraint checking for that row.
   c. Execute all AFTER ROW triggers for that row.
3. Execute all AFTER STATEMENT triggers.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 40

Trigger Execution Model

A single DML statement can potentially fire up to four types of triggers:

- BEFORE and AFTER statement triggers
- BEFORE and AFTER row triggers

A triggering event or a statement within the trigger can cause one or more integrity constraints to be checked. However, you can defer constraint checking until a COMMIT operation is performed.

Triggers can also cause other triggers—known as cascading triggers—to fire.

All actions and checks performed as a result of a SQL statement must succeed. If an exception is raised within a trigger and the exception is not explicitly handled, then all actions performed because of the original SQL statement are rolled back (including actions performed by firing triggers). This guarantees that integrity constraints can never be compromised by triggers.

When a trigger fires, the tables referenced in the trigger action may undergo changes by other users' transactions. In all cases, a read-consistent image is guaranteed for the modified values that the trigger needs to read (query) or write (update).

Note: Integrity checking can be deferred until the COMMIT operation is performed.

4.2: Trigger Types

Implementing an Integrity Constraint with an After Trigger

```
-- Integrity constraint violation error –2991 raised.
UPDATE employees SET department_id = 999
 WHERE employee_id = 170;
```

```
CREATE OR REPLACE TRIGGER employee_dept_fk_trg
AFTER UPDATE OF department_id ON employees
FOR EACH ROW
BEGIN
 INSERT INTO departments VALUES(:new.department_id,
            'Dept '||:new.department_id, NULL, NULL);
EXCEPTION
  WHEN DUP_VAL_ON_INDEX THEN
   NULL; -- mask exception if department exists
END;
/
```

```
-- Successful after trigger is fired
UPDATE employees SET department_id = 999
 WHERE employee_id = 170;
```

```
1 rows updated
```

Capgemini CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 41

Implementing an Integrity Constraint with an After Trigger

The example in the slide explains a situation in which the integrity constraint can be taken care of by using an AFTER trigger. The EMPLOYEES table has a foreign key constraint on the DEPARTMENT_ID column of the DEPARTMENTS table.

In the first SQL statement, the DEPARTMENT_ID of the employee 170 is modified to 999. Because department 999 does not exist in the DEPARTMENTS table, the statement raises exception –2291 for the integrity constraint violation.

The EMPLOYEE_DEPT_FK_TRG trigger is created and it inserts a new row into the DEPARTMENTS table by using :NEW.DEPARTMENT_ID for the value of the new department's DEPARTMENT_ID. The trigger fires when the UPDATE statement modifies the DEPARTMENT_ID of employee 170 to 999. When the foreign key constraint is checked, it is successful because the trigger inserted the department 999 into the DEPARTMENTS table. Therefore, no exception occurs unless the department already exists when the trigger attempts to insert the new row. However, the EXCEPTION handler traps and masks the exception allowing the operation to succeed.

Note: Although the example shown in the slide is somewhat contrived due to the limited data in the HR schema, the point is that if you defer the constraint check until the commit, you then have the ability to engineer a trigger to detect that constraint failure and repair it prior to the commit action.

4.2: Trigger Types

Database trigger – Instead off

- Create or replace trigger <trigger_name>
- Instead of insert on<table_name/view_name>
- for each row
- <pl_sql

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 42

INSTEAD OF Triggers

Use INSTEAD OF triggers to modify data in which the DML statement has been issued against an inherently un-updatable view. These triggers are called INSTEAD OF triggers because, unlike other triggers, the Oracle server fires the trigger instead of executing the triggering statement. These triggers are used to perform INSERT, UPDATE, and DELETE operations directly on the underlying tables. You can write INSERT, UPDATE, and DELETE statements against a view, and the INSTEAD OF trigger works invisibly in the background to make the right actions take place. A view cannot be modified by normal DML statements if the view query contains set operators, group functions, clauses such as GROUP BY, CONNECT BY, START, the DISTINCT operator, or joins. For example, if a view consists of more than one table, an insert to the view may entail an insertion into one table and an update to another. So you write an INSTEAD OF trigger that fires when you write an insert against the view. Instead of the original insertion, the trigger body executes, which results in an insertion of data into one table and an update to another table.

Note: If a view is inherently updatable and has INSTEAD OF triggers, then the triggers take precedence. INSTEAD OF triggers are row triggers. The CHECK option for views is not enforced when insertions or updates to the view are performed by using INSTEAD OF triggers. The INSTEAD OF trigger body must enforce the check.

4.2: Trigger Types

Instead of Trigger

```
create or replace trigger emp_details_insert
Instead of insert on emp_details
for each row
begin

insert into emp(EMPNO,ENAME,JOB,MGR,HIREDATE,DEPTNO)
values(:new.empno,:new.ename,:new.job,:new.mgr,:new.hiredate,:new.deptn
o);

insert into emp_addr(EMPNO,ADDRESS,CONTACT)
values (:new.empno,:new.address,:new.contact);

end;
/
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 44

Creating an INSTEAD OF Trigger

You can create an INSTEAD OF trigger in order to maintain the base tables on which a view is based.

The example in the slide illustrates an employee being inserted into the view EMP_DETAILS, whose query is based on the EMPLOYEES and DEPARTMENTS tables. The NEW_EMP_DEPT (INSTEAD OF) trigger executes in place of the INSERT operation that causes the trigger to fire. The INSTEAD OF trigger then issues the appropriate INSERT and UPDATE to the base tables used by the EMP_DETAILS view. Therefore, instead of inserting the new employee record into the EMPLOYEES table, the following actions take place:

1. The NEW_EMP_DEPT INSTEAD OF trigger fires.
2. A row is inserted into the NEW_EMPS table.
3. The DEPT_SAL column of the NEW_DEPTS table is updated. The salary value supplied for the new employee is added to the existing total salary of the department to which the new employee has been assigned.

Note: Before you run the example in the slide, you must create the required structures shown on the next two pages.

4.2: Trigger Types

Example of Instead of Trigger

- EMP table
- EMPNO
- ENAME
- JOB
- MGR
- HIREDATE
- SAL
- COMM
- DEPTNO

EMP_ADDR table

EMPNO
ADDRESS
CONTACT

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 45

4.2: Trigger Types

Defining Complex View

- create view emp_details
- as
- select e.empno,ename,address,contact,job,mgr,hiredate,deptno from emp e,emp_addr a
- where e.empno = a.empno

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 47

4.2: Trigger Types

Creating an INSTEAD OF Trigger to Perform DML on Complex Views

```
CREATE TABLE new_emps AS
 SELECT employee_id,last_name,salary,department_id
      FROM employees;

CREATE TABLE new_depts AS
 SELECT d.department_id,d.department_name,
      sum(e.salary) dept_sal
      FROM employees e, departments d
 WHERE e.department_id = d.department_id;

CREATE VIEW emp_details AS
 SELECT e.employee_id, e.last_name, e.salary,
      e.department_id, d.department_name
 FROM employees e, departments d
 WHERE e.department_id = d.department_id
GROUP BY d.department_id,d.department_name;
```

Capgemini CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 48

Creating an INSTEAD OF Trigger (continued)

The example in the slide creates two new tables, NEW_EMPS and NEW_DEPTS, that are based on the EMPLOYEES and DEPARTMENTS tables, respectively. It also creates an EMP_DETAILS view from the EMPLOYEES and DEPARTMENTS tables.

If a view has a complex query structure, then it is not always possible to perform DML directly on the view to affect the underlying tables. The example requires creation of an INSTEAD OF trigger, called NEW_EMP_DEPT, shown on the next page. The NEW_DEPT_EMP trigger handles DML in the following way:

- When a row is inserted into the EMP_DETAILS view, instead of inserting the row directly into the view, rows are added into the NEW_EMPS and NEW_DEPTS tables, using the data values supplied with the INSERT statement.
- When a row is modified or deleted through the EMP_DETAILS view, corresponding rows in the NEW_EMPS and NEW_DEPTS tables are affected.

Note: INSTEAD OF triggers can be written only for views, and the BEFORE and AFTER timing options are not valid.a

4.2: Trigger Types

The Status of a Trigger

- A trigger is in either of two distinct modes:
  - Enabled: The trigger runs its trigger action if a triggering statement is issued and the trigger restriction (if any) evaluates to true (default).
  - Disabled: The trigger does not run its trigger action, even if a triggering statement is issued and the trigger restriction (if any) would evaluate to true.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 49

4.2: Trigger Types

Creating a Disabled Trigger

- Before Oracle Database 11g, if you created a trigger whose body had a PL/SQL compilation error, then DML to the table failed.
- In Oracle Database 11g, you can create a disabled trigger and then enable it only when you know it will be compiled successfully.

```
CREATE OR REPLACE TRIGGER mytrg
  BEFORE INSERT ON mytable FOR EACH ROW
  DISABLE
BEGIN
  :New.ID := my_seq.Nextval;
. . .
END;
/
```

Capgemini CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 50

Creating a Disabled Trigger

Before Oracle Database 11*g*, if you created a trigger whose body had a PL/SQL compilation error, then DML to the table failed. The following error message was displayed:

```
ORA-04098: trigger 'TRG' is invalid and failed re-validation
```

In Oracle Database 11*g*, you can create a disabled trigger, and then enable it only when you know it will be compiled successfully.

You can also temporarily disable a trigger in the following situations:

- An object it references is not available.
- You need to perform a large data load, and you want it to proceed quickly without firing triggers.
- You are reloading data.

Note: The code example in the slide assumes that you have an existing sequence named `my_seq`.

4.2: Trigger Types

Managing Triggers Using the ALTER and DROP SQL Statements

```
-- Disable or reenable a database trigger:

ALTER TRIGGER trigger_name DISABLE | ENABLE;
```

```
-- Disable or reenable all triggers for a table:

ALTER TABLE table_name DISABLE | ENABLE ALL TRIGGERS;
```

```
-- Recompile a trigger for a table:

ALTER TRIGGER trigger_name COMPILE;
```

```
-- Remove a trigger from the database:

DROP TRIGGER trigger_name;
```

Capgemini CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 51

Managing Triggers

A trigger has two modes or states: `ENABLED` and `DISABLED`. When a trigger is first created, it is enabled by default. The Oracle server checks integrity constraints for enabled triggers and guarantees that triggers cannot compromise them. In addition, the Oracle server provides read-consistent views for queries and constraints, manages the dependencies, and provides a two-phase commit process if a trigger updates remote tables in a distributed database.

Disabling a Trigger

Use the `ALTER TRIGGER` command to disable a trigger. You can also disable all triggers on a table by using the `ALTER TABLE` command. You can disable triggers to improve performance or to avoid data integrity checks when loading massive amounts of data with utilities such as SQL*Loader. You might also disable a trigger when it references a database object that is currently unavailable, due to a failed network connection, disk crash, offline data file, or offline tablespace.

Recompiling a Trigger

Use the `ALTER TRIGGER` command to explicitly recompile a trigger that is invalid.

Removing Triggers

When a trigger is no longer required, use a SQL statement in SQL Developer or SQL*Plus to remove it. When you remove a table, all triggers on that table are also removed.

Managing Triggers Using SQL Developer

You can use the Triggers node in the Connections navigation tree to manage triggers. Right-click a trigger name, and then select one of the following options:

- Edit
- Compile
- Compile for Debug
- Rename
- Drop Trigger
- Enable
- Disable

Testing Triggers

Testing code can be a time-consuming process. Do the following when testing triggers:

- Ensure that the trigger works properly by testing a number of cases separately:
  - Test the most common success scenarios first.
  - Test the most common failure conditions to see that they are properly managed.
- The more complex the trigger, the more detailed your testing is likely to be. For example, if you have a row trigger with a `WHEN` clause specified, then you should ensure that the trigger fires when the conditions are satisfied. Or, if you have cascading triggers, you need to test the effect of one trigger on the other and ensure that you end up with the desired results.
- Use the `DBMS_OUTPUT` package to debug triggers.

4.2: Trigger Types

Viewing Trigger Information

- You can view the following trigger information:

Data Dictionary View	Description
USER_OBJECTS	Displays object information
USER/ALL/DBA_TRIGGERS	Displays trigger information
USER_ERRORS	Displays PL/SQL syntax errors for a trigger

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 54

Viewing Trigger Information

The slide shows the data dictionary views that you can access to get information regarding the triggers.

The `USER_OBJECTS` view contains the name and status of the trigger and the date and time when the trigger was created.

The `USER_ERRORS` view contains the details about the compilation errors that occurred while a trigger was compiling. The contents of these views are similar to those for subprograms.

The `USER_TRIGGERS` view contains details such as name, type, triggering event, the table on which the trigger is created, and the body of the trigger.

The `SELECT Username FROM USER_USERS;` statement gives the name of the owner of the trigger, not the name of the user who is updating the table.

4.2: Trigger Types

Using USER_TRIGGERS

```
DESCRIBE user_triggers
```

```
DESCRIBE user_triggers
Name                        Null     Type
--------------------------- -------- ----------------------------------------
TRIGGER_NAME                         VARCHAR2(30)
TRIGGER_TYPE                         VARCHAR2(16)
TRIGGERING_EVENT                     VARCHAR2(227)
TABLE_OWNER                          VARCHAR2(30)
BASE_OBJECT_TYPE                     VARCHAR2(16)
TABLE_NAME                           VARCHAR2(30)
COLUMN_NAME                          VARCHAR2(4000)
REFERENCING_NAMES                    VARCHAR2(128)
WHEN_CLAUSE                          VARCHAR2(4000)
STATUS                               VARCHAR2(8)
DESCRIPTION                          VARCHAR2(4000)
ACTION_TYPE                          VARCHAR2(11)
TRIGGER_BODY                         LONG()
CROSSEDITION                         VARCHAR2(7)
BEFORE_STATEMENT                     VARCHAR2(3)
BEFORE_ROW                           VARCHAR2(3)
AFTER_ROW                            VARCHAR2(3)
AFTER_STATEMENT                      VARCHAR2(3)
INSTEAD_OF_ROW                       VARCHAR2(3)
FIRE_ONCE                            VARCHAR2(3)
APPLY_SERVER_ONLY                    VARCHAR2(3)

21 rows selected
```

```
SELECT trigger_type, trigger_body
FROM user_triggers
WHERE trigger_name = 'SECURE_EMP';
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 55

Using USER_TRIGGERS

If the source file is unavailable, then you can use the SQL Worksheet in SQL Developer or SQL*Plus to regenerate it from USER_TRIGGERS. You can also examine the ALL_TRIGGERS and DBA_TRIGGERS views, each of which contains the additional column OWNER, for the owner of the object. The result for the second example in the slide is as follows:

```
TRIGGER_TYPE     TRIGGER_BODY
---------------- --------------------------------------------------------------------------------
BEFORE STATEMENT BEGIN
                  IF (TO_CHAR(SYSDATE,'DY') IN ('SAT','SUN')) OR
                     (TO_CHAR(SYSDATE,'HH24') NOT BETWEEN '08' AND '18') THEN
                    IF DELETING THEN
                      RAISE_APPLICATION_ERROR(-20502,
                        'You may delete from EMPLOYEES table only during normal business hours.');
                    ELSIF INSERTING THEN
                      RAISE_APPLICATION_ERROR(-20500,
                        'You may insert into EMPLOYEES table only during normal business hours.');
                    ELSIF UPDATING('SALARY') THEN
                      RAISE_APPLICATION_ERROR(-20503,
                        'You may update SALARY only during normal business hours.');
                    ELSE
                      RAISE_APPLICATION_ERROR(-20504,
                        'You may update EMPLOYEES table only during normal business hours.');
                    END IF;
                  END IF;
                 END;
```

4.2: Trigger Types

Managing Triggers

- Disable or re-enable a database trigger Alter trigger trigger_name disable | enable
- Disable or re-enable all triggers for a table Alter table table name disable | enable all triggers
- Recompile a trigger Alter trigger name compile

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 56

Summary

- Create database triggers that are invoked by DML operations
- Create statement and row trigger types
- Use database trigger-firing rules
- Enable, disable, and manage database triggers
- Develop a strategy for testing triggers
- Remove database triggers

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 57

Add the notes here.

Review Question

- Question 1: Triggers should not issue Transaction Control Statements (TCL)
  - True / False
- Question 2: BEFORE DROP and AFTER DROP triggers are fired when a schema object is dropped
  - True / False
- Question 3: The :new and :old records must be used in WHEN clause with a colon
  - True / False

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 58

Add the notes here.

Review Question

- Question 4: A ___ is a table that is currently being modified by a DML statement
- Question 5: A ___ is fired once on behalf of the triggering statement, regardless of the number of rows in the table that the triggering statement affects

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 59

Add the notes here.

Advanced PLSQL

Lesson 01: Oracle Supplied packages

Lesson Objectives

- To understand the following topics:
- Oracle Supplied Packages like
  - DBMS_OUTPUT
  - UTL_FILE
  - UTL_MAIL

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 2

Oracle supplied packages

- The Oracle-supplied packages:
  - Are provided with the Oracle server
  - Extend the functionality of the database
  - Enable access to certain SQL features that are normally restricted for PL/SQL
- For example, the `DBMS_OUTPUT` package was originally designed to debug PL/SQL programs.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 3

Examples of Some Oracle-Supplied Packages

- Here is an abbreviated list of some Oracle-supplied packages:
  - DBMS_OUTPUT
  - UTL_FILE
  - UTL_MAIL
  - DBMS_ALERT
  - DBMS_LOCK
  - DBMS_SESSION
  - DBMS_APPLICATION_INFO
  - HTP
  - DBMS_SCHEDULER

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 4

Add the notes here.

1.1: Oracle Supplied

Packages:DBMS_OUTPUT DBMS_OUTPUT package

- PL/SQL has no input/output capability
- However, built-in package DBMS_OUTPUT is provided to generate reports
- The procedure PUT_LINE is also provided that places the contents in the buffer

```
PUT_LINE (VARCHAR2 OR NUMBER OR DATE)
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 5

1.1: Oracle Supplied

Packages:DBMS_OUTPUT
How the DBMS_OUTPUT Package Works

- The DBMS_OUTPUT package enables you to send messages from stored subprograms and triggers.
  - PUT and PUT_LINE place text in the buffer.
  - GET_LINE and GET_LINES read the buffer.
  - Messages are not sent until the sending subprogram or trigger completes.
  - Use SET SERVEROUTPUT ON to display messages in SQL Developer and SQL*Plus.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 6

Add the notes here.

DBMS_OUTPUT package

- Syntax:

```
SQL>SET SERVEROUTPUT ON
DECLARE
 V_Variable VARCHAR2(25) :=' Used for'
|| 'Debugging ';
BEGIN
DBMS_OUTPUT.PUT_LINE(V_Variable);
END;
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 7

The code will be written on SQL prompt

1.1: Oracle Supplied

Packages:DBMS_OUTPUT Displaying Output

- DBMS_OUTPUT:
  - DBMS_OUTPUT provides a mechanism for displaying information from the PL/SQL program on to your screen (that is your session's output device).
  - The DBMS_OUTPUT package is created when the Oracle database is installed.
    - The “dbmsoutp.sql” script contains the source code for the specification of this package.
    - This script is called by the “catproc.sql” script, which is normally run immediately after database creation.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 8

Note:

- The catproc.sql script creates the public synonym DBMS_OUTPUT for the package.
- Instance-wise access to this package is provided on installation, so no additional steps should be necessary in order to use DBMS_OUTPUT.

1.1: Oracle Supplied

Packages:DBMS_OUTPUT DBMS_OUTPUT Program Names

Table: DBMS_OUTPUT programs

Name	Description	Use in SQL?
DISABLE	Disables output from the package; the DBMS_OUTPUT buffer will not be flushed to the screen.	Yes
ENABLE	Enables output from the package.	Yes
GET_LINE	Gets a single line from the buffer.	Yes
GET_LINES	Gets specified number of lines from the buffer and passes them into a PL/SQL table.	Yes
NEW_LINE	Inserts an end-of-line mark in the buffer.	Yes
PUT	Puts information into the buffer.	Yes
PUT_LINE	Puts information into the buffer and appends an end-of-line marker after that data.	Yes

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 9

DBMS_OUTPUT Concepts:

- Each user has a DBMS_OUTPUT buffer of up to 1,000,000 bytes in size. You can write information to this buffer by calling the DBMS_OUTPUT.PUT and DBMS_OUTPUT.PUT_LINE programs.
  - If you are using DBMS_OUTPUT from within SQL*Plus, this information will be automatically displayed when your program terminates.
  - You can (optionally) explicitly retrieve information from the buffer with calls to DBMS_OUTPUT.GET and DBMS_OUTPUT.GET_LINE.
- The DBMS_OUTPUT buffer can be set to a size between 2,000 and 1,000,000 bytes with the DBMS_OUTPUT.ENABLE procedure.
  - If you do not enable the package, no information will be displayed or be retrievable from the buffer.
- The buffer stores three different types of data in their internal representations, namely VARCHAR2, NUMBER, and DATE.
  - These types match the overloading available with the PUT and PUT_LINE procedures.
  - Note that DBMS_OUTPUT does not support Boolean data in either its buffer or its overloading of the PUT procedures.

DBMS_OUTPUT - Example

- In this example, the following anonymous PL/SQL block uses DBMS_OUTPUT to display the name and salary of each staff member in department 10:

```
DECLARE
    CURSOR emp_cur IS SELECT staff_name, staff_sal
     FROM staff_master  WHERE dept_code = 10
        ORDER BY staff_sal DESC;
BEGIN FOR emp_rec IN emp_cur
    LOOP
        DBMS_OUTPUT.PUT_LINE (‘Employee ' ||
        emp_rec.staff_name || ' earns ' ||
        TO_CHAR (emp_rec.staff_sal) || ‘rupees');
    END LOOP;
    END;
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 10

DBMS_OUTPUT Concepts:

- The program shown in the slide generates the following output when executed in SQL*Plus:

  Employee John earns 32000 rupees
  Employee Mohan earns 24000 rupees.

1.1: Oracle Supplied

Packages:DBMS_OUTPUT
Writing to DBMS_OUTPUT buffer

- You can write information to the DBMS_OUTPUT buffer with calls to the PUT, NEW_LINE, and PUT_LINE procedures.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 11

Writing to DBMS_OUTPUT buffer

- The DBMS_OUTPUT.PUT procedure:
  - The PUT procedure puts information into the buffer, but does not append a newline marker into the buffer.
  - Use PUT if you want to place information in the buffer (usually with more than one call to PUT), but not also automatically issue a newline marker.
  - The specification for PUT is overloaded, so that you can pass data in its native format to the package without having to perform conversions.

```
PROCEDURE DBMS_OUTPUT.PUT (A VARCHAR2);
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 12

Writing to DBMS_OUTPUT buffer: DBMS_OUTPUT.PUT procedure:

Note:

- The specification for PUT is overloaded, so that you can pass data in its native format to the package without having to perform conversions.

```
PROCEDURE DBMS_OUTPUT.PUT (A VARCHAR2);
PROCEDURE DBMS_OUTPUT.PUT (A NUMBER);
PROCEDURE DBMS_OUTPUT.PUT (A DATE);
```

where A is the data being passed.

Example:

- In the following example, three simultaneous calls to PUT, place the employee name, department ID number, and hire date into a single line in the DBMS_OUTPUT buffer:

```
DBMS_OUTPUT.PUT (:employee.lname || ', ' ||
:employee.fname);
DBMS_OUTPUT.PUT (:employee.department_id);
DBMS_OUTPUT.PUT (:employee.hiredate);
```

- If you follow these PUT calls with a NEW_LINE call, that information can then be retrieved with a single call to GET_LINE.

Writing to DBMS_OUTPUT buffer

- The DBMS_OUTPUT.NEW_LINE procedure:
  - The NEW_LINE procedure inserts an end-of-line marker in the buffer.
  - Use NEW_LINE after one or more calls to PUT in order to terminate those entries in the buffer with a newline marker.
  - Given below is the specification for NEW_LINE:

```
PROCEDURE DBMS_OUTPUT.NEW_LINE;
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 13

Writing to DBMS_OUTPUT buffer

- The DBMS_OUTPUT.PUT_LINE procedure:
  - The PUT_LINE procedure puts information into the buffer, and then appends a newline marker into the buffer.
  - The specification for PUT_LINE is overloaded, so that you can pass data in its native format to the package without having to perform conversions.

```
PROCEDURE DBMS_OUTPUT.PUT_LINE (A VARCHAR2);
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 14

Writing to DBMS_OUTPUT buffer: DBMS_OUTPUT.PUT_LINE procedure:
Note:

- The specification for PUT_LINE is overloaded, so that you can pass data in its native format to the package without having to perform conversions:

```
PROCEDURE DBMS_OUTPUT.PUT_LINE (A
VARCHAR2);
PROCEDURE DBMS_OUTPUT.PUT_LINE (A NUMBER);
PROCEDURE DBMS_OUTPUT.PUT_LINE (A DATE);
```

  where A is the data being passed

- The PUT_LINE procedure is the one most commonly used in SQL*Plus to debug PL/SQL programs.
- When you use PUT_LINE in these situations, you do not need to call GET_LINE to extract the information from the buffer. Instead, SQL*Plus will automatically dump out the DBMS_OUTPUT buffer when your PL/SQL block finishes executing. (You will not see any output until the program ends.)

contd.

Retrieving Data from the DBMS_OUTPUT buffer:

- You can use the GET_LINE and GET_LINES procedures to extract information from the DBMS_OUTPUT buffer.
- If you are using DBMS_OUTPUT from within SQL*Plus, however, you will never need to call either of these procedures. Instead, SQL*Plus will automatically extract the information and display it on the screen for you.

The DBMS_OUTPUT.GET_LINE procedure:

- The GET_LINE procedure retrieves one line of information from the buffer.
- Given below is the specification for the procedure:

```
PROCEDURE DBMS_OUTPUT.GET_LINE
        (line OUT VARCHAR2,
        status OUT INTEGER);
```

- The parameters are summarized as shown below:

Parameter	Description
Line	Retrieved line of text
Status	GET request status

contd.

1.2: Oracle Supplied

Packages: UTL_FILE
Using the UTL_FILE Package

- The UTL_FILE package extends PL/SQL programs to read and write operating system text files:
  - Provides a restricted version of operating system stream file I/O for text files
  - Can access files in operating system directories defined by a CREATE DIRECTORY statement

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 16

Add the notes here.

1.2: Oracle Supplied

Packages: UTL_FILE
Some of the UTL_FILE Procedures and Functions

Subprogram	Description
ISOPEN function	Determines if a file handle refers to an open file
FOPEN function	Opens a file for input or output
FCLOSE function	Closes all open file handles
FCOPY procedure	Copies a contiguous portion of a file to a newly created file
FGETATTR procedure	Reads and returns the attributes of a disk file
GET_LINE procedure	Reads text from an open file
FREMOVE procedure	Deletes a disk file, if you have sufficient privileges
FRENAME procedure	Renames an existing file to a new name
PUT procedure	Writes a string to a file
PUT_LINE procedure	Writes a line to a file, and so appends an operating system-specific line terminator

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 17

Add the notes here.

Add the notes here.

Using the Available Declared Exceptions in the UTL_FILE Package

Exception Name	Description
INVALID_PATH	File location invalid
INVALID_MODE	The open_mode parameter in FOPEN is invalid
INVALID_FILEHANDLE	File handle invalid
INVALID_OPERATION	File could not be opened or operated on as requested
READ_ERROR	Operating system error occurred during the read operation
WRITE_ERROR	Operating system error occurred during the write operation
INTERNAL_ERROR	Unspecified PL/SQL error

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 19

Add the notes here.

1.2: Oracle Supplied

Packages: UTL_FILE FOPEN and IS_OPEN Functions: Example

- This FOPEN function opens a file for input or output.

```
FUNCTION FOPEN (p_location  IN VARCHAR2,
                p_filename  IN VARCHAR2,
                p_open_mode IN VARCHAR2)
RETURN UTL_FILE.FILE_TYPE;
```

- The IS_OPEN function determines whether a file handle refers to an open file.

```
FUNCTION IS_OPEN (p_file IN FILE_TYPE)
RETURN BOOLEAN;
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 20

1.2: Oracle Supplied

Packages: UTL_FILE
Using UTL_FILE: Example

```
CREATE OR REPLACE PROCEDURE sal_status(
  p_dir IN VARCHAR2, p_filename IN VARCHAR2) IS
  f_file UTL_FILE.FILE_TYPE;
  CURSOR cur_emp IS
    SELECT last_name, salary, department_id
    FROM employees ORDER BY department_id;
  v_newdeptno employees.department_id%TYPE;
  v_olddeptno employees.department_id%TYPE := 0;
BEGIN
  f_file:= UTL_FILE.FOPEN (p_dir, p_filename, 'W');
  UTL_FILE.PUT_LINE(f_file,
   'REPORT: GENERATED ON ' || SYSDATE);
  UTL_FILE.NEW_LINE (f_file);
. . .
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 21

1.2: Oracle Supplied

Packages: UTL_FILE
Using UTL_FILE: Example

```
. . .
FOR emp_rec IN cur_emp LOOP
   IF emp_rec.department_id <> v_olddeptno THEN
     UTL_FILE.PUT_LINE (f_file,
     'DEPARTMENT: ' || emp_rec.department_id);
        UTL_FILE.NEW_LINE (f_file);
   END IF;
   UTL_FILE.PUT_LINE (f_file,
     '  EMPLOYEE: ' || emp_rec.last_name ||
            ' earns: ' || emp_rec.salary);
   v_olddeptno := emp_rec.department_id;
   UTL_FILE.NEW_LINE (f_file);
 END LOOP;
 UTL_FILE.PUT_LINE(f_file,'*** END OF REPORT ***');
 UTL_FILE.FCLOSE (f_file);
EXCEPTION
 WHEN UTL_FILE.INVALID_FILEHANDLE THEN
  RAISE_APPLICATION_ERROR(-20001,'Invalid File.');
 WHEN UTL_FILE.WRITE_ERROR THEN
  RAISE_APPLICATION_ERROR (-20002, 'Unable to write to file');
END sal_status;/
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 22

1.3: Oracle Supplied

Packages: UTL_MAIL
What Is the UTL_MAIL Package?

- A utility for managing email
- Requires the setting of the SMTP_OUT_SERVER database initialization parameter
- Provides the following procedures:
  - SEND for messages without attachments
  - SEND_ATTACH_RAW for messages with binary attachments
  - SEND_ATTACH_VARCHAR2 for messages with text attachments

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 23

1.3: Oracle Supplied
Packages: UTL_MAIL
Setting Up and Using the UTL_MAIL: Overview
1. Install the UTL_MAIL package.
2. Define the SMTP_OUT_SERVER Initialization parameter.
3. Grant user the required privilege.
4. Invoke a UTL_MAIL subprogram.
5. Send an email.
PL/SQL
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING
Copyright © Capgemini 2015. All Rights Reserved
24

1.3: Oracle Supplied

Packages: UTL_MAIL
Summary of UTL_MAIL Subprograms

Subprogram	Description
SEND procedure	Packages an email message, locates SMTP information, and delivers the message to the SMTP server for forwarding to the recipients
SEND_ATTACH_RAW Procedure	Represents the SEND procedure overloaded for RAW attachments
SEND_ATTACH_VARCHAR2 Procedure	Represents the SEND procedure overloaded for VARCHAR2 attachments

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 25

1.3: Oracle Supplied

Packages: UTL_MAIL
Installing and Using UTL_MAIL

- As SYSDBA, using SQL Developer or SQL*Plus:
  - Install the UTL_MAIL package

```
@?/rdbms/admin/utlmail.sql
@?/rdbms/admin/prvtmail.plb
```

  - Set the SMTP_OUT_SERVER

```
ALTER SYSTEM SET SMTP_OUT_SERVER='smtp.server.com'
SCOPE=SPFILE
```

- As a developer, invoke a UTL_MAIL procedure:

```
BEGIN
 UTL_MAIL.SEND('otn@oracle.com','user@oracle.com',
   message => 'For latest downloads visit OTN',
   subject => 'OTN Newsletter');
END;
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 26

1.3: Oracle Supplied

Packages: UTL_MAIL
The SEND Procedure Syntax

- Packages an email message into the appropriate format, locates SMTP information, and delivers the message to the SMTP server for forwarding to the recipients

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 27

1.3: Oracle Supplied

Packages: UTL_MAIL
The SEND Procedure Syntax

```
UTL_MAIL.SEND (
   sender       IN    VARCHAR2 CHARACTER SET ANY_CS,
   recipients   IN    VARCHAR2 CHARACTER SET ANY_CS,
   cc           IN    VARCHAR2 CHARACTER SET ANY_CS
                       DEFAULT NULL,
   bcc          IN    VARCHAR2 CHARACTER SET ANY_CS
                        DEFAULT NULL,
   subject      IN    VARCHAR2 CHARACTER SET ANY_CS
                        DEFAULT NULL,
   message      IN    VARCHAR2 CHARACTER SET ANY_CS,
   mime_type    IN    VARCHAR2
                        DEFAULT 'text/plain; charset=us-
ascii',
   priority     IN    PLS_INTEGER DEFAULT NULL);
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 28

1.3: Oracle Supplied

Packages: UTL_MAIL The SEND_ATTACH_RAW Procedure

- This procedure is the SEND procedure overloaded for RAW attachments.

```
UTL_MAIL.SEND_ATTACH_RAW (
  sender          IN    VARCHAR2 CHARACTER SET ANY_CS,
  recipients      IN    VARCHAR2 CHARACTER SET ANY_CS,
  cc              IN    VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL,
  bcc             IN    VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL,
  subject         IN    VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL,
  message         IN    VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL,
  mime_type       IN    VARCHAR2 DEFAULT CHARACTER SET ANY_CS
                        DEFAULT 'text/plain; charset=us-ascii',
  priority        IN    PLS_INTEGER DEFAULT 3,
  attachment      IN    RAW,
  att_inline      IN    BOOLEAN DEFAULT TRUE,
  att_mime_type   IN    VARCHAR2 CHARACTER SET ANY_CS
                        DEFAULT 'text/plain; charset=us-ascii',
  att_filename    IN    VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL);
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 29

1.3: Oracle Supplied Packages: UTL_MAIL

Sending Email with a Binary Attachment: Example

```
CREATE OR REPLACE PROCEDURE send_mail_logo IS
BEGIN
  UTL_MAIL.SEND_ATTACH_RAW(
    sender => 'me@oracle.com',
    recipients => 'you@somewhere.net',
    message =>
      '<HTML><BODY>See attachment</BODY></HTML>',
    subject => 'Oracle Logo',
    mime_type => 'text/html'
    attachment => get_image('oracle.gif'),
    att_inline => true,
    att_mime_type => 'image/gif',
    att_filename => 'oralogo.gif');
END;
/
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 30

1.3: Oracle Supplied Packages: UTL_MAIL

The SEND_ATTACH_VARCHAR2 Procedure

- This procedure is the SEND procedure overloaded for VARCHAR2 attachments.

```
UTL_MAIL.SEND_ATTACH_VARCHAR2 (
  sender          IN    VARCHAR2 CHARACTER SET ANY_CS,
  recipients      IN    VARCHAR2 CHARACTER SET ANY_CS,
  cc              IN    VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL,
  bcc             IN    VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL,
  subject         IN    VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL,
  message         IN    VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL,
  mime_type       IN    VARCHAR2 CHARACTER SET ANY_CS
                        DEFAULT 'text/plain; charset=us-ascii',
  priority        IN    PLS_INTEGER DEFAULT 3,
  attachment      IN    VARCHAR2 CHARACTER SET ANY_CS,
  att_inline      IN    BOOLEAN DEFAULT TRUE,
  att_mime_type   IN    VARCHAR2 CHARACTER SET ANY_CS
                        DEFAULT 'text/plain; charset=us-ascii',
  att_filename    IN    VARCHAR2CHARACTER SET ANY_CS DEFAULT NULL);
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 31

1.3: Oracle Supplied Packages: UTL_MAIL

Sending Email with a Text Attachment: Example

```
CREATE OR REPLACE PROCEDURE send_mail_file IS
BEGIN
  UTL_MAIL.SEND_ATTACH_VARCHAR2(
    sender => 'me@oracle.com',
    recipients => 'you@somewhere.net',
    message =>
      '<HTML><BODY>See attachment</BODY></HTML>',
    subject => 'Oracle Notes',
    mime_type => 'text/html'
    attachment => get_file('notes.txt'),
    att_inline => false,
    att_mime_type => 'text/plain',
    att_filename => 'notes.txt');
END;
/
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 32

Summary

- In this lesson you have learnt:
  - How the `DBMS_OUTPUT` package works
  - How to use `UTL_FILE` to direct output to operating system files
  - About the main features of `UTL_MAIL`

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 33

Add the notes here.

Review Question

- Which of the following is the most frequently used package in reports
  - DBMS_OUTPUT
  - UTL_FILE
  - SRW
  - DBMS_SQL
- With the UTL_FILE package, PL/SQL programs can read and write operating system text files
  - True/False
- Question 3: _____ can be opened on the server and passed to the client as a unit rather than fetching one row at a time.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 34

Add the notes here.

Advanced PLSQL

Lesson 02: Design Considerations

Lesson Objectives

- To understand the following topics:
  - Design Considerations like
    - Bulk Binding concepts
    - Ref cursors
    - Using NOCOPY hint
    - Using PARALLEL_ENABLE hint
    - Using Cross-session PL/SQL function
    - Using DETERMINISTIC clause
    - Using returning clause

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 2

2.1: Design Considerations: bulk binding
Bulk Binding

- Bulk binding binds whole arrays of values in a single operation, rather than using a loop to perform a FETCH, INSERT, UPDATE, and DELETE operation multiple times

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 3

2.1: Design Considerations: bulk binding

Bulk Binding: Syntax and Keywords

- The FORALL keyword instructs the PL/SQL engine to bulk bind input collections before sending them to the SQL engine.

```
FORALL index IN lower_bound .. upper_bound
  [SAVE EXCEPTIONS]
  sql_statement;
```

- The BULK COLLECT keyword instructs the SQL engine to bulk bind output collections before returning them to the PL/SQL engine.

```
... BULK COLLECT INTO
      collection_name[,collection_name] ...
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 4

2.1: Design Considerations: bulk binding

Bulk Binding FORALL: Example

```
CREATE PROCEDURE raise_salary(p_percent NUMBER) IS
  TYPE numlist_type IS TABLE OF NUMBER
       INDEX BY BINARY_INTEGER;
  v_id  numlist_type; -- collection
BEGIN
 v_id(1):= 100; v_id(2):= 102; v_id(3):= 104; v_id(4) := 110;
 -- bulk-bind the PL/SQL table
  FORALL i IN v_id.FIRST .. v_id.LAST
    UPDATE employees
      SET salary = (1 + p_percent/100) * salary
      WHERE employee_id = v_id(i);
END;
/
```

```
EXECUTE raise_salary(10)
```

```
anonymous block completed
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 5

2.1: Design Considerations: bulk binding

Using BULK COLLECT INTO with Queries

- The SELECT statement supports the BULK COLLECT INTO syntax.

```
CREATE PROCEDURE get_departments(p_loc NUMBER) IS
  TYPE dept_tab_type IS
    TABLE OF departments%ROWTYPE;
  v_depts dept_tab_type;
BEGIN
  SELECT * BULK COLLECT INTO v_depts
  FROM departments
  WHERE location_id = p_loc;
  FOR i IN 1 .. v_depts.COUNT LOOP
    DBMS_OUTPUT.PUT_LINE(v_depts(i).department_id
     ||' '|| v_depts(i).department_name);
  END LOOP;
END;
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 5

2.1: Design Considerations: bulk binding

Using BULK COLLECT INTO with Cursors

- The FETCH statement has been enhanced to support the BULK COLLECT INTO syntax.

```
CREATE OR REPLACE PROCEDURE get_departments(p_loc NUMBER) IS
  CURSOR cur_dept IS
    SELECT * FROM departments
    WHERE location_id = p_loc;
  TYPE dept_tab_type IS TABLE OF cur_dept%ROWTYPE;
  v_depts dept_tab_type;
BEGIN
  OPEN cur_dept;
  FETCH cur_dept BULK COLLECT INTO v_depts;
  CLOSE cur_dept;
 FOR i IN 1 .. v_depts.COUNT LOOP
    DBMS_OUTPUT.PUT_LINE(v_depts(i).department_id
     ||' '|| v_depts(i).department_name);
  END LOOP;
END;
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 7

2.1: Design Considerations: bulk binding

Using BULK COLLECT INTO with a RETURNING Clause

```
CREATE OR REPLACE PROCEDURE raise_salary(p_rate NUMBER) IS
  TYPE emplist_type IS TABLE OF NUMBER;
  TYPE numlist_type IS TABLE OF employees.salary%TYPE
    INDEX BY BINARY_INTEGER;
  v_emp_ids  emplist_type := emplist_type(100,101,102,104);
  v_new_sals numlist_type;
BEGIN
 FORALL i IN v_emp_ids.FIRST .. v_emp_ids.LAST
  UPDATE employees
    SET commission_pct = p_rate * salary
  WHERE employee_id = v_emp_ids(i)
  RETURNING salary BULK COLLECT INTO v_new_sals;
 FOR i IN 1 ..     v_new_sals.COUNT LOOP
  DBMS_OUTPUT.PUT_LINE(v_new_sals(i));
 END LOOP;
END;
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 8

2.1: Design Considerations: bulk binding

Using Bulk Binds in Sparse Collections

- Current releases:

Index by
PLS_INTEGER

```
FOR ALL . . .
IN INDICES OF
```

- Prior releases:

Index by
BINARY_INTEGER

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 9

2.1: Design Considerations: bulk binding

Using Bulk Binds in Sparse Collections

```
-- The INDICES OF syntax allows the bound arrays
-- themselves to be sparse.

FORALL index_name IN INDICES OF sparse_array_name
      BETWEEN LOWER_BOUND AND UPPER_BOUND -- optional
      SAVE EXCEPTIONS -- optional, but recommended
      INSERT INTO table_name VALUES sparse_array(index_name);
. . .
```

```
-- The VALUES OF syntax lets you indicate a subset
-- of the binding arrays.

FORALL index_name IN VALUES OF index_array_name
      SAVE EXCEPTIONS -- optional,but recommended
      INSERT INTO table_name VALUES
     binding_array_name(index_name);
. . .
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 10

2.1: Design Considerations: bulk binding

Using Bulk Binds in Sparse Collections

- The typical application for this feature is an order entry and order processing system where:
  - Users enter orders through the Web
  - Orders are placed in a staging table before validation
  - Data is later validated based on complex business rules (usually implemented programmatically using PL/SQL)
  - Invalid orders are separated from valid ones
  - Valid orders are inserted into a permanent table for processing

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 11

2.1: Design Considerations: bulk binding

Using Bulk Bind with Index Array

```
CREATE OR REPLACE PROCEDURE ins_emp2 AS
  TYPE emptab_type IS TABLE OF employees%ROWTYPE;
  v_emp emptab_type;
  TYPE values_of_tab_type IS TABLE OF PLS_INTEGER
        INDEX BY PLS_INTEGER;
  v_num   values_of_tab_type;
   . . .
BEGIN
   . . .
   FORALL k IN VALUES OF v_num
   INSERT INTO new_employees VALUES v_emp(k);
END;
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 12

2.2: Design Considerations: REF CURSOR types

REF cursor

- Defining REF CURSOR types:
- Syntax:

```
TYPE ref_type_name IS REF CURSOR RETURN return_type;
    DECLARE
        TYPE DeptCurTyp IS REF CURSOR RETURN
department_master%ROWTYPE;
```

- where:
- ref_type_name is a type specifier used in subsequent declarations of cursor variables
- Return_type must represent a record or a row in a database table.
- REF CURSOR types are strong (restrictive), or weak (non-restrictive)

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 13

Add the notes here.

2.2: Design Considerations: REF CURSOR types

REF cursor example

```
DECLARE
    TYPE staffCurTyp IS REF CURSOR
    RETURN staff_master%ROWTYPE;  -- Strong types

    TYPE GenericCurTyp IS REF CURSOR;  -- Weak types
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 14

Add the notes here.

2.2: Design Considerations: REF CURSOR types

REF cursor example

- Declaring Cursor Variables:
  - Example 1:

```
DECLARE
TYPE DeptCurTyp IS REF CURSOR RETURN
department_master%ROWTYPE;
dept_cv        DeptCurTyp;  -- Declare cursor variable
```

  - You cannot declare cursor variables in a package.
  - Example 2:

```
TYPE TmpCurTyp IS REF CURSOR RETURN
staff_master%ROWTYPE;
tmp_cv TmpCurTyp;  -- Declare cursor variable
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 15

Add the notes here.

2.2: Design Considerations: REF CURSOR types

REF cursor example

```
DECLARE
      TYPE staffcurtyp is REF CURSOR RETURN
       staff_master%rowtype;
      staff_cv    staffcurtyp;  -- declare cursor variable
      staff_cur        staff_master%rowtype;
BEGIN
       open staff_cv for  select * from staff_master;
 LOOP
          EXIT WHEN staff_cv%notfound;
          FETCH staff_cv into staff_cur;
          INSERT into temp_table  VALUES (staff_cv.staff_code,
              staff_cv.staff_name,staff_cv.staff_sal);
       END LOOP;
       CLOSE staff_cv;
     END;
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 16

Add the notes here.

2.3: Design Considerations: using NOCOPY

Using the NOCOPY Hint

- The typical application for this feature is an order entry and order processing system where:
  - Users enter orders through the Web
  - Orders are placed in a staging table before validation
  - Data is later validated based on complex business rules (usually implemented programmatically using PL/SQL)
  - Invalid orders are separated from valid ones
  - Valid orders are inserted into a permanent table for processing

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 17

2.3: Design Considerations: using NOCOPY

Effects of the NOCOPY Hint

- If the subprogram exits with an exception that is not handled:
  - You cannot rely on the values of the actual parameters passed to a `NOCOPY` parameter
  - Any incomplete modifications are not "rolled back"
- The remote procedure call (RPC) protocol enables you to pass parameters only by value.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 18

When Does the PL/SQL Compiler Ignore the NOCOPY Hint?

- The NOCOPY hint has no effect if:
  - The actual parameter:
    - Is an element of associative arrays (index-by tables)
    - Is constrained (for example, by scale or NOT NULL)
    - And formal parameter are records, where one or both records were declared by using %ROWTYPE or %TYPE, and constraints on corresponding fields in the records differ
    - Requires an implicit data type conversion
  - The subprogram is involved in an external or remote procedure call

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 19

2.4: Design Considerations: using PARALLEL_ENABLE

Using the PARALLEL_ENABLE Hint

- Can be used in functions as an optimization hint
- Indicates that a function can be used in a parallelized query or parallelized DML statement

```
CREATE OR REPLACE FUNCTION f2 (p_p1 NUMBER)
  RETURN NUMBER PARALLEL_ENABLE IS
BEGIN
  RETURN p_p1 * 2;
END f2;
```

```
FUNCTION f2 Compiled.
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 20

2.5: Design Considerations: using Cross-session PL/SQL function

Using the Cross-Session PL/SQL Function Result Cache

- Each time a result-cached PL/SQL function is called with different parameter values, those parameters and their results are stored in cache.
- The function result cache is stored in a shared global area (SGA), making it available to any session that runs your application.
- Subsequent calls to the same function with the same parameters uses the result from cache.
- Performance and scalability are improved.
- This feature is used with functions that are called frequently and dependent on information that changes infrequently.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 21

Enabling Result-Caching for a Function

- You can make a function result-cached as follows:
  - Include the RESULT_CACHE clause in the following:
    - The function declaration
    - The function definition
  - Include an optional RELIES_ON clause to specify any tables or views on which the function results depend.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 22

Declaring and Defining a Result-Cached Function: Example

```
CREATE OR REPLACE FUNCTION emp_hire_date (p_emp_id NUMBER)
     RETURN VARCHAR
 RESULT_CACHE RELIES_ON (employees) IS
 v_date_hired DATE;
BEGIN
  SELECT hire_date INTO v_date_hired
  FROM HR.Employees
  WHERE Employee_ID = p_emp_ID;
  RETURN to_char(v_date_hired);
END;
```

```
FUNCTION emp_hire_date Compiled.
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 23

2.6: Design Considerations: using DETERMINISTIC clause

Using the DETERMINISTIC Clause with Functions

- Specify DETERMINISTIC to indicate that the function returns the same result value whenever it is called with the same values for its arguments.
- This helps the optimizer avoid redundant function calls.
- If a function was called previously with the same arguments, the optimizer can elect to use the previous result.
- Do not specify DETERMINISTIC for a function whose result depends on the state of session variables or schema objects.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 24

2.7: Design Considerations: using returning clause

Using the RETURNING Clause

- Improves performance by returning column values with INSERT, UPDATE, and DELETE statements
- Eliminates the need for a SELECT statement

```
CREATE OR REPLACE PROCEDURE update_salary(p_emp_id NUMBER) IS
  v_name    employees.last_name%TYPE;
  v_new_sal employees.salary%TYPE;
BEGIN
  UPDATE employees
    SET salary = salary * 1.1
  WHERE employee_id = p_emp_id
  RETURNING last_name, salary INTO v_name, v_new_sal;
  DBMS_OUTPUT.PUT_LINE(v_name || ' new salary is ' ||
      v_new_sal);
END update_salary;
/
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 25

Summary

In this lesson you have learnt:

- Design Considerations like
  - Bulk Binding concepts
  - REF CURSOR types
  - Using NOCOPY hint
  - Using PARALLEL_ENABLE hint
  - Using Cross-session PL/SQL function
  - Using DETERMINISTIC clause
  - Using returning clause

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 26

Add the notes here.

Review Question

- To binding entire columns of Oracle data ____ clause is used by users
  - BIND_COLLECT
  - BULK_COLLECT
  - GROUP_COLUMN
  - GROUP_COLUMNS
- REF CURSOR types are _____ or _____ type.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 27

Add the notes here.

Advanced PLSQL

Lesson 03: Different types of pragma

Lesson Objectives

- In this lesson you will learn
  - Different types of Pragma like
  - EXCEPTION_INIT
  - PRAGMA AUTONOMOUS_TRANSACTION
  - PRAGMA SERIALLY_REUSABLE
  - PRAGMA RESTRICT_REFRENCES

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 2

Oracle PL/SQL PRAGMA

- In Oracle PL/SQL, **PRAGMA** refers to a compiler directive or "hint" it is used to provide an instruction to the compiler.
- The directive restricts member subprograms to query or modify database tables and packaged variables.
- Pragma directives are processed at compile time where they pass necessary information to the compiler;
- They are not processed at runtime.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 3

Types of PLSQL Pragma

- The 4 types of Pragma directives available in Oracle are listed below:
  - PRAGMA EXCEPTION_INIT
  - PRAGMA AUTONOMOUS_TRANSACTION
  - PRAGMA SERIALLY_REUSABLE
  - PRAGMA RESTRICT_REFRENCES

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 4

3.1: Type of Pragma: Exception_INIT

Standardizing Exception (EXCEPTION_INIT)

- Create a standardized error-handling package that includes all named and programmer-defined exceptions to be used in the application.

```
CREATE OR REPLACE PACKAGE error_pkg IS
  e_fk_err     EXCEPTION;
  e_seq_nbr_err EXCEPTION;
  PRAGMA EXCEPTION_INIT (e_fk_err, -2292);
  PRAGMA EXCEPTION_INIT (e_seq_nbr_err, -2277);
  ...
END error_pkg;
/
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 5

3.1: Type of Pragma: Exception_INIT

Standardizing Exception

- Consider writing a subprogram for common exception handling to:
  - Display errors based on SQLCODE and SQLERRM values for exceptions
  - Track run-time errors easily by using parameters in your code to identify:
    - The procedure in which the error occurred
    - The location (line number) of the error
    - RAISE_APPLICATION_ERROR using stack trace capabilities, with the third argument set to TRUE

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 5

3.2: Type of Pragma: Autonomous_transations

Autonomous Transactions

- Are independent transactions started by another main transaction
- Are specified with PRAGMA AUTONOMOUS_TRANSACTION

Main Transaction

```
PROCEDURE proc1 IS
  emp_id  NUMBER;
BEGIN
  emp_id := 1234;
  COMMIT;
  INSERT ...
  proc2;
  DELETE ...
  COMMIT;
END proc1;
```

Autonomous (independent) Transaction

```
PROCEDURE proc2 IS
 PRAGMA
    AUTONOMOUS_TRANSACT
    ION;
 dept_id  NUMBER := 90;
BEGIN
  UPDATE ...
  INSERT ...
  COMMIT;   -- Requi
END proc2;
```

1 2 3 4 5 6 7

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

3.2: Type of Pragma: Autonomous_transations

Features of Autonomous Transactions

- Are independent of the main transaction
- Suspend the calling transaction until the autonomous transactions are completed
- Are not nested transactions
- Do not roll back if the main transaction rolls back
- Enable the changes to become visible to other transactions upon a commit
- Are started and ended by individual subprograms and not by nested or anonymous PL/SQL blocks

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 8

3.2: Type of Pragma: Autonomous_transations

Using Autonomous Transactions: Example

```
CREATE TABLE usage (card_id NUMBER, loc NUMBER)
/
CREATE TABLE txn (acc_id NUMBER, amount NUMBER)
/
CREATE OR REPLACE PROCEDURE log_usage (p_card_id NUMBER, p_loc NUMBER) IS

BEGIN
  INSERT INTO usage
  VALUES (p_card_id, p_loc);
  COMMIT;
END log_usage;
/
CREATE OR REPLACE PROCEDURE bank_trans(p_cardnbr NUMBER,p_loc NUMBER) IS
BEGIN
   INSERT INTO txn VALUES (9001, 1000);
   log_usage (p_cardnbr, p_loc);
END bank_trans;
/
EXECUTE bank_trans(50, 2000)
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 9

3.3: Type of Pragma: Serially_reusable

SERIALLY_REUSABLE Pragma

- The pragma SERIALLY_REUSABLE indicates that the package state is needed only for the duration of one call to the server.
- An example could be an OCI call to the database or a stored procedure call through a database link.
- After this call, the storage for the package variables can be reused, reducing the memory overhead for long-running sessions.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 10

3.3: Type of Pragma: Serially_reusable

Creating a Serially Reusable Package

```
CREATE PACKAGE pkg1 IS
  PRAGMA SERIALLY_REUSABLE;
  num NUMBER := 0;
  PROCEDURE init_pkg_state(n NUMBER);
  PROCEDURE print_pkg_state;
END pkg1;
/
CREATE PACKAGE BODY pkg1 IS
  PRAGMA SERIALLY_REUSABLE;
  PROCEDURE init_pkg_state (n NUMBER) IS
  BEGIN
    pkg1.num := n;
  END;
  PROCEDURE print_pkg_state IS
  BEGIN
    DBMS_OUTPUT.PUT_LINE('Num: ' || pkg1.num);
  END;
END pkg1;
/
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 11

3.4: Type of Pragma: Restrict_references

RESTRICT_REFERENCES Pragma

- If any SQL statement inside the function body violates a rule, you get an error at run time (when the statement is parsed).
- To check for violations of the rules at compile time, you can use the compiler directive PRAGMA RESTRICT_REFERENCES.
- This pragma asserts that a function does not read and/or write database tables and/or package variables.
- Functions that do any of these read or write operations are difficult to optimize, because any call might produce different results or encounter errors.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 12

3.4: Type of Pragma: Restrict_references

Usage of RESTRICT_REFERENCES

- You can declare the pragma RESTRICT_REFERENCES only in a package spec or object type spec
- You can specify up to four constraints RNDS, RNPS, WNDS, WNPS in any order.
- A RESTRICT_REFERENCES pragma can apply to only one subprogram declaration
- A pragma that references the name of overloaded subprograms always applies to the most recent subprogram declaration.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 13

3.4: Type of Pragma: Restrict_references

RESTRICT_REFERENCES Example

```
-- create the debug table
CREATE TABLE debug_output (msg VARCHAR2(200));

-- create the package spec
CREATE PACKAGE debugging AS
  FUNCTION log_msg (msg VARCHAR2) RETURN VARCHAR2;
  PRAGMA RESTRICT_REFERENCES(log_msg, WNDS, RNDS);
END debugging;
/
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 14

3.4: Type of Pragma: Restrict_references

RESTRICT_REFERENCES Example ..contd

```
-- create the package body
CREATE PACKAGE BODY debugging AS
  FUNCTION log_msg (msg VARCHAR2) RETURN VARCHAR2 IS
    PRAGMA AUTONOMOUS_TRANSACTION;
  BEGIN
    -- the following insert does not violate the constraint
    -- WNDS because this is an autonomous routine
    INSERT INTO debug_output VALUES (msg);
    COMMIT;
    RETURN msg;
  END;
END debugging;
/
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 15

3.4: Type of Pragma: Restrict_references

RESTRICT_REFERENCES Example ..contd.

```
-- call the packaged function from a query
DECLARE
  my_emp_id    NUMBER(6);
  my_last_name VARCHAR2(25);
  my_count     NUMBER;
BEGIN
  my_emp_id := 120;
  SELECT debugging.log_msg(last_name) INTO my_last_name FROM
employees
    WHERE employee_id = my_emp_id;
-- even if you roll back in this scope, the insert into 'debug_output' remains
-- committed because it is part of an autonomous transaction
  ROLLBACK;
END;
/
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 16

Summary

In this lesson you have learnt:

- Different types of PLSQL pragma
  - EXCEPTION_INIT
  - PRAGMA AUTONOMOUS_TRANSACTION
  - PRAGMA SERIALLY_REUSABLE
  - PRAGMA RESTRICT_REFRENCES

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 17

Add the notes here.

Review Question

- Which compiler directive to check the purity level of functions?
  A. PRAGMA SEARIALLY_REUSABLE.
  B.PRAGMA RESTRICT_REFERRENCES.
  C.PRAGMA RESTRICT_PURITY_LEVEL
  D.PRAGMA RESTRICT_FUNCTION_REFERRENCE
- Oracle PL/SQL PRAGMA are processed at runtime
  - True/False
- The pragma ____ indicates that the package state is needed only for the duration of one call to the server

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 18

Add the notes here.

Advanced PLSQL

Lesson 04: PLSQL collection elements

Lesson Objectives

In this lesson you will learn

- PLSQL Collection elements
  - PLSQL Tables
  - Nested tables
  - Varrays
  - Associative Arrays

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 2

PLSQL Collections

- A Collection is a group of elements of the same kind
- There are three types of Collections that you can use in PL/SQL.
  - PL/SQL Tables
  - Nested Tables
  - Variable Arrays(VARRAY).
  - Associative Arrays

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 3

4.1: PLSQL tables

PLSQL Tables

- Collection of elements with the same name and same datatype.
- A PL/SQL table has two columns, one is a PRIMARY KEY called the index, and the other holds the elements
- The value column may have any datatype
- The key column must be binary_integer

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 4

PL/SQL Table Example

```
DECLARE
        TYPE num_table IS TABLE OF NUMBER
        INDEX BY binary_integer;
        num_rec num_table;
BEGIN
        SELECT salary
        INTO num_rec(1)
        FROM sales_person
        WHERE sales_person_id = 800;
        Dbms_output.Put_line (num_rec(1));
END;
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 5

4.2: PLSQL Nested tables

Nested Tables

- Nested Tables act as one-column database tables, which can store multiple rows of data from a database table.
- Oracle does not store rows in a nested table in any particular order, but if you retrieve a table into a PL/SQL Collection, the rows are indexed consecutively starting at 1.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 6

Nested Tables

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 7

Nested Table Example

```
DECLARE
        TYPE color IS TABLE OF VARCHAR2 (20);
        rainbow color;
BEGIN
        rainbow := color ('RED', 'ORANGE', 'YELLOW',
                'GREEN', 'BLUE', 'INDIGO', 'VIOLET');
        FOR ctr IN 1..7 LOOP
                Dbms_output.Put_line (rainbow (ctr));
        END LOOP;
END;
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 8

PL/SQL tables Vs Nested tables

PL/SQL tables

- Cannot be used to define the type of a database column
- You cannot SELECT, INSERT, UPDATE, or DELETE elements in a PL/SQL Table.
- Elements can have negative indices
- Increasing the number of elements is easy you just have to assign a new element.

Nested tables

- Can be used to define the type of a database column
- You can SELECT, INSERT, UPDATE, or DELETE elements in a Nested Table
- Elements cannot have negative indices
- To increase the number of elements, you must use the EXTEND method to increase the size of the Collection

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 9

4.3: VARRAYS

Variable Arrays

- VARRAY, allow you to connect a single identifier with an entire Collection of data.
- They are different from Nested tables in that you must specify an upper limit for the number of elements
- A VARRAY is generally used when you must retrieve an entire Collection that is not very large.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 10

VARRAY

1	Bombay
2	Sydney
3	Oxford
4	London
..	....
10	Tokyo

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 11

VARRAY Example

```
DECLARE
        TYPE temperature IS VARRAY(52) OF NUMBER;
        weekly_temp temperature := temperature(60, 65, 70, 65, 59, 60, 74);
        temp_count binary_integer;
BEGIN
        temp_count := weekly_temp.count;
        IF weekly_temp.LIMIT - temp_count > 0 THEN
                weekly_temp.EXTEND;
                weekly_temp (temp_count +1) := 73;
        END IF;
        FOR i IN 1..weekly_temp.COUNT LOOP
                Dbms_output.Put_line (i || chr(9) || weekly_temp(i));
        END LOOP;
END;
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 12

4.4: Associative arrays

Associative Arrays (INDEX BY Tables)

- An associative array is a PL/SQL collection with two columns:
  - Primary key of integer or string data type
  - Column of scalar or record data type

Key	Values
1	JONES
2	HARDEY
3	MADURO
4	KRAMER

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 13

Associative Array Structure
1
Unique key column
...
1
5
3
...
PLS_INTEGER
2
---- Values column ----
<or>
...
Jones
Smith
Maduro
...
Scalar
...
110 ADMIN Jones
103 ADMIN Smith
176 IT_PROG Maduro
...
Record
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING
Copyright © Capgemini 2015. All Rights Reserved
14

Associative Array Structure

- Syntax:

```
1  TYPE type_name IS TABLE OF
       {column_type | variable%TYPE
       | table.column%TYPE} [NOT NULL]
       | table%ROWTYPE
       | INDEX BY PLS_INTEGER | BINARY_INTEGER
       | VARCHAR2(<size>);
2  identifier        type_name;
```

- Example:

```
...
TYPE ename_table_type IS TABLE OF
 employees.last_name%TYPE
 INDEX BY PLS_INTEGER;
...
ename_table ename_table_type;
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 15

Creating and Accessing Associative Arrays

```
...
DECLARE
  TYPE ename_table_type IS TABLE OF
    employees.last_name%TYPE
    INDEX BY PLS_INTEGER;
  TYPE hiredate_table_type IS TABLE OF DATE
    INDEX BY PLS_INTEGER;
  ename_table      ename_table_type;
  hiredate_table   hiredate_table_type;
BEGIN
  ename_table(1)   := 'CAMERON';
  hiredate_table(8) := SYSDATE + 7;
    IF ename_table.EXIS
    INSERT INTO ...
    END;
/
```

```
anonymous block completed
ENAME                     HIREDT
------------------------- --------------------
CAMERON                   23-JUN-09

1 rows selected
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 16

Using INDEX BY Table Methods

- The following methods make associative arrays easier to use:
  - EXISTS
  - COUNT
  - FIRST
  - LAST
  - PRIOR
  - NEXT
  - DELETE

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 17

INDEX BY Table of Records Option

- Define an associative array to hold an entire row from a table.

```
DECLARE
  TYPE dept_table_type IS TABLE OF
      departments%ROWTYPE INDEX PLS_INTEGER;
  dept_table dept_table_type;
  -- Each element of dept_table is a record
Begin
  SELECT * INTO dept_table(1) FROM departments
    WHERE department_id = 10;
  DBMS_OUTPUT.PUT_LINE(dept_table(1).department_id || ||
   dept_table(1).department_name || ||
   dept_table(1).manager_id);
END;
/
```

Results | Script Output | Exp

```
anonymous block completed
10 Administration 200
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 18

INDEX BY Table of Records Option: Example 2

```
DECLARE
  TYPE emp_table_type IS TABLE OF
    employees%ROWTYPE INDEX BY PLS_INTEGER;
  my_emp_table  emp_table_type;
  max_count        NUMBER(3):= 104;
BEGIN
  FOR i IN 100..max_count
  LOOP
   SELECT * INTO my_emp_table(i) FROM employees
   WHERE employee_id = i;
  END LOOP;
  FOR i IN my_emp_table.FIRST..my_emp_table.LAST
  LOOP
    DBMS_OUTPUT.PUT_LINE(my_emp_table(i).last_name);
  END LOOP;
END;
/
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 19

Summary of Collection Types
Associative array
Index by
PLS_INTEGER
1 2 3 4 5 6
a f i o t v
Index by
VARCHAR2
Nested table
Varray
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING
Copyright © Capgemini 2015. All Rights Reserved
20

Summary

- In this lesson you have learnt:
- Collection elements like
  - PLSQL Tables
  - Nested tables
  - Varrays
  - Associative Arrays

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 21

Add the notes here.

Review Question

- List PLSQL collections elements.
- Nested Table is known as a sparse collection because a nested table can contain empty elements.
  - True/False
- _____ can be opened on the server and passed to the client as a unit rather than fetching one row at a time.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 22

Add the notes here.

Advanced PLSQL

Lesson 05: Oracle 11g features

Lesson Objectives

In this lesson you will learn

- Oracle 11g Features
  - Use of sequence
  - Compound Triggers
  - Subprogram Inlining
  - PL/SQL Continue statement
  - PL/SQL Compiler Warnings

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 2

5.1 Use of Sequence

- A “Sequence” is an object, which can be used to generate sequential numbers.
- A Sequence is used to fill up columns, which are declared as UNIQUE or PRIMARY KEY.
- A Sequence uses “NEXTVAL” to retrieve the next value in the sequence order.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 3

Creating a Sequence

- Here is one more example of sequence:
  - s1 will generate numbers 1,2,3....,10000, and then stop.

```
CREATE SEQUENCE  s1
        INCREMENT BY 1
        START WITH 1
        MAXVALUE 10000
        NOCYCLE ;
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 4

NEXTVAL and CURRVAL pseudo columns

- NEXTVAL returns the next available sequence value.
  - It returns a unique value every time it is referenced, even for different users.
- CURRVAL obtains the current sequence value.
- NEXTVAL must be issued for the Sequence before CURRVAL can be referenced.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 5

Characteristics of Sequence

- Characteristics of a Sequence:
  - Caching the Sequence values in memory to give faster access to those Sequence values
- Gaps in Sequence values can occur when:
  - a rollback occurs
  - the system crashes
  - a Sequence is used in another table
- Viewing the next available value by querying the USER_SEQUENCES table, when the sequence is created with NOCACHE

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 5

Drop a Sequence

- A Sequence can be removed from the data dictionary by using the DROP SEQUENCE statement.
- Once removed, the Sequence can no longer be referenced.

```
DROP SEQUENCE dept_deptid_seq;
Sequence dropped.
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 7

Using Sequences in PL/SQL Expressions

- Starting in 11g:

```
DECLARE
  v_new_id NUMBER;
BEGIN
  v_new_id := my_seq.NEXTVAL;
END;
/
```

- Before 11g:

```
DECLARE
  v_new_id NUMBER;
BEGIN
  SELECT my_seq.NEXTVAL INTO v_new_id FROM Dual;
END;
/
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 8

Accessing Sequence ValuesIn Oracle Database 11*g*, you can use the NEXTVAL and CURRVAL pseudocolumns in any PL/SQL context, where an expression of the NUMBER data type may legally appear. Although the old style of using a SELECT statement to query a sequence is still valid, it is recommended that you do not use it.Before Oracle Database 11*g*, you were forced to write a SQL statement in order to use a sequence object value in a PL/SQL subroutine. Typically, you would write a SELECT statement to reference the pseudocolumns of NEXTVAL and CURRVAL to obtain a sequence number. This method created a usability problem. In Oracle Database 11*g*, the limitation of forcing you to write a SQL statement to retrieve a sequence value is eliminated. With the sequence enhancement feature:Sequence usability is improved.The developer has to type less.The resulting code is clearer

5.2 Compound Triggers

- A single trigger on a table that allows you to specify actions for each of the following four timing points:
  - Before the firing statement
  - Before each row that the firing statement affects
  - After each row that the firing statement affects
  - After the firing statement

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 9

Working with Compound Triggers

- The compound trigger body supports a common PL/SQL state that the code for each timing point can access.
- The compound trigger common state is:
  - Established when the triggering statement starts
  - Destroyed when the triggering statement completes
- A compound trigger has a declaration section and a section for each of its timing points.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 10

The Benefits of Using a Compound Trigger

- You can use compound triggers to:
  - Program an approach where you want the actions you implement for the various timing points to share common data.
  - Accumulate rows destined for a second table so that you can periodically bulk-insert them
  - Avoid the mutating-table error (ORA-04091)by allowing rows destined for a second table to accumulate and then bulk-inserting them

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 11

Timing-Point Sections of a Table Compound Trigger

- A compound trigger defined on a table has one or more of the following timing-point sections. Timing-point sections must appear in the order shown in the table.

Timing Point	Compound Trigger Section
Before the triggering statement executes	`BEFORE` statement
After the triggering statement executes	`AFTER` statement
Before each row that the triggering statement affects	`BEFORE EACH ROW`
After each row that the triggering statement affects	`AFTER EACH ROW`

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 12

Compound Trigger Structure for Tables

```
CREATE OR REPLACE TRIGGER schema.trigger
FOR dml_event_clause ON schema.table
COMPOUND TRIGGER
  -- Initial section
    -- Declarations
    -- Subprograms

  -- Optional section
  BEFORE STATEMENT IS ...;

  -- Optional section
  AFTER STATEMENT IS ...;

  -- Optional section
  BEFORE EACH ROW IS ...;

  -- Optional section
  AFTER EACH ROW IS ...;
```

1

2

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 13

Compound Trigger Structure for Views

```
CREATE OR REPLACE TRIGGER schema.trigger
FOR dml_event_clause ON schema.view
COMPOUND TRIGGER

-- Initial section
  -- Declarations
  -- Subprograms

-- Optional section (exclusive)
INSTEAD OF EACH ROW IS
...;
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 14

Compound Trigger Restrictions

- A compound trigger must be a DML trigger and defined on either a table or a view.
- The body of a compound trigger must be compound trigger block, written in PL/SQL.
- A compound trigger body cannot have an initialization block; therefore, it cannot have an exception section.
- An exception that occurs in one section must be handled in that section. It cannot transfer control to another section.
- `:OLD` and `:NEW` cannot appear in the declaration, `BEFORE STATEMENT`, or the `AFTER STATEMENT` sections.
- Only the `BEFORE EACH ROW` section can change the value of `:NEW`.
- The firing order of compound triggers is not guaranteed unless you use the `FOLLOWS` clause.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 15

Trigger Restrictions on Mutating Tables

- A mutating table is:
  - A table that is being modified by an UPDATE, DELETE, or INSERT statement, or
  - A table that might be updated by the effects of a DELETE CASCADE constraint
- The session that issued the triggering statement cannot query or modify a mutating table.
- This restriction prevents a trigger from seeing an inconsistent set of data.
- This restriction applies to all triggers that use the FOR EACH ROW clause.
- Views being modified in the INSTEAD OF triggers are not considered mutating.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 16

Mutating Table: Example

```
CREATE OR REPLACE TRIGGER check_salary
  BEFORE INSERT OR UPDATE OF salary, job_id
  ON employees
  FOR EACH ROW
  WHEN (NEW.job_id <> 'AD_PRES')
DECLARE
  v_minsalary employees.salary%TYPE;
  v_maxsalary employees.salary%TYPE;
BEGIN
  SELECT MIN(salary), MAX(salary)
   INTO v_minsalary, v_maxsalary
   FROM employees
   WHERE job_id = :NEW.job_id;
  IF :NEW.salary < v_minsalary OR :NEW.salary > v_maxsalary THEN
    RAISE_APPLICATION_ERROR(-20505,'Out of range');
  END IF;
END;
/
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 17

Mutating Table: Example

```
UPDATE employees
SET salary = 3400
WHERE last_name = 'Stiles';
```

```
TRIGGER check_salary Compiled.

Error starting at line 1 in command:
UPDATE employees
 SET salary = 3400
 WHERE last_name = 'Stiles'
Error report:
SQL Error: ORA-04091: table ORA42.EMPLOYEES is mutating, trigger/function may not see it
ORA-06512: at "ORA42.CHECK_SALARY", line 5
ORA-04088: error during execution of trigger 'ORA42.CHECK_SALARY'
04091. 00000 -  "table %s.%s is mutating, trigger/function may not see it"
*Cause:    A trigger (or a user defined plsql function that is referenced in
           this statement) attempted to look at (or modify) a table that was
           in the middle of being modified by the statement which fired it.
*Action:   Rewrite the trigger (or function) so it does not read that table.
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 18

Using a Compound Trigger to Resolve the Mutating Table Error

```
CREATE OR REPLACE TRIGGER check_salary
  FOR INSERT OR UPDATE OF salary, job_id
  ON employees
  WHEN (NEW.job_id <> 'AD_PRES')

  TYPE salaries_t          IS TABLE OF employees.salary%TYPE;
  min_salaries             salaries_t;
  max_salaries             salaries_t;

  TYPE department_ids_t      IS TABLE OF employees.department_id%TYPE;
  department_ids             department_ids_t;

  TYPE department_salaries_t  IS TABLE OF employees.salary%TYPE
                              INDEX BY VARCHAR2(80);
  department_min_salaries     department_salaries_t;
  department_max_salaries     department_salaries_t;

  -- example continues on next slide
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 19

Using a Compound Trigger to Resolve the Mutating Table Error

```
. . .
BEFORE STATEMENT IS
  BEGIN
    SELECT MIN(salary), MAX(salary), NVL(department_id, -1)
     BULK COLLECT INTO  min_Salaries, max_salaries, department_ids
    FROM    employees
    GROUP BY department_id;
    FOR j IN 1..department_ids.COUNT() LOOP
      department_min_salaries(department_ids(j)) := min_salaries(j);
      department_max_salaries(department_ids(j)) := max_salaries(j);
    END LOOP;
END BEFORE STATEMENT;

AFTER EACH ROW IS
  BEGIN
    IF :NEW.salary < department_min_salaries(:NEW.department_id)
      OR :NEW.salary > department_max_salaries(:NEW.department_id) THEN
      RAISE_APPLICATION_ERROR(-20505,'New Salary is out of acceptable
                                  range');
    END IF;
  END AFTER EACH ROW;
END check_salary;
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 20

5.3 Subprogram Inlining

- Inlining in PL/SQL is an optimization where the PL/SQL compiler replaces calls to subprograms (functions and procedures) with the code of the subprograms.
- A performance gain is almost always achieved because calling a subprogram requires the creation of a callstack entry, possible creation of copies of variables, and the handling of return values.
- With inlining, those steps are avoided.
- Automatic subprogram inlining can reduce the overheads associated with calling subprograms, whilst leaving your original source code in its normal modular state.
  - The process of subprogram inlining is controlled by the PLSQL_OPTIMIZE_LEVEL parameter and the INLINE pragma

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 21

Subprogram Inlining

- As an example, consider the following, where the PL/SQL optimization level is explicitly set to 2, and then an anonymous PL/SQL block is executed:

```
ALTER SESSION SET PLSQL_OPTIMIZE_LEVEL=2;
DECLARE
PROCEDURE wr(pStr IN VARCHAR2) IS
BEGIN
dbms_output.put_line(rpad(lpad(pStr,15,'='),30,'='));
dbms_output.put_line(dbms_utility.format_call_stack());
END;
BEGIN
wr('At Start');
wr('Done');
END;
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 22

5.4 PL/SQL Continue statement

- Many programming languages have a continue statement that can be used inside a looping structure to cause the next iteration of the loop to occur, rather than process the remainder of the current loop iteration
- In complex looping structures this can save some processing time
- Oracle Database 11g now provides the continue statement for this very purpose
- Syntactically it is the same as the EXIT statement, and it allows for an optional WHEN clause and label

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 23

Continue Example

```
Declare
v_counter number:=0;
begin
for count in 1..10
LOOP
v_counter:=v_counter + 1;
dbms_output.put_line('v_Counter = '||v_counter);
continue when v_counter > 5;
v_counter:=v_counter+1;
END LOOP;
end;
/
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 24

5.5 PL/SQL Compiler Warnings

- The PL/SQL compiler now warns the user if the WHEN OTHERS exception handler does not raise an error.
- The user should include RAISE or RAISE_APPLICATION_ERROR to indicate the exact exception.
- Set the parameter
  - PLSQL_WARNINGS='enable:all';

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 25

Summary

In this lesson you have learnt:

- Oracle 11g features such as
  - Use of sequence
  - Compound Triggers
  - Subprogram Inlining
  - PL/SQL Continue statement
  - PL/SQL Compiler Warnings

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 26

Add the notes here.

Review Question

- Question 1: The compound trigger makes it easier to program an approach where you want the actions you implement for the various timing points to share common data.
  - True/False
- Question 2:We can now directly assign the next and current value of a sequence as an assignment as in PL/SQL code using _____ and _____.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 27

Add the notes here.

Advanced PLSQL

Lesson 06: Best Practices of PLSQL and Dynamic SQL

Lesson Objectives

- In this lesson you will learn
- Best Practices of PLSQL code
  - Conditional Compilation
  - Using Selection Directives
  - Obfuscation
- Dynamic SQL

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 2

6.1.1: Best practices: Conditional compilation

Conditional Compilation

- Enables you to customize the functionality in a PL/SQL application without removing any source code:
  - Utilize the latest functionality with the latest database release or disable the new features to run the application against an older release of the database.
  - Activate debugging or tracing functionality in the development environment and hide that functionality in the application while it runs at a production site.

$IF, $THEN, $ELSE,
$ELSIF, $END, $$, $ERROR

Reserved preprocessor control tokens

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 3

How Does Conditional Compilation Work?

Selection directives:
Use the $IF token.

Inquiry directives:
Use the $$ token.

Error directives:
Use the $ERROR token.

DBMS_PREPROCESSO
R
package

DBMS_DB_VERSION
package

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 4

6.1.2: Best practices: Using Selection Directives

Using Selection Directives

```
$IF <Boolean-expression> $THEN Text
$ELSEIF <Boolean-expression> $THEN Text
. . .
$ELSE Text
$END
```

```
DECLARE
CURSOR cur IS SELECT employee_id FROM
employees WHERE
$IF myapp_tax_package.new_tax_code $THEN
  salary > 20000;
$ELSE
  salary > 50000;
$END
BEGIN
  OPEN cur;
. . .
END;
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 5

Using Predefined and User-Defined Inquiry Directives

PLSQL_CCFLAGS
PLSQL_CODE_TYPE
PLSQL_OPTIMIZE_LEVEL
PLSQL_WARNINGS
NLS_LENGTH_SEMANTICS
PLSQL_LINE
PLSQL_UNIT

Predefined inquiry directives

```
PLSQL_CCFLAGS = 'plsql_ccflags:true,debug:true,debug:0';
```

User-defined inquiry directives

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 6

The PLSQL_CCFLAGS Parameter and the Inquiry Directive

- Use the PLSQL_CCFLAGS parameter to control conditional compilation of each PL/SQL library unit independently.

```
PLSQL_CCFLAGS = '<v1>:<c1>,<v2>:<c2>,...,<vn>:<cn>'
```

```
ALTER SESSION SET
PLSQL_CCFLAGS = 'plsql_ccflags:true, debug:true, debug:0';
```

PLSQL_CCFLAGS initialization parameter

Inquiry directive

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING
Copyright © Capgemini 2015. All Rights Reserved

Displaying the PLSQL_CCFLAGS Initialization Parameter Setting

```
SELECT name, type, plsql_ccflags
FROM   user_plsql_object_settings
```

Results:

	NAME	TYPE	PLSQL_CCFLAGS
1	DEPT_PKG	PACKAGE	(null)
2	DEPT_PKG	PACKAGE BODY	(null)
3	TAXES_PKG	PACKAGE	(null)
4	TAXES_PKG	PACKAGE BODY	(null)
5	EMP_PKG	PACKAGE	(null)
6	EMP_PKG	PACKAGE BODY	(null)
7	SECURE_DML	PROCEDURE	(null)
8	SECURE_EMPLOYEES	TRIGGER	(null)
9	ADD_JOB_HISTORY	PROCEDURE	plsql_ccflags:true, debug:true, debug:0
10	UPDATE_JOB_HISTORY	TRIGGER	(null)

. . .

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 8

The PLSQL_CCFLAGS Parameter and the Inquiry Directive: Example

```
ALTER SESSION SET PLSQL_CCFLAGS = 'Tracing:true';
CREATE OR REPLACE PROCEDURE P IS
BEGIN
  $IF $$tracing $THEN
    DBMS_OUTPUT.PUT_LINE ('TRACING');
  $END
END P;
```

```
ALTER SESSION SET succeeded.
PROCEDURE P Compiled.
```

```
SELECT name, plsql_ccflags
FROM USER_PLSQL_OBJECT_SETTINGS
WHERE name = 'P';
```

Results:

	NAME	PLSQL_CCFLAGS
1	P	Tracing:true

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 9

Using Conditional Compilation Error Directives to Raise User-Defined Errors

```
$ERROR varchar2_static_expression $END
```

```
ALTER SESSION SET Plsql_CCFlags = ' Trace_Level:3 '
/ CREATE PROCEDURE P IS
BEGIN
  $IF    $$Trace_Level = 0 $THEN ...;
  $ELSIF $$Trace_Level = 1 $THEN ...;
  $ELSIF $$Trace_Level = 2 $THEN ...;
  $else $error 'Bad: '||$$Trace_Level $END –- error
                        -- directive
  $END -- selection directive ends
END P;
```

```
SHOW ERRORS
Errors for PROCEDURE P:
LINE/COL ERROR
-------- -----------------------------------
6/9      PLS-00179: $ERROR: Bad: 3
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 10

Using Static Expressions with Conditional Compilation

- Boolean static expressions:
- TRUE, FALSE, NULL, IS NULL, IS NOT NULL
- > , < , >= , <= , = , <>, NOT, AND, OR
- PLS_INTEGER static expressions:
- -2147483648 to 2147483647, NULL
- VARCHAR2 static expressions include:
- ||, NULL, TO_CHAR
- Static constants:

```
static_constant CONSTANT datatype := static_expression;
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 11

The DBMS_DB_VERSION Package: Boolean Constants

Capgemini CONSULTING.TECHNOLOGY.OUTSOURCING
Copyright © Capgemini 2015. All Rights Reserved 12

The DBMS_DB_VERSION Package Constants

Name	Value	Description
VER_LE_9	FALSE	Version <= 9.
VER_LE_9_1	FALSE	Version <= 9 and release <= 1.
VER_LE_9_2	FALSE	Version <= 9 and release <= 2.
VER_LE_10	TRUE	Version <= 10.
VER_LE_10_1	FALSE	Version <= 10 and release <= 1.
VER_LE_10_2	TRUE	Version <=10 and release <= 2.
VER_LE_11	FALSE	Version <= 11.
VER_LE_11_1	TRUE	Version <=11 and release <= 1.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 13

Using Conditional Compilation with Database Versions: Example

```
ALTER SESSION SET PLSQL_CCFLAGS = 'my_debug:FALSE, my_tracing:FALSE';
CREATE PACKAGE my_pkg AS
  SUBTYPE my_real IS
  -- Check the database version, if >= 10g, use BINARY_DOUBLE data type,
  -- else use NUMBER data type
    $IF DBMS_DB_VERSION.VERSION < 10 $THEN     NUMBER;
    $ELSE BINARY_DOUBLE;
    $END
  my_pi my_real; my_e my_real;
END my_pkg;
/
CREATE PACKAGE BODY my_pkg AS
BEGIN
  $IF DBMS_DB_VERSION.VERSION < 10 $THEN
      my_pi := 3.14016408289008292431940027343666863227;
      my_e  := 2.71828182845904523536028747135266249775;
  $ELSE
      my_pi := 3.14016408289008292431940027343666863227d;
      my_e  := 2.71828182845904523536028747135266249775d;
  $END
END my_pkg;
/
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 14

Using Conditional Compilation with Database Versions: Example

```
CREATE OR REPLACE PROCEDURE circle_area(p_radius my_pkg.my_real) IS
  v_my_area my_pkg.my_real;
  v_my_datatype VARCHAR2(30);
BEGIN
  v_my_area := my_pkg.my_pi * p_radius * p_radius;
  DBMS_OUTPUT.PUT_LINE('Radius: ' || TO_CHAR(p_radius)
                  || ' Area: ' || TO_CHAR(v_my_area) );
  $IF $$my_debug $THEN   if my_debug is TRUE, run some debugging code
    SELECT DATA_TYPE INTO v_my_datatype FROM USER_ARGUMENTS
      WHERE OBJECT_NAME = 'CIRCLE_AREA' AND ARGUMENT_NAME = 'P_RADIUS';
    DBMS_OUTPUT.PUT_LINE('Datatype of the RADIUS argument is: ' || v_my_datatype);
  $END
END; /
```

```
PROCEDURE circle_area(p_radius Compiled.
```

```
CALL circle_area(50); -- Using Oracle Database 11g Release 2
```

```
CALL circle_area(50) succeeded.
Radius: 5.0E+001 Area: 7.8504102072252062E+003
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 15

Using DBMS_PREPROCESSOR Procedures to Print or Retrieve Source Text

```
CALL
    DBMS_PREPROCESSOR.PRINT_POST_PROCESSED_SOURCE('PACK
    AGE', 'ORA61', 'MY_PKG');
```

Results | Script Output | Explain | Autotrace | DBMS Output | OWA Output

```
CALL DBMS_PREPROCESSOR.PRINT_POST_PROCESSED_SOURCE('PACKAGE', succeeded.
PACKAGE my_pkg AS
  SUBTYPE my_real IS

-- Check the database version, if >= 10g, use BINARY_DOUBLE data type,
-- else use NUMBER data type


              BINARY_DOUBLE;

  my_pi my_real; my_e my_real;
END my_pkg;
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 16

6.1.3: Best practices: Using Selection Directives

Obfuscation

- Obfuscation (or wrapping) of a PL/SQL unit is the process of hiding the PL/SQL source code.
- Wrapping can be done with the wrap utility and `DBMS_DDL` subprograms.
- The Wrap utility is run from the command line and it processes an input SQL file, such as a SQL*Plus installation script.
- The `DBMS_DDL` subprograms wrap a single PL/SQL unit, such as a single `CREATE PROCEDURE` command, that has been generated dynamically.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 17

Benefits of Obfuscating

- It prevents others from seeing your source code.
- Your source code is not visible through the USER_SOURCE, ALL_SOURCE, or DBA_SOURCE data dictionary views.
- SQL*Plus can process the obfuscated source files.
- The Import and Export utilities accept wrapped files.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 18

What's New in Dynamic Obfuscating Since Oracle 10*g*?

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 19

Nonobfuscated PL/SQL Code: Example

```
SET SERVEROUTPUT ON
BEGIN -- The ALL_SOURCE view family shows source code
  EXECUTE IMMEDIATE '
    CREATE OR REPLACE PROCEDURE P1 IS
    BEGIN
      DBMS_OUTPUT.PUT_LINE ("I am not wrapped");
    END P1;
  ';
END;
/
CALL p1();
```

```
anonymous block completed
CALL p1() succeeded.
I'm not wrapped
```

```
SELECT text FROM user_source
WHERE name = 'P1' ORDER BY line;
```

	TEXT
1	PROCEDURE P1 IS
2	BEGIN
3	DBMS_OUTPUT.PUT_LINE ('I am not wrapped');
4	END P1;

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 20

Obfuscated PL/SQL Code: Example

```
BEGIN -- ALL_SOURCE view family obfuscates source code
  DBMS_DDL.CREATE_WRAPPED ( '
    CREATE OR REPLACE PROCEDURE P1 IS
    BEGIN
      DBMS_OUTPUT.PUT_LINE ("I am wrapped now");
    END P1;
  ' );
END;
/
CALL p1();
```

```
anonymous block completed
call p1() succeeded.
I am wrapped now
```

```
SELECT text FROM user_source
WHERE name = 'P1' ORDER BY line;
```

```
TEXT
--------------------------------------------------------------------------------
PROCEDURE P1 wrapped
a000000
b2
abcd
. . .
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 21

Dynamic Obfuscation: Example

```
SET SERVEROUTPUT ON

DECLARE
c_code CONSTANT VARCHAR2(32767) :=
' CREATE OR REPLACE PROCEDURE new_proc AS
  v_VDATE  DATE;
  BEGIN
    v_VDATE := SYSDATE;
    DBMS_OUTPUT.PUT_LINE(v_VDATE) ;
  END; ' ;
BEGIN
  DBMS_DDL.CREATE_WRAPPED (c_CODE);
END;
/
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 22

The PL/SQL Wrapper Utility

- The PL/SQL wrapper is a stand-alone utility that hides application internals by converting PL/SQL source code into portable object code.
- Wrapping has the following features:
  - Platform independence
  - Dynamic loading
  - Dynamic binding
  - Dependency checking
  - Normal importing and exporting when invoked

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 23

Running the Wrapper Utility

```
WRAP INAME=input_file_name [ONAME=output_file_name]
```

- Do not use spaces around the equal signs.
- The INAME argument is required.
- The default extension for the input file is .sql, unless it is specified with the name.
- The ONAME argument is optional.
- The default extension for output file is .plb, unless specified with the ONAME argument.

Examples

```
WRAP INAME=demo_04_hello.sql
WRAP INAME=demo_04_hello
WRAP INAME=demo_04_hello.sql ONAME=demo_04_hello.plb
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 24

Results of Wrapping

```
-- Original PL/SQL source code in input file:

CREATE PACKAGE banking IS
  min_bal := 100;
  no_funds EXCEPTION;
     ...
     END banking;
     /
```

```
-- Wrapped code in output file:

CREATE PACKAGE banking
     wrapped

012abc463e ...

/
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 25

DBMS_DDL Package Versus the Wrap Utility

Functionality	DBMS_DDL	Wrap Utility
Code obfuscation	Yes	Yes
Dynamic Obfuscation	Yes	No
Obfuscate multiple programs at a time	No	Yes

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 26

Guidelines for Wrapping

- You must wrap only the package body, not the package specification.
- The wrapper can detect syntactic errors but cannot detect semantic errors.
- The output file should not be edited. You maintain the original source code and wrap again as required.
- To ensure that all the important parts of your source code are obfuscated, view the wrapped file in a text editor before distributing it.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 27

6.1.2: Dynamic SQL code

Dynamic SQL

- Use dynamic SQL to create a SQL statement whose structure may change during run time. Dynamic SQL:
  - Is constructed and stored as a character string, string variable, or string expression within the application
  - Is a SQL statement with varying column data, or different conditions with or without placeholders (bind variables)
  - Enables DDL, DCL, or session-control statements to be written and executed from PL/SQL
  - Is executed with Native Dynamic SQL statements or the DBMS_SQL package

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 28

Using Dynamic SQL

- Use dynamic SQL when the full text of the dynamic SQL statement is unknown until run time; therefore, its syntax is checked at *run time* rather than at *compile time*.
- Use dynamic SQL when one of the following items is unknown at precompile time:
  - Text of the SQL statement such as commands, clauses, and so on
  - The number and data types of host variables
  - References to database objects such as tables, columns, indexes, sequences, usernames, and views
- Use dynamic SQL to make your PL/SQL programs more general and flexible.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 29

Native Dynamic SQL (NDS)

- Provides native support for dynamic SQL directly in the PL/SQL language.
- Provides the ability to execute SQL statements whose structure is unknown until execution time.
- If the dynamic SQL statement is a SELECT statement that returns multiple rows, NDS gives you the following choices:
- Use the EXECUTE IMMEDIATE statement with the BULK COLLECT INTO clause
- Use the OPEN-FOR, FETCH, and CLOSE statements
- In Oracle Database 11g, NDS supports statements larger than 32 KB by accepting a CLOB argument.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 30

Using the EXECUTE IMMEDIATE Statement

- Use the EXECUTE IMMEDIATE statement for NDS or PL/SQL anonymous blocks:

```
EXECUTE IMMEDIATE dynamic_string
 [INTO {define_variable
    [, define_variable] ... | record}]
 [USING [IN|OUT|IN OUT] bind_argument
    [, [IN|OUT|IN OUT] bind_argument] ... ];
```

- INTO is used for single-row queries and specifies the variables or records into which column values are retrieved.
- USING is used to hold all bind arguments. The default parameter mode is IN.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 31

Available Methods for Using NDS

Method #	SQL Statement Type	NDS SQL Statements Used
Method 1	***Non-query without*** host variables	EXECUTE IMMEDIATE without the USING and INTO clauses
Method 2	***Non-query with*** known number of input host variables	EXECUTE IMMEDIATE with a USING clause
Method 3	***Query with known*** number of select-list items and input host variables	EXECUTE IMMEDIATE with the USING and INTO clauses
Method 4	***Query with unknown*** number of select-list items or input host variables	Use the DBMS_SQL package

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 32

Dynamic SQL with a DDL Statement: Examples

```
-- Create a table using dynamic SQL

CREATE OR REPLACE PROCEDURE create_table(
  p_table_name VARCHAR2, p_col_specs  VARCHAR2) IS
BEGIN
  EXECUTE IMMEDIATE 'CREATE TABLE ' || p_table_name ||
                    ' (' || p_col_specs || ')';
END;
/
```

```
-- Call the procedure

BEGIN
  create_table('EMPLOYEE_NAMES',
      'id NUMBER(4) PRIMARY KEY, name VARCHAR2(40)');
END;
/
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 33

Dynamic SQL with DML Statements

```
-- Delete rows from any table:
CREATE FUNCTION del_rows(p_table_name VARCHAR2)
RETURN NUMBER IS
BEGIN
  EXECUTE IMMEDIATE 'DELETE FROM '|| p_table_name;
  RETURN SQL%ROWCOUNT;
END;
/
BEGIN DBMS_OUTPUT.PUT_LINE(
  del_rows('EMPLOYEE_NAMES')|| ' rows deleted.');
END;
/
```

```
-- Insert a row into a table with two columns:
CREATE PROCEDURE add_row(p_table_name VARCHAR2,
   p_id NUMBER, p_name VARCHAR2) IS
BEGIN
  EXECUTE IMMEDIATE 'INSERT INTO '|| p_table_name ||
        ' VALUES (:1, :2)' USING p_id, p_name;
END;
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 34

Dynamic SQL with a Single-Row Query: Example

```
CREATE FUNCTION get_emp( p_emp_id NUMBER )
RETURN employees%ROWTYPE IS
  v_stmt VARCHAR2(200);
  v_emprec employees%ROWTYPE;
BEGIN
  v_stmt := 'SELECT * FROM employees ' ||
            'WHERE employee_id = :p_emp_id';
  EXECUTE IMMEDIATE v_stmt INTO v_emprec USING p_emp_id ;
  RETURN v_emprec;
END;
/
DECLARE
  v_emprec employees%ROWTYPE := get_emp(100);
BEGIN
  DBMS_OUTPUT.PUT_LINE('Emp: '|| v_emprec.last_name);
END;
/
```

```
FUNCTION get_emp(p_emp_id Compiled.
anonymous block completed
Emp: King
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 35

Executing a PL/SQL Anonymous Block Dynamically

```
CREATE FUNCTION annual_sal( p_emp_id NUMBER)
RETURN NUMBER IS
  v_plsql varchar2(200) :=
    'DECLARE '||
    ' rec_emp employees%ROWTYPE; '||
    'BEGIN '||
    ' rec_emp := get_emp(:empid); ' ||
    ' :res := rec_emp.salary  * 12; ' ||
    'END;';
  v_result NUMBER;
BEGIN
 EXECUTE IMMEDIATE v_plsql
        USING IN p_emp_id, OUT v_result;
  RETURN v_result;
END;
/
EXECUTE DBMS_OUTPUT.PUT_LINE(annual_sal(100))
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 36

Using Native Dynamic SQL to Compile PL/SQL Code

- Compile PL/SQL code with the ALTER statement:
  - ALTER PROCEDURE name COMPILE
  - ALTER FUNCTION name COMPILE
  - ALTER PACKAGE name COMPILE SPECIFICATION
  - ALTER PACKAGE name COMPILE BODY

```
CREATE PROCEDURE compile_plsql(p_name VARCHAR2,
 p_plsql_type VARCHAR2, p_options VARCHAR2 := NULL) IS
  v_stmt varchar2(200) := 'ALTER '|| p_plsql_type ||
                          ' '|| p_name || ' COMPILE';
BEGIN
 IF p_options IS NOT NULL THEN
   v_stmt := v_stmt || ' ' || p_options;
 END IF;
 EXECUTE IMMEDIATE v_stmt;
END;
/
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 37

Using the DBMS_SQL Package

- The DBMS_SQL package is used to write dynamic SQL in stored procedures and to parse DDL statements.
- You must use the DBMS_SQL package to execute a dynamic SQL statement that has an unknown number of input or output variables, also known as Method 4.
- In most cases, NDS is easier to use and performs better than DBMS_SQL except when dealing with Method 4.
- For example, you must use the DBMS_SQL package in the following situations:
  - You do not know the SELECT list at compile time
  - You do not know how many columns a SELECT statement will return, or what their data types will be

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 38

Using the DBMS_SQL Package Subprograms

- Examples of the package procedures and functions:
  - OPEN_CURSOR
  - PARSE
  - BIND_VARIABLE
  - EXECUTE
  - FETCH_ROWS
  - CLOSE_CURSOR

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 39

Using DBMS_SQL with a DML Statement: Deleting Rows

```
CREATE OR REPLACE FUNCTION delete_all_rows
  (p_table_name  VARCHAR2) RETURN NUMBER IS
   v_cur_id      INTEGER;
   v_rows_del    NUMBER;
BEGIN
  v_cur_id := DBMS_SQL.OPEN_CURSOR;
  DBMS_SQL.PARSE(v_cur_id,
    'DELETE FROM '|| p_table_name, DBMS_SQL.NATIVE);
  v_rows_del := DBMS_SQL.EXECUTE (v_cur_id);
  DBMS_SQL.CLOSE_CURSOR(v_cur_id);
  RETURN v_rows_del;
END;
/
```

```
CREATE TABLE temp_emp AS SELECT * FROM employees;
BEGIN
 DBMS_OUTPUT.PUT_LINE('Rows Deleted: ' ||
delete_all_rows('temp_emp'));
END;
/
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 40

Using DBMS_SQL with a Parameterized DML Statement

```
CREATE PROCEDURE insert_row (p_table_name VARCHAR2,
 p_id VARCHAR2, p_name VARCHAR2, p_region NUMBER) IS
  v_cur_id       INTEGER;
  v_stmt         VARCHAR2(200);
  v_rows_added   NUMBER;
BEGIN
  v_stmt := 'INSERT INTO '|| p_table_name ||
           ' VALUES (:cid, :cname, :rid)';
  v_cur_id := DBMS_SQL.OPEN_CURSOR;
  DBMS_SQL.PARSE(v_cur_id, v_stmt, DBMS_SQL.NATIVE);
  DBMS_SQL.BIND_VARIABLE(v_cur_id, ':cid', p_id);
  DBMS_SQL.BIND_VARIABLE(v_cur_id, ':cname', p_name);
  DBMS_SQL.BIND_VARIABLE(v_cur_id, ':rid', p_region);
  v_rows_added := DBMS_SQL.EXECUTE(v_cur_id);
  DBMS_SQL.CLOSE_CURSOR(v_cur_id);
  DBMS_OUTPUT.PUT_LINE(v_rows_added||' row added');
END;
/
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 41

Summary

- In this lesson you have learnt:
  - Best Practices of PLSQL code
  - Conditional Compilation
  - Using Selection Directives
  - Obfuscation
- Dynamic SQL

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 42

Add the notes here.

Review Question

- Question 1:_____ unit is the process of hiding the PL/SQL source code.

- Question 2: NDS provides the ability to execute SQL statements whose structure is unknown until execution time.
  - True/False

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved. 43

Add the notes here.

PLSQL

©2016 Capgemini. All rights reserved.
The information contained in this document is proprietary and confidential. For Capgemini only.

Document Revision History

Date	Revision No.	Author	Summary of Changes
July 2016	1.0	Shraddha Jadhav and Rahul Kulkarni	Integration refinements as per integrated RDBMS LOT Structure

©2016 Capgemini. All rights reserved.
The information contained in this document is proprietary and confidential. For Capgemini only.

Table of Contents

Document Revision History *3*

Table of Contents *3*

Getting Started *4*

Overview ..... ***Error! Bookmark not defined.***

Setup Checklist for Programming Foundation with Pseudocode for All LOTs ..... ***Error! Bookmark not defined.***

Instructions ..... ***Error! Bookmark not defined.***

Lab 1. Introduction to PL/SQL and Cursors *7*

Lab 2. Exception Handling *9*

Lab 3. Procedures, Functions and Packages *11*

Lab 4. Case Study 1 ..... *15*

Lab 5. Case Study 2 ..... *17*

Lab 6. Triggers *19*

Lab 7. Oracle Supplied packages *20*

Lab 8. Design Considerations *22*

Lab 9. Different types of pragma *27*

Lab 10. Collection elements *30*

Lab 11. Oracle 11g features *31*

Lab 12. Best practices of PL SQL code, Dynamic SQL *33*

Appendices *35*

Appendix A: Programming Standards *38*

©2016 Capgemini. All rights reserved.
The information contained in this document is proprietary and confidential. For Capgemini only.

Getting Started

Overview

This lab book is a guided tour for learning Oracle 9i. It comprises 'To Do' assignments. Follow the steps provided and work out the 'To Do' assignments.

Setup Checklist for Oracle 9i

Here is what is expected on your machine in order for the lab to work.

Minimum System Requirements

- Intel Pentium 90 or higher (P166 recommended)
- Microsoft Windows 95, 98, or NT 4.0, 2k, XP.
- Memory: 32MB of RAM (64MB or more recommended)

Please ensure that the following is done:

- Oracle Client is installed on every machine
- Connectivity to Oracle Server

Instructions

- For all coding standards refer Appendix A. All lab assignments should refer coding standards.
- Create a directory by your name in drive <drive>. In this directory, create a subdirectory Oracle 9i_assgn. For each lab exercise create a directory as lab <lab number>.

Learning More (Bibliography if applicable)

- Oracle10g - SQL - Student Guide - Volume 1 by Oracle Press
- Oracle10g - SQL - Student Guide - Volume 2 by Oracle Press
- Oracle10g database administration fundamentals volume 1 by Oracle Press
- Oracle10g Complete Reference by Oracle Press
- Oracle10g SQL with an Introduction to PL/SQL by Lannes L. Morris-Murphy

©2016 Capgemini. All rights reserved.

The information contained in this document is proprietary and confidential. For Capgemini only.

Problem Statement / Case Study

Table descriptions

Emp

Name	Null?	Type
Empno	Not Null	Number(4)
Ename	-	Varchar2(10)
job		Varchar2(9)
mgr		Number(4)
Hiredate		Date
Sal	-	Number(7,2)
Comm	-	Number(7,2)
Deptno	-	Number(2)

Designation_Masters

Name	Null?	Type
Design_code	Not Null	Number(3)
Design_name		Varchar2(50)

Department_Masters

Name	Null?	Type
Dept_Code	Not Null	Number(2)
Dept_name		Varchar2(50)

Student_Masters

Name	Null?	Type
Student_Code	Not Null	Number(6)
Student_name	Not Null	Varchar2(50)
Dept_Code		Number(2)
Student_dob		Date
Student_Address		Varchar2(240)

©2016 Capgemini. All rights reserved.

The information contained in this document is proprietary and confidential. For Capgemini only.

Student_Marks

Name	Null?	Type
Student_Code		Number(6)
Student_Year	Not Null	Number
Subject1		Number(3)
Subject2		Number(3)
Subject3		Number(3)

Staff_Masters

Name	Null?	Type
Staff_code	Not Null	Number(8)
Staff_Name	Not Null	Varchar2(50)
Design_code		Number
Dept_code		Number
HireDate		Date
Staff_dob		Date
Staff_address		Varchar2(240)
Mgr_code		Number(8)
Staff_sal		Number (10,2)

Book_Masters

Name	Null?	Type
Book_Code	Not Null	Number(10)
Book_Name	Not Null	Varchar2(50)
Book_pub_year		Number
Book_pub_author	Not Null	Varchar2(50)

Book_Transactions

Name	Null?	Type
Book_Code		Number
Student_code		Number
Staff_code		Number
Book_Issue_date	Not Null	Date
Book_expected_return_date	Not Null	Date
Book_actual_return_date		Date

©2016 Capgemini. All rights reserved.

The information contained in this document is proprietary and confidential. For Capgemini only.

Lab 1. Introduction to PL/SQL and Cursors

Goals	The following set of exercises are designed to implement the following • PL/SQL variables and data types • Create, Compile and Run anonymous PL/SQL blocks • Usage of Cursors
Time	1hr 30 min

2.1

Identify the problems(if any) in the below declarations:

```
DECLARE
V_Sample1 NUMBER(2);
V_Sample2 CONSTANT NUMBER(2) ;
V_Sample3 NUMBER(2) NOT NULL ;
V_Sample4 NUMBER(2) := 50;
V_Sample5 NUMBER(2) DEFAULT 25;
```

Example 1: Declaration Block

2.2
The following PL/SQL block is incomplete.
Modify the block to achieve requirements as stated in the comments in the block.

```
DECLARE --outer block
var_num1 NUMBER := 5;
BEGIN

DECLARE --inner block
var_num1 NUMBER := 10;
BEGIN
DBMS_OUTPUT.PUT_LINE('Value for var_num1:' ||var_num1);
--Can outer block variable (var_num1) be printed here.If Yes,Print the same.
END;
--Can inner block variable(var_num1)  be printed here.If Yes,Print the same.
END;
```

Example 2: PL/SQL block

2.3. Write a PL/SQL block to retrieve all staff (code, name, salary) under specific department number and display the result. (Note: The Department_Code will be accepted from user. Cursor to be used.**)**

2.4. Write a PL/SQL block to increase the salary by 30 % or 5000 whichever minimum for a given Department_Code.

2.5. Write a PL/SQL block to generate the following report for a given Department code

©2016 Capgemini. All rights reserved.

The information contained in this document is proprietary and confidential. For Capgemini only.

Student_Code Sudent_Name Subject1 Subject2 Subject3 Total Percentage Grade

Note: Display suitable error massage if wrong department code has entered and if there is no student in the given department.

For Grade:
Student should pass in each subject individually (pass marks 60).
Percent >= 80 then grade= A
Percent >= 70 and < 80 then grade= B
Percent >= 60 and < 70 then grade= C
Else D

2.6. Write a PL/SQL block to retrieve the details of the staff belonging to a particular department. Department code should be passed as a parameter to the cursor.

©2016 Capgemini. All rights reserved.

The information contained in this document is proprietary and confidential. For Capgemini only.

Lab 2. Exception Handling

Goals	Implementing Exception Handling ,Analyzing and Debugging
Time	1 hr

3.1: Modify the programs created in Lab2 to implement Exception Handling

3.2 The following PL/SQL block attempts to calculate bonus of staff for a given MGR_CODE. Bonus is to be considered as twice of salary. Though Exception Handling has been implemented but block is unable to handle the same.

Debug and verify the current behavior to trace the problem.

```
DECLARE
V_BONUS V_SAL%TYPE;
V_SAL STAFF_MASTER.STAFF_SAL%TYPE;

BEGIN
SELECT STAFF_SAL INTO V_SAL
FROM STAFF_MASTER
WHERE MGR_CODE=100006;

V_BONUS:=2*V_SAL;
DBMS_OUTPUT.PUT_LINE('STAFF SALARY IS ' || V_SAL);
DBMS_OUTPUT.PUT_LINE('STAFF BONUS IS ' || V_BONUS);

EXCEPTION
WHEN NO_DATA_FOUND THEN
DBMS_OUTPUT.PUT_LINE('GIVEN CODE IS NOT VALID.ENTER VALID CODE');
END;
```

Example 3: PL/SQL block

3.3 Rewrite the above block to achieve the requirement.

3.4
Predict the output of the following block ? What corrections would be needed to make it more efficient?

©2016 Capgemini. All rights reserved.

The information contained in this document is proprietary and confidential. For Capgemini only.

```
BEGIN
    DECLARE
    fname emp.ename%TYPE;
    BEGIN
    SELECT ename INTO fname
    FROM emp
    WHERE 1=2;

    DBMS_OUTPUT.PUT_LINE('This statement will print');
    EXCEPTION
    WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Some inner block error');
    END;

EXCEPTION
WHEN NO_DATA_FOUND THEN
DBMS_OUTPUT.PUT_LINE('No data found in fname');

WHEN OTHERS THEN
DBMS_OUTPUT.PUT_LINE('Some outer block error');
END;
```

Example 4: PL/SQL Block with Exception Handling

3.5 Debug the above block to trace the flow of control.
Additionally one can make appropriate changes in Select statement defined in the block to check the flow.

3.6: Write a PL/SQL program to check for the commission for an employee no 7369. If no commission exists, then display the error message. Use Exceptions.

3.7: Write a PL/SQL block to drop any user defined table.

©2016 Capgemini. All rights reserved.

The information contained in this document is proprietary and confidential. For Capgemini only.

Lab 3. Procedures, Functions and Packages

Goals	The following set of exercises are designed to implement the following • Implement business logic using Database Programming like Procedures and Functions • Implement validations in Procedures and Functions • Working with Packages • Performance Tuning
Time	3.5 Hrs

Note: Procedures and functions should handle validations, pre-defined oracle server and user defined exceptions wherever applicable. Also use cursors wherever applicable.

4.1 Write a PL/SQL block to find the maximum salary of the staff in the given department. Note: Department code should be passed as parameter to the cursor.

4.2. Write a function to compute age. The function should accept a date and return age in years.

4.3. Write a procedure that accept staff code and update staff name to Upper case. If the staff name is null raise a user defined exception.

4.4 Write a procedure to find the manager of a staff. Procedure should return the following – Staff_Code, Staff_Name, Dept_Code and Manager Name.

4.5. Write a function to compute the following. Function should take Staff_Code and return the cost to company.
DA = 15% Salary, HRA= 20% of Salary, TA= 8% of Salary.
Special Allowance will be decided based on the service in the company.

< 1 Year	Nil
>=1 Year< 2 Year	10% of Salary
>=2 Year< 4 Year	20% of Salary
>4 Year	30% of Salary

4.6. Write a procedure that displays the following information of all staff

Staff_Name Department Name Designation Salary Status

Note: - Status will be (Greater, Lesser or Equal) respective to average salary of their own department. Display an error message Staff_Master table is empty if there is no matching record.

4.7. Write a procedure that accept Staff_Code and update the salary and store the old salary details in Staff_Master_Back (Staff_Master_Back has the same structure without any constraint) table.

Exp < 2 then no Update
Exp > 2 and < 5 then 20% of salary
Exp > 5 then 25% of salary

©2016 Capgemini. All rights reserved.

The information contained in this document is proprietary and confidential. For Capgemini only.

4.8. Create a procedure that accepts the book code as parameter from the user. Display the details of the students/staff that have borrowed that book and has not returned the same. The following details should be displayed

Student/Staff Code Student/Staff Name Issue Date Designation Expected Ret_Date

4.9. Write a package which will contain a procedure and a function.

Function: This function will return years of experience for a staff. This function will take the hiredate of the staff as an input parameter. The output will be rounded to the nearest year (1.4 year will be considered as 1 year and 1.5 year will be considered as 2 year).

Procedure: Capture the value returned by the above function to calculate the additional allowance for the staff based on the experience.
Additional Allowance = Year of experience x 3000
Calculate the additional allowance and store Staff_Code, Date of Joining, and Experience in years and additional allowance in Staff_Allowance table.

4.10. Write a procedure to insert details into Book_Transaction table. Procedure should accept the book code and staff/student code. Date of issue is current date and the expected return date should be 10 days from the current date. If the expected return date falls on Saturday or Sunday, then it should be the next working day.

4.11: Write a function named ‘get_total_records’, to pass the table name as a parameter, and get back the number of records that are contained in the table. Test your function with multiple tables.

4.12

Tune the following Oracle Procedure enabling to gain better performance.

Objective: The Procedure should update the salary of an employee and at the same time retrieve the employee's name and new salary into PL/SQL variables.

©2016 Capgemini. All rights reserved.

The information contained in this document is proprietary and confidential. For Capgemini only.

```
CREATE OR REPLACE PROCEDURE update_salary (emp_id NUMBER) IS
  v_name    VARCHAR2(15);
  v_newsal NUMBER;
BEGIN
  UPDATE emp_copy SET sal = sal * 1.1
  WHERE empno = emp_id;

  SELECT ename, sal INTO v_name, v_newsal
  FROM emp_copy
  WHERE empno = emp_id;

  DBMS_OUTPUT.PUT_LINE('Emp Name:' || v_name);
  DBMS_OUTPUT.PUT_LINE('Ename:' || v_newsal);
END;
```

Example 5: Oracle Procedure

4.13

The following procedure attempts to delete data from table passed as parameter.This procedure has compilation errors. Identify and correct the problem.

```
CREATE or REPLACE PROCEDURE gettable(table_name in varchar2) AS
BEGIN
DELETE FROM table_name;
END;
```

Example 6: Oracle Procedure

4.14

Write a procedure which prints the following report using procedure:

The procedure should take deptno as user input and appropriately print the emp details.

Also display :

Number of Employees,Total Salary,Maximum Salary,Average Salary

Note: The block should achieve the same without using Aggregate Functions.

Sample output for deptno 10 is shown below:

©2016 Capgemini. All rights reserved.

The information contained in this document is proprietary and confidential. For Capgemini only.

```
Employee Name :  CLARK
Employee Job : MANAGER
Employee Salary :  2450
Employee Comission :
************************************
Employee Name :  KING
Employee Job : PRESIDENT
Employee Salary :  5000
Employee Comission :
************************************
Employee Name :  MILLER
Employee Job : CLERK
Employee Salary :  1300
Employee Comission :
************************************
Number of Employees :  3
Total Salary :  8750
Maximum Salary :  5000
Average Salary :  2916.67
-----------------------------------
```

Figure 1 :Report

4.15: Write a query to view the list of all procedures ,functions and packages from the Data Dictionary.

©2016 Capgemini. All rights reserved.

The information contained in this document is proprietary and confidential. For Capgemini only.

Lab 4. Case Study 1

Goals	Implementation of Procedures/Functions ,Packages with Testing and Review
Time	2.5hrs

Consider the following tables for the case study.

Customer_Masters

Name	Null?	Type
Cust_Id	Not Null	Number(6)
Cust_Name	Not Null	Varchar2(20)
Address		Varchar2(50)
Date_of_acc_creation		Date
Customer_Type		Char(3)

Note: Customer type can be either IND or NRI

Account_Masters Table

Name	Null?	Type
Account_Number	Not Null	Number(6)
Cust_ID		Number(6)
Account_Type		Char(3)
Ledger_Balance		Number(10)

Note: Account type can be either Savings (SAV) or Salary (SAL) account.
For savings account minimum amount should be 5000.

Transaction_Masters

Name	Null?	Type
Transaction_Id	Not Null	Number(6)
Account_Number		Number(6)
Date_of_Transaction		Date
From_Account_Number	Not Null	Number(6)
To_Account_Number	Not Null	Number(6)
Amount	Not Null	Number(10)
Transaction_Type	Not Null	Char(2)

Note: Transaction type can be either Credit (CR) or Debit (DB).

Procedure and function should be written inside a package.
All validations should be taken care.

©2016 Capgemini. All rights reserved.

The information contained in this document is proprietary and confidential. For Capgemini only.

5.1 Create appropriate Test Cases for the case study followed up by Self/Peer to Peer Review and close any defects for the same.

5.2Write a procedure to accept customer name, address, and customer type and account type. Insert the details into the respective tables.

5.3. Write a procedure to accept customer id, amount and the account number to which the customer requires to transfer money. Following validations need to be done

- Customer id should be valid
- From account number should belong to that customer
- To account number cannot be null but can be an account which need not exist in account masters (some other account)
- Adequate balance needs to be available for debit

5.4 Ensure all the Test cases defined are executed. Have appropriate Self/Peer to Peer Code Review and close any defects for the same.

©2016 Capgemini. All rights reserved.

The information contained in this document is proprietary and confidential. For Capgemini only.

Lab 5. Case Study 2

Goals	Implementation of Procedures/Functions ,Packages with Testing and Review
Time	2.5hrs

Consider the following table (myEmp) structure for the case study

EmpNo Ename City Designation Salary

The following procedure accepts Task number and based on the same performs an appropriate task.

```
PROCEDURE run_task (task_number_in IN INTEGER)
IS
BEGIN
  IF task_number_in = 1
  THEN
    add_emp;
    --should add new emps in myEmp.
    --EmpNo should be inserted through Sequence.
    --All other data to be taken as parameters.Default location is Mumbai.
  END IF;
  IF task_number_in = 2
  THEN
    raise_sal;
    --should modify salary of an existing emp.
    --should take new salary and empno as input parameters
    --Should handle exception in case empno not found
    --upper limit of rasing salary is 30%. should raise exception appropriately
  END IF;
   IF task_number_in = 3
   THEN
    remove_emp;
          --should remove an existing emp
          --should take empno as parameter
      --Handle exception if empno not available
  END IF;
END run_task;
```

Example 7: Sample Oracle Procedure

©2016 Capgemini. All rights reserved.

The information contained in this document is proprietary and confidential. For Capgemini only.

However ,it has been observed the method adopted in above procedure is inefficient.

6.1

Create appropriate Test Cases for the case study followed up by Self/Peer to Peer Review and close any defects for the same.

6.2

Recreate the procedure (run_task) which is more efficient in performing the same.

6.3

Also, create relevant procedures (add_emp , raise_sal ,remove_emp)

with relevant logic (read comments)to verify the same.

6.4 Extend the above implementation using Packages

6.5) Ensure all the Test cases defined are executed. Have appropriate Self/Peer to Peer

Code Review and close any defects for the same.

©2016 Capgemini. All rights reserved.

The information contained in this document is proprietary and confidential. For Capgemini only.

Lab 6. Triggers

Goals	Triggers
Time	1 hr

1. Create a trigger called CHECK_SALARY_TRG on the staff_master table that fires before an INSERT or UPDATE operation on each row. The trigger must call the CHECK_SALARY procedure to carry out the business logic. The trigger should pass the new job ID and salary to the procedure parameters.

2. Update the CHECK_SALARY_TRG trigger to fire only when the job ID or salary values have actually changed.

    a. Implement the business rule using a WHEN clause to check whether the JOB_ID or SALARY values have changed.
    Note: Make sure that the condition handles the NULL in the OLD.column_name values if an INSERT operation is performed; otherwise, an insert operation will fail.
    b. Test the trigger by executing the EMP_PKG.ADD_EMPLOYEE procedure with the following parameter values: first_name='Eleanor', last name='Beh', email='EBEH', job='IT_PROG', sal=5000.

3. You are asked to prevent employees from being deleted during business hours.

    a. Write a statement trigger called DELETE_EMP_TRG on the EMPLOYEES table to prevent rows from being deleted during weekday business hours, which are from 9:00 a.m. to 6:00 p.m.

©2016 Capgemini. All rights reserved.

The information contained in this document is proprietary and confidential. For Capgemini only.

Lab 7. Oracle Supplied packages

Goals	• Understand oracle supplied packages
Time	1 Hours.

1.1. Create a procedure called EMPLOYEE_REPORT that generates an employee report in a file in the operating system, using the UTL_FILE package. The report should generate a list of employees who have exceeded the average salary of their departments.

a) Your program should accept two parameters. The first parameter is the output
directory. The second parameter is the name of the text file that is written.
Note: Use the directory location value UTL_FILE. Add an exception-handling section to handle errors that may be encountered when using the UTL_FILE package.

Solution :
-- The course has a directory alias provided called "REPORTS_DIR" that
-- is associated with the /home/oracle/labs/plpu/reports
-- physical directory. Use the directory alias name
-- in quotes for the first parameter to create a file in the appropriate directory.

©2016 Capgemini. All rights reserved.

The information contained in this document is proprietary and confidential. For Capgemini only.

```
CREATE OR REPLACE PROCEDURE employee_report(
  p_dir IN VARCHAR2, p_filename IN VARCHAR2) IS
  f UTL_FILE.FILE_TYPE;
  CURSOR cur_avg IS
    SELECT last_name, department_id, salary
    FROM employees outer
    WHERE salary > (SELECT AVG(salary)
                    FROM  employees inner
                    GROUP BY outer.department_id)
    ORDER BY department_id;
BEGIN
  f := UTL_FILE.FOPEN(p_dir, p_filename,'W');
  UTL_FILE.PUT_LINE(f, 'Employees who earn more than average salary:
');
  UTL_FILE.PUT_LINE(f, 'REPORT GENERATED ON ' ||SYSDATE);
  UTL_FILE.NEW_LINE(f);
  FOR emp IN cur_avg
  LOOP
    UTL_FILE.PUT_LINE(f,
    RPAD(emp.last_name, 30) || ' ' ||
    LPAD(NVL(TO_CHAR(emp.department_id,'9999'),'-'), 5) || ' ' ||
    LPAD(TO_CHAR(emp.salary, '$99,999.00'), 12));
  END LOOP;
  UTL_FILE.NEW_LINE(f);
  UTL_FILE.PUT_LINE(f, '*** END OF REPORT ***');
  UTL_FILE.FCLOSE(f);
END employee_report;
/
      EXECUTE employee_report('REPORTS_DIR','sal_rpt61.txt')
```

b). Invoke the program, using the second parameter with a name such as sal_rpt*xx*.txt, where *xx* represents your user number
(for example, 01, 15, and so on). The following is a sample output from the report file:

```
Employees who earn more than average salary:
REPORT GENERATED ON 26-FEB-04
Hartstein                 20     $13,000.00
Raphaely                  30     $11,000.00
Marvis                    40     $6,500.00
...
          *** END OF REPORT ***
```

Note: The data displays the employee's last name, department ID, and salary.

©2016 Capgemini. All rights reserved.

The information contained in this document is proprietary and confidential. For Capgemini only.

Lab 8. Design Considerations

Goals	• Understand Design Considerations
Time	1 Hours.

1.2. In the body of the package, define a private variable called emp_table based on the type defined in the specification to hold employee records. Implement the get_employees procedure to bulk fetch the data into the table.

```
CREATE OR REPLACE PACKAGE emp_pkg IS

  TYPE emp_tab_type IS TABLE OF employees%ROWTYPE;

  PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_email employees.email%TYPE,
    p_job employees.job_id%TYPE DEFAULT 'SA_REP',
    p_mgr employees.manager_id%TYPE DEFAULT 145,
    p_sal employees.salary%TYPE DEFAULT 1000,
    p_comm employees.commission_pct%TYPE DEFAULT 0,
    p_deptid employees.department_id%TYPE DEFAULT 30);

  PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_deptid employees.department_id%TYPE);

  PROCEDURE get_employee(
    p_empid IN employees.employee_id%TYPE,
    p_sal OUT employees.salary%TYPE,
    p_job OUT employees.job_id%TYPE);

  FUNCTION get_employee(p_emp_id
employees.employee_id%type)
    return employees%rowtype;

  FUNCTION get_employee(p_family_name
employees.last_name%type)
    return employees%rowtype;
```

1.3. Create a new procedure in the specification and body, called show_employees, that does not take arguments and displays the

©2016 Capgemini. All rights reserved.

The information contained in this document is proprietary and confidential. For Capgemini only.

contents of the private PL/SQL table variable (if any data exists).Hint: Use the print_employee procedure.

```
CREATE OR REPLACE PACKAGE emp_pkg IS


  TYPE emp_tab_type IS TABLE OF employees%ROWTYPE;

  PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_email employees.email%TYPE,
    p_job employees.job_id%TYPE DEFAULT 'SA_REP',
    p_mgr employees.manager_id%TYPE DEFAULT 145,
    p_sal employees.salary%TYPE DEFAULT 1000,
    p_comm employees.commission_pct%TYPE DEFAULT 0,
    p_deptid employees.department_id%TYPE DEFAULT 30);

  PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_deptid employees.department_id%TYPE);

  PROCEDURE get_employee(
    p_empid IN employees.employee_id%TYPE,
    p_sal OUT employees.salary%TYPE,
    p_job OUT employees.job_id%TYPE);

  FUNCTION get_employee(p_emp_id employees.employee_id%type)
    return employees%rowtype;

  FUNCTION get_employee(p_family_name employees.last_name%type)
    return employees%rowtype;

  PROCEDURE get_employees(p_dept_id employees.department_id%type);

  PROCEDURE init_departments;

  PROCEDURE print_employee(p_rec_emp employees%rowtype);

  PROCEDURE show_employees;
```

-- Package BODY

```
CREATE OR REPLACE PACKAGE BODY emp_pkg IS
  TYPE boolean_tab_type IS TABLE OF BOOLEAN
```

©2016 Capgemini. All rights reserved.

The information contained in this document is proprietary and confidential. For Capgemini only.

```
    INDEX BY BINARY_INTEGER;

  valid_departments boolean_tab_type;
  emp_table        emp_tab_type;

  FUNCTION valid_deptid(p_deptid IN
departments.department_id%TYPE)
    RETURN BOOLEAN;

  PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_email employees.email%TYPE,
    p_job employees.job_id%TYPE DEFAULT 'SA_REP',
    p_mgr employees.manager_id%TYPE DEFAULT 145,
    p_sal employees.salary%TYPE DEFAULT 1000,
    p_comm employees.commission_pct%TYPE DEFAULT 0,
    p_deptid employees.department_id%TYPE DEFAULT 30) IS
  BEGIN
    IF valid_deptid(p_deptid) THEN
      INSERT INTO employees(employee_id, first_name, last_name,
email,
        job_id, manager_id, hire_date, salary, commission_pct,
department_id)
      VALUES (employees_seq.NEXTVAL, p_first_name, p_last_name,
p_email,
        p_job, p_mgr, TRUNC(SYSDATE), p_sal, p_comm, p_deptid);
    ELSE
      RAISE_APPLICATION_ERROR (-20204, 'Invalid department ID.
Try again.');
    END IF;
  END add_employee;

  PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_deptid employees.department_id%TYPE) IS
    p_email employees.email%type;
  BEGIN
    p_email := UPPER(SUBSTR(p_first_name, 1,
1)||SUBSTR(p_last_name, 1, 7));
    add_employee(p_first_name, p_last_name, p_email, p_deptid =>
p_deptid);
  END;

  PROCEDURE get_employee(
    p_empid IN employees.employee_id%TYPE,
    p_sal OUT employees.salary%TYPE,
    p_job OUT employees.job_id%TYPE) IS
```

©2016 Capgemini. All rights reserved.

The information contained in this document is proprietary and confidential. For Capgemini only.

```
  BEGIN
    SELECT salary, job_id
    INTO p_sal, p_job
    FROM employees
    WHERE employee_id = p_empid;
  END get_employee;

  FUNCTION get_employee(p_emp_id employees.employee_id%type)
    return employees%rowtype IS
    rec_emp employees%rowtype;
  BEGIN
    SELECT * INTO rec_emp
    FROM employees
    WHERE employee_id = p_emp_id;
    RETURN rec_emp;
  END;

  FUNCTION get_employee(p_family_name
employees.last_name%type)
    return employees%rowtype IS
    rec_emp employees%rowtype;
  BEGIN
    SELECT * INTO rec_emp
    FROM employees
    WHERE last_name = p_family_name;
    RETURN rec_emp;
  END;

/* New get_employees procedure. */

  PROCEDURE get_employees(p_dept_id
employees.department_id%type) IS
  BEGIN
    SELECT * BULK COLLECT INTO emp_table
    FROM EMPLOYEES
    WHERE department_id = p_dept_id;
  END;

  PROCEDURE init_departments IS
  BEGIN
    FOR rec IN (SELECT department_id FROM departments)
    LOOP
      valid_departments(rec.department_id) := TRUE;
    END LOOP;
  END;

  PROCEDURE print_employee(p_rec_emp employees%rowtype) IS
  BEGIN
    DBMS_OUTPUT.PUT_LINE(p_rec_emp.department_id ||' '||
```

©2016 Capgemini. All rights reserved.

The information contained in this document is proprietary and confidential. For Capgemini only.

```
                    p_rec_emp.employee_id||' '||
                    p_rec_emp.first_name||' '||
                    p_rec_emp.last_name||' '||
                    p_rec_emp.job_id||' '||
                    p_rec_emp.salary);
  END;

  PROCEDURE show_employees IS
  BEGIN
    IF emp_table IS NOT NULL THEN
      DBMS_OUTPUT.PUT_LINE('Employees in Package table');
      FOR i IN 1 .. emp_table.COUNT
      LOOP
        print_employee(emp_table(i));
      END LOOP;
    END IF;
  END show_employees;

  FUNCTION valid_deptid(p_deptid IN
departments.department_id%TYPE)
   RETURN BOOLEAN IS
    v_dummy PLS_INTEGER;
  BEGIN
    RETURN valid_departments.exists(p_deptid);
  EXCEPTION
    WHEN NO_DATA_FOUND THEN
    RETURN FALSE;
END valid_deptid;

BEGIN
  init_departments;
END emp_pkg;

/
SHOW ERRORS
```

©2016 Capgemini. All rights reserved.

The information contained in this document is proprietary and confidential. For Capgemini only.

Lab 9. Types of PLSQL pragma

Goals	• Understand Different type of PLSQL pragma.
Time	1 Hours.

1.4. The purpose of this example is to show the usage of predefined exceptions. Write a PL/SQL block to select the name of the employee with a given salary value.

a. Delete all records in the messages table. Use the DEFINE command to define a variable sal and initialize it to 6000.

b. In the declarative section declare two variables: ename of type employees.last_name and emp_sal of type employees.salary. Pass the value of the substitution variables to emp_sal.

c. In the executable section, retrieve the last names of employees whose salaries are equal to the value in emp_sal.
Note: Do not use explicit cursors. If the salary entered returns only one row, insert into the messages table the employee's name and the salary amount.

d. If the salary entered does not return any rows, handle the exception with an appropriate exception handler and insert into the messages table the message "No employee with a salary of <salary>."

e. If the salary entered returns more than one row, handle the exception with an appropriate exception handler and insert into the messages table the message "More than one employee with a salary of <salary>."

f. Handle any other exception with an appropriate exception handler and insert into the messages table the message "Some other error occurred."

g. Display the rows from the messages table to check whether the PL/SQL block has executed successfully. Sample output is shown below.

©2016 Capgemini. All rights reserved.

The information contained in this document is proprietary and confidential. For Capgemini only.

Solution :

```
SET VERIFY OFF
DECLARE
  v_ename     employees.last_name%TYPE;
  v_emp_sal   employees.salary%TYPE := 6000;
BEGIN
  SELECT      last_name
  INTO                v_ename
  FROM                employees
  WHERE       salary = v_emp_sal;
  INSERT INTO messages (results)
  VALUES (v_ename || ' - ' || v_emp_sal);

  EXCEPTION
  WHEN no_data_found THEN
  INSERT INTO messages (results)
  VALUES ('No employee with a salary of '|| TO_CHAR(v_emp_sal));
  WHEN too_many_rows THEN
  INSERT INTO messages (results)
  VALUES ('More than one employee with a salary of '||
TO_CHAR(v_emp_sal));
```

3.2. The purpose of this example is to show how to declare exceptions with a standard Oracle server error. Use the Oracle server error ORA-02292 (integrity constraint violated – child record found).

a. In the declarative section, declare an exception childrecord_exists. Associate the declared exception with the standard Oracle server error –02292.
b. In the executable section, display 'Deleting department 40.....'. Include a DELETE statement to delete the department with department_id 40
c. Include an exception section to handle the childrecord_exists exception and display the appropriate message. Sample output is shown below.

```
Deleting department 40........
Cannot delete this department. There are employees in this department
(child records exist.)
PL/SQL procedure successfully completed.
```

3.Load the script lab_07_04_soln.sql.

a. Observe the declarative section of the outer block. Note that the
no_such_employee exception is declared.

©2016 Capgemini. All rights reserved.

The information contained in this document is proprietary and confidential. For Capgemini only.

b. Look for the comment “RAISE EXCEPTION HERE.” If the value of emp_id is
not between 100 and 206, then raise the no_such_employee exception.

c. Look for the comment “INCLUDE EXCEPTION SECTION FOR OUTER
BLOCK” and handle the exceptions no_such_employee and too_many_rows.
Display appropriate messages when the exceptions occur. The employees table has only one employee working in the HR department and therefore the code is written accordingly. The too_many_rows exception is handled to indicate that the select statement retrieves more than one employee working in the HR department.

d.Close the outer block.

e.Save your script as lab_08_03_soln.sql.

f.Execute the script. Enter the employee number and the department number and observe the output. Enter different values and check for different conditions.
The sample output for employee ID 203 and department ID 100 is shown below.

```
SET SERVEROUTPUT ON
DECLARE
  e_childrecord_exists EXCEPTION;
  PRAGMA EXCEPTION_INIT(e_childrecord_exists, -02292);
BEGIN
   DBMS_OUTPUT.PUT_LINE(' Deleting department 40........');
  delete from departments where department_id=40;
EXCEPTION
  WHEN e_childrecord_exists THEN
  DBMS_OUTPUT.PUT_LINE(' Cannot delete this department. There
are employees in this department (child records exist.) ');
END;
```

©2016 Capgemini. All rights reserved.

The information contained in this document is proprietary and confidential. For Capgemini only.

```
NUMBER OF RECORDS MODIFIED : 6
The following employees' salaries are updated
Nancy Greenberg
Daniel Faviet
John Chen
Ismael Sciarra
Jose Manuel Urman
Luis Popp
PL/SQL procedure successfully completed.
```

©2016 Capgemini. All rights reserved.

The information contained in this document is proprietary and confidential. For Capgemini only.

Lab 10. Collection elements

Goals	• Understand Different collection elements.
Time	1 Hour

a. Example of nested tables

```
DECLARE
  TYPE names_table IS TABLE OF VARCHAR2(10);
  TYPE grades IS TABLE OF INTEGER;

  names names_table;
  marks grades;
  total integer;
BEGIN
  names := names_table('Kavita', 'Pritam', 'Ayan', 'Rishav',
'Aziz');
  marks:= grades(98, 97, 78, 87, 92);
  total := names.count;
  dbms_output.put_line('Total '|| total || ' Students');
  FOR i IN 1 .. total LOOP
    dbms_output.put_line('Student:'||names(i)||', Marks:' ||
marks(i));
  end loop;
END;
/
```

©2016 Capgemini. All rights reserved.

The information contained in this document is proprietary and confidential. For Capgemini only.

Lab 11. Oracle 11g features

Goals	• Understand oracle 11g features.
Time	2 Hours.

1.5. Create the ADD_EMPLOYEE procedure to add an employee to the EMPLOYEES table. The row should be added to the EMPLOYEES table if the VALID_DEPTID function returns TRUE; otherwise, alert the user with an appropriate message. Provide the following parameters (with defaults specified in parentheses): first_name, last_name, email, job (SA_REP), mgr (145), sal (1000), comm (0), and deptid (30). Use the EMPLOYEES_SEQ sequence to set the employee_id column, and set hire_date to TRUNC(SYSDATE).

Solution :

```
CREATE OR REPLACE PROCEDURE add_employee(
   p_first_name employees.first_name%TYPE,
   p_last_name  employees.last_name%TYPE,
   p_email      employees.email%TYPE,
   p_job        employees.job_id%TYPE        DEFAULT 'SA_REP',
   p_mgr        employees.manager_id%TYPE    DEFAULT 145,
   p_sal        employees.salary%TYPE        DEFAULT 1000,
   p_comm       employees.commission_pct%TYPE DEFAULT 0,
   p_deptid     employees.department_id%TYPE  DEFAULT 30)
IS
BEGIN
 IF valid_deptid(p_deptid) THEN
   INSERT INTO employees(employee_id, first_name,
last_name, email,
     job_id, manager_id, hire_date, salary, commission_pct,
department_id)
   VALUES (employees_seq.NEXTVAL, p_first_name,
p_last_name, p_email,
     p_job, p_mgr, TRUNC(SYSDATE), p_sal, p_comm,
p_deptid);
 ELSE
   RAISE_APPLICATION_ERROR (-20204, 'Invalid department
ID. Try again.');
 END IF;
END add_employee;
```

©2016 Capgemini. All rights reserved.

The information contained in this document is proprietary and confidential. For Capgemini only.

Lab 12. Best practices of PL SQL code, Dynamic SQL

Goals	• Understand best practices of PL SQL code and dynamic sql.
Time	1 Hour.

6.1. Create a package called TABLE_PKG that uses Native Dynamic SQL to create or drop a table, and to populate, modify, and delete rows from the table.

a. Create a package specification with the following procedures:
PROCEDURE make(table_name VARCHAR2, col_specs VARCHAR2)
PROCEDURE add_row(table_name VARCHAR2, col_values VARCHAR2,
cols VARCHAR2 := NULL)
PROCEDURE upd_row(table_name VARCHAR2, set_values VARCHAR2, conditions VARCHAR2 := NULL)
PROCEDURE del_row(table_name VARCHAR2,conditions VARCHAR2 := NULL);
PROCEDURE remove(table_name VARCHAR2)
Ensure that subprograms manage optional default parameters with NULL values.

Solution :

```
CREATE OR REPLACE PACKAGE table_pkg IS
  PROCEDURE make(p_table_name VARCHAR2, p_col_specs
VARCHAR2);
  PROCEDURE add_row(p_table_name VARCHAR2,
p_col_values VARCHAR2,
    p_cols VARCHAR2 := NULL);
  PROCEDURE upd_row(p_table_name VARCHAR2,
p_set_values VARCHAR2,
    p_conditions VARCHAR2 := NULL);
  PROCEDURE del_row(p_table_name VARCHAR2,
p_conditions VARCHAR2 := NULL);
  PROCEDURE remove(p_table_name VARCHAR2);
END table_pkg;
/
SHOW ERRORS
```

b. Create the package body that accepts the parameters and dynamically constructs the appropriate SQL statements that are executed using Native Dynamic SQL, except for the remove

©2016 Capgemini. All rights reserved.

The information contained in this document is proprietary and confidential. For Capgemini only.

procedure that should be written using the DBMS_SQL package.

Solution :

```
CREATE OR REPLACE PACKAGE BODY table_pkg IS
  PROCEDURE execute(p_stmt VARCHAR2) IS
  BEGIN
    DBMS_OUTPUT.PUT_LINE(p_stmt);
    EXECUTE IMMEDIATE p_stmt;
  END;

  PROCEDURE make(p_table_name VARCHAR2, p_col_specs
VARCHAR2) IS
    v_stmt VARCHAR2(200) := 'CREATE TABLE '||
p_table_name ||
                        ' (' || p_col_specs || ')';
  BEGIN
    execute(v_stmt);
  END;
  PROCEDURE add_row(p_table_name VARCHAR2,
p_col_values VARCHAR2,
    p_cols VARCHAR2 := NULL) IS
    v_stmt VARCHAR2(200) := 'INSERT INTO '|| p_table_name;
  BEGIN
    IF p_cols IS NOT NULL THEN
      v_stmt := v_stmt || ' (' || p_cols || ')';
    END IF;
    v_stmt := v_stmt || ' VALUES (' || p_col_values || ')';
    execute(v_stmt);
  END;

  PROCEDURE upd_row(p_table_name VARCHAR2,
p_set_values VARCHAR2,
    p_conditions VARCHAR2 := NULL) IS
    v_stmt VARCHAR2(200) := 'UPDATE '|| p_table_name || '
SET ' || p_set_values;
  BEGIN
    IF p_conditions IS NOT NULL THEN
      v_stmt := v_stmt || ' WHERE ' || p_conditions;
    END IF;
    execute(v_stmt);
  END;

  PROCEDURE del_row(p_table_name VARCHAR2,
p_conditions VARCHAR2 := NULL) IS
    v_stmt VARCHAR2(200) := 'DELETE FROM '||
p_table_name;
  BEGIN
```

©2016 Capgemini. All rights reserved.

The information contained in this document is proprietary and confidential. For Capgemini only.

```
    IF p_conditions IS NOT NULL THEN
      v_stmt := v_stmt || ' WHERE ' || p_conditions;
    END IF;
    execute(v_stmt);
  END;

  PROCEDURE remove(p_table_name VARCHAR2) IS
    cur_id INTEGER;
    v_stmt VARCHAR2(100) := 'DROP TABLE '||p_table_name;
  BEGIN
    cur_id := DBMS_SQL.OPEN_CURSOR;
    DBMS_OUTPUT.PUT_LINE(v_stmt);
    DBMS_SQL.PARSE(cur_id, v_stmt, DBMS_SQL.NATIVE);
    -- Parse executes DDL statements,no EXECUTE is required.
    DBMS_SQL.CLOSE_CURSOR(cur_id);
  END;

END table_pkg;
/
SHOW ERRORS
```

©2016 Capgemini. All rights reserved.

The information contained in this document is proprietary and confidential. For Capgemini only.

Appendices

Appendix A: Oracle Standards

Key points to keep in mind:

1. Write comments in your stored Procedures, Functions and SQL batches generously, whenever something is not very obvious. This helps other programmers to clearly understand your code. Do not worry about the length of the comments, as it will not impact the performance.
2. Prefix the table names with owner names, as this improves readability, and avoids any unnecessary confusion.

Some more Oracle standards:

To be shared by Faculty in class

©2016 Capgemini. All rights reserved.

The information contained in this document is proprietary and confidential. For Capgemini only.

Appendix B: Coding Best Practices

1. Perform all your referential integrity checks and data validations by using constraints (foreign key and check constraints). These constraints are faster than triggers. So use triggers only for auditing, custom tasks, and validations that cannot be performed by using these constraints.
2. Do not call functions repeatedly within your stored procedures, triggers, functions, and batches. For example: You might need the length of a string variable in many places of your procedure. However do not call the LENGTH function whenever it is needed. Instead call the LENGTH function once, and store the result in a variable, for later use.

©2016 Capgemini. All rights reserved.

The information contained in this document is proprietary and confidential. For Capgemini only.