

Oracle (PL/SQL)

Lesson 3: Introduction to PL/SQL

Lesson Objectives

- To understand the following topics:
 - Features of PL/SQL
 - PL/SQL Block structure
 - Handling variables in PL/SQL
 - Declaring a PL/SQL table
 - Variable scope and Visibility
 - SQL in PL/SQL
 - Programmatic Constructs



3.1: Introduction to 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).



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).

3.1: Introduction to PL/SQL

Features of PL/SQL

- PL/SQL provides the following features:
 - Tight Integration with SQL
 - Better performance
 - Several SQL statements can be bundled together into one PL/SQL block and sent to the server as a single unit.
 - Standard and portable language
 - 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.



Copyright © Capgemini 2015. All Rights Reserved 4

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.

3.1: Introduction to PL/SQL

PL/SQL Block Structure

- A PL/SQL block comprises of the following structure

- DECLARE – Optional
 - Variables, cursors, user-defined exceptions
- BEGIN – Mandatory
 - SQL statements
 - PL/SQL statements
- EXCEPTION – Optional
 - Actions to perform when errors occur
- END; – Mandatory

```
DECLARE
...
BEGIN
...
EXCEPTION
...
END;
```



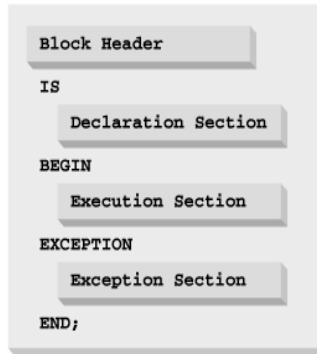
Copyright © Capgemini 2015. All Rights Reserved 5

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.

PL/SQL block structure (contd.):



- **Header**

It is relevant for named blocks only. The header determines the way the named block or program must be called.

- **Declaration section**

The part of the block that declares variables, cursors, and sub-blocks that are referenced in the Execution and Exception sections.

- **Execution section**

It is the part of the PL/SQL blocks containing the executable statements; the code that is executed by the PL/SQL runtime engine.

- **Exception section**

It is the section that handles exceptions for normal processing (warnings and error conditions).

3.2: PL/SQL Block Structure

Block Types

- There are three types of blocks in PL/SQL:
 - Anonymous
 - Named:
 - Procedure
 - Function

Anonymous

```
[DECLARE]

BEGIN
  --statements

[EXCEPTION]

END;
```

Procedure

```
PROCEDURE name
IS
  BEGIN
    --statements


  [EXCEPTION]

END;
```

Function

```
FUNCTION name
RETURN datatype
IS
  BEGIN
    --statements
    RETURN value;
  [EXCEPTION]

END;
```

 Copyright © Capgemini 2015. All Rights Reserved 7

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”.

Representation of a PL/SQL block:

```

DECLARE                                -- Declaration Section
    V_Salary  NUMBER(7,2);
/* V_Salary is a variable declared in a PL/SQL block. This variable is
used to store JONES' salary. */
Low_Sal EXCEPTION;                    -- an exception
BEGIN                                -- Execution Section
SELECT sal INTO V_Salary

FROM emp WHERE ename = 'JONES'
        FOR UPDATE of sal ;
        IF V_Salary < 3000 THEN
            RAISE Low_Sal ;
        END IF;
EXCEPTION                            -- Exception Section
    WHEN Low_Sal THEN
        UPDATE emp SET sal = sal + 500 WHERE ename = 'JONES' ;
END ;                                -- End of Block
/                                       -- PL/SQL block terminator

Output
SQL> /

PL/SQL procedure successfully completed.

```

- The notations used in a PL/SQL block are given below:
 1. -- is a single line comment.
 2. /* */ is a multi-line comment.
 3. Every statement must be terminated by a semicolon (;).
 4. PL/SQL block is terminated by a slash (/) on a line by itself.
- A PL/SQL block must have a “Declaration section” and an “Execution section”. It can optionally have an Exception section, as well.

3.3: Handling Variables in PL/SQL

Points to Remember

- While handling variables in PL/SQL:
 - declare and initialize variables within the declaration section
 - assign new values to variables within the executable section
 - pass values into PL/SQL blocks through parameters
 - view results through output variables

3.3: Handling Variables in PL/SQL

Guidelines

- Given below are a few guidelines for declaring variables:
 - follow the naming conventions
 - initialize the variables designated as NOT NULL
 - initialize the identifiers by using the assignment operator (:=) or by using the DEFAULT reserved word
 - declare at most one Identifier per line

3.3: Handling Variables in PL/SQL

Types of Variables

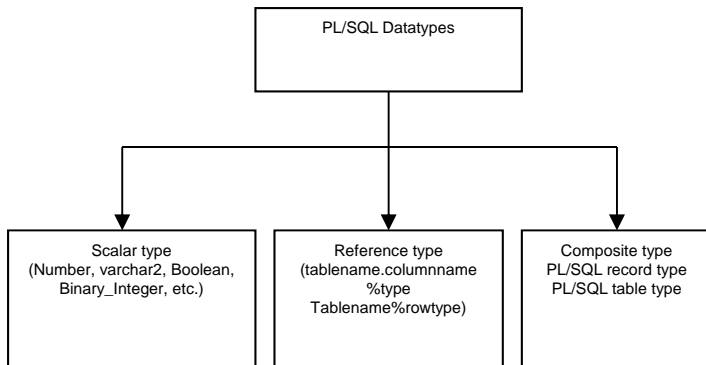
- PL/SQL variables
 - Scalar
 - Composite
 - Reference
 - LOB (large objects)
- Non-PL/SQL variables
 - Bind and host variables



Copyright © Capgemini 2015. All Rights Reserved 11

Types of Variables: PL/SQL Datatype:

- All PL/SQL datatypes are classified as scalar, reference and Composite type.
- Scalar datatypes do not have any components within it, while composite datatypes have other datatypes within them.
- A reference datatype is a pointer to another datatype.



3.3: Handling Variables in PL/SQL

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;
```



Copyright © Capgemini 2015. All Rights Reserved 12

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.

Declaring PL/SQL Variables (contd.):**For example:**

```
DECLARE
  v_description  varchar2 (25);
  v_sal          number (5) not null:
= 3000;
  v_compcode     varchar2 (20)
constant: = 'abc
              consultants';
  v_comm         not null default  0;
```

3.3: Handling Variables in PL/SQL

Base Scalar Data Types

- Base Scalar Datatypes:
 - Given below is a list of Base Scalar Datatypes:
 - VARCHAR2 (maximum_length)
 - NUMBER [(precision, scale)]
 - DATE
 - CHAR [(maximum_length)]
 - LONG
 - LONG RAW
 - BOOLEAN
 - BINARY_INTEGER
 - PLS_INTEGER



Copyright © Capgemini 2015. All Rights Reserved 14

Base Scalar Datatypes:

1. NUMBER

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

2. 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;

3. 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.

4. 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.

3.3: Handling Variables in PL/SQL

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;
```



Copyright © Capgemini 2015. All Rights Reserved 15

Base Scalar Datatypes (contd.):

5. LONG

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

6. 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.

7. 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;
/
```

3.3: Handling Variables in PL/SQL

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



Copyright © Capgemini 2015. All Rights Reserved 16

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:

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

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

3.3: Handling Variables in PL/SQL

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;  
...
```



Copyright © Capgemini 2015. All Rights Reserved 17

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;
```

3.3: Handling Variables in PL/SQL

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;
```



Copyright © Capgemini 2015. All Rights Reserved 18

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;
/
```

3.3: Handling Variables in PL/SQL

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;
```

3.3: Handling Variables in PL/SQL

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;
```

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

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



Copyright © Capgemini 2015. All Rights Reserved 20

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);
```

3.3: Handling Variables in PL/SQL

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);
```

3.3: Handling Variables in PL/SQL

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.

3.3: Handling Variables in PL/SQL

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.



Copyright © Capgemini 2015. All Rights Reserved 23

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.

3.3: Handling Variables in PL/SQL

Record Data Types (Contd...)

■ Syntax:

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



Copyright © Capgemini 2015. All Rights Reserved 24

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.

3.3: Handling Variables in PL/SQL

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),
```



Copyright © Capgemini 2015. All Rights Reserved 25

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);
```

3.3: Handling Variables in PL/SQL

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;
```

3.3: Handling Variables in PL/SQL

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



Copyright © Capgemini 2015. All Rights Reserved 27

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/SQL 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.

1.4: Declaring a PL/SQL table

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.



Copyright © Capgemini 2015. All Rights Reserved 28

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.

1.4: Declaring a PL/SQL table

Table Data Type - Examples

Example 1:

```
DECLARE
  ▪ To create a PL/SQL table named as "student_table" of char column.
  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;
```



Copyright © Capgemini 2015. All Rights Reserved 29

Declaring a PL/SQL table (contd.):

• Example 3:

➤ To declare a NOT NULL constraint

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

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

1.4: Declaring a PL/SQL table

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
```



Copyright © Capgemini 2015. All Rights Reserved 30

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.

1.4: Declaring a PL/SQL table

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;
```



Copyright © Capgemini 2015. All Rights Reserved 31

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:

PL/SQL table_name (primary key value)

- 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.

1.4: Declaring a PL/SQL table

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.



Copyright © Capgemini 2015. All Rights Reserved 32

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';
```


3.3: Handling Variables in PL/SQL

Scope and Visibility of Variables

■ Scope of Variables:

- The scope of a variable is the portion of a program in which the variable can be accessed.
- The scope of the variable is from the “variable declaration” in the block till the “end” of the block.
- When the variable goes out of scope, the PL/SQL engine will free the memory used to store the variable, as it can no longer be referenced.



Copyright © Capgemini 2015. All Rights Reserved 33

Scope and Visibility of Variables:

- References to an identifier are resolved according to its scope and visibility.
 - The scope of an identifier is that region of a program unit (block, subprogram, or package) from which you can reference the identifier.
 - An identifier is visible only in the regions from which you can reference the identifier using an unqualified name.
- Identifiers declared in a PL/SQL block are considered “local” to that “block” and “global” to all its “sub-blocks”.
 - If a global identifier is re-declared in a sub-block, both identifiers remain in scope. However, the local identifier is visible within the sub-block only because you must use a qualified name to reference the global identifier.
- Although you cannot declare an identifier twice in the same block, you can declare the same identifier in two different blocks.
 - The two items represented by the identifier are “distinct”, and any change in one does not affect the other. However, a block cannot reference identifiers declared in other blocks at the same level because those identifiers are neither local nor global to the block.

3.3: Handling Variables in PL/SQL

Scope and Visibility of Variables (Contd...)

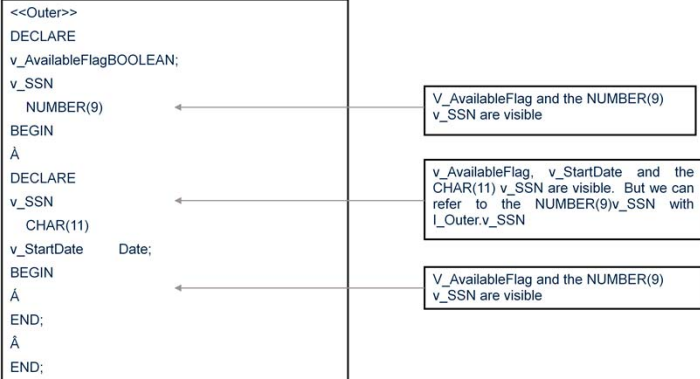
- Visibility of Variables:

- The visibility of a variable is the portion of the program, where the variable can be accessed without having to qualify the reference. The visibility is always within the scope, it is not visible.

3.3: Handling Variables in PL/SQL

Scope and Visibility of Variables (Contd...)

- Pictorial representation of visibility of a variable:



3.3: Handling Variables in PL/SQL

Scope and Visibility of Variables (Contd...)

```
<<OUTER>>
DECLARE
  V_Flag BOOLEAN ;
  V_Var1 CHAR(9);
BEGIN
  <<INNER>>
  DECLARE
    V_Var1 NUMBER(9);
    V_Date DATE;
  BEGIN
    NULL;
  END;
  NULL;
END;
```

3.4: SQL in PL/SQL

Types of Statements

- Given below are some of the SQL statements that are used in PL/SQL:
 - INSERT statement
 - The syntax for the INSERT statement remains the same as in SQL-INSERT.
 - For example:

```
DECLARE
    v_dname varchar2(15) := 'Accounts';
BEGIN
    INSERT into department_master
    VALUES (50, v_dname);
END;
```

3.4: SQL in PL/SQL

Types of Statements (Contd...)

- DELETE statement

- For example:

```
DECLARE
    v_sal_cutoff number := 2000;
BEGIN
    DELETE FROM staff_master
    WHERE staff_sal < v_sal_cutoff;
END;
```

3.4: SQL in PL/SQL

Types of Statements (Contd...)

- UPDATE statement
 - For example:

```
DECLARE
    v_sal_incr number(5) := 1000;
BEGIN
    UPDATE staff_master
    SET staff_sal = staff_sal + v_sal_incr
    WHERE staff_name='Smith';
END;
```

3.4: SQL in PL/SQL

Types of Statements (Contd...)

- SELECT statement
 - Syntax:

```
SELECT Column_List INTO Variable_List
FROM Table_List
[WHERE expr1]
[CONNECT BY expr2 [START WITH expr3]]
[GROUP BY expr4] [HAVING expr5]
[UNION | INTERSECT | MINUS SELECT ...]
[ORDER BY expr | ASC | DESC]
[FOR UPDATE [OF Col1,...] [NOWAIT]]
INTO Variable_List;
```


3.4: SQL in PL/SQL

Types of Statements (Contd...)

- The column values returned by the SELECT command must be stored in variables.
- The Variable_List should match Column_List in both COUNT and DATATYPE.
- Here the variable lists are PL/SQL (Host) variables. They should be defined before use.



Copyright © Capgemini 2015. All Rights Reserved 41

SELECT Statement:

Note:

- The SELECT clause is used if the selected row must be modified through a DELETE or UPDATE command.
- Since the contents of the row must not be modified between the SELECT command and the UPDATE command, it is necessary to lock the row after the SELECT command.
 - FOR UPDATE will lock the selected row.
 - OF Col1,... lists the columns which can be modified by the UPDATE command.
 - If OF Col1,... is missing, all the columns can be modified.
- It is possible that when SELECT..UPDATE is issued, the concerned row is already locked by some other user or a previous SELECT..UPDATE command.
 - If NOWAIT is not given, then the current SELECT..UPDATE command will wait until the lock is cleared.
 - IF NOWAIT is given, then the SELECT..UPDATE will return immediately with a failure.

3.4: SQL in PL/SQL

Types of Statements (Contd...)

- Example: <<BLOCK1>>

```
DECLARE
    deptno  number(10) := 30;
    dname   varchar2(15);
BEGIN
    SELECT dept_name INTO dname FROM
    department_master WHERE dept_code = Block1. deptno;
    DELETE FROM department_master
           WHERE dept_code = Block1. deptno ;
END;
```



Copyright © Capgemini 2015. All Rights Reserved 42

SELECT statement (contd.):

SELECT statement (contd.):**More examples**

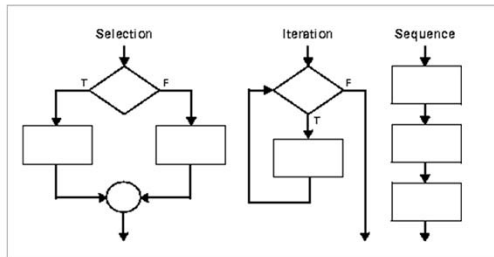
```
DECLARE
dept_code          number(10) := 30;
v_dname            varchar2(15);
BEGIN
SELECT dept_name INTO v_dname FROM
department_master WHERE
dept_code=dept_code
DELETE FROM department_master
WHERE dept_code = dept_code ;
END;
```

- Here the SELECT statement will select names of all the departments and not only deptno. 30.
- The DELETE statement will delete all the employees.
 - This happens because when the PL/SQL engine sees a condition `expr1 = expr2`, the `expr1` and `expr2` are first checked to see whether they match the database columns first and then the PL/SQL variables.
 - So in the above example, where you see `deptno = deptno`, both are treated as database columns, and the condition will become TRUE for every row of the table.
- If a block has a label, then variables with same names as database columns can be used by using `<blockname>.` `Variable_Name` notation.
- It is not a good programming practice to use same names for PL/SQL variables and database columns.

3.5: Programmatic Constructs in PL/SQL

Types of Programmatic Constructs

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



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.

3.5: Programmatic Constructs in PL/SQL

IF Construct

- 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;
```



Copyright © Capgemini 2015. All Rights Reserved 45

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.

3.5: Programmatic Constructs in PL/SQL

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;
```



Copyright © Capgemini 2015. All Rights Reserved 45

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.

3.5: Programmatic Constructs in PL/SQL

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;
```



Copyright © Capgemini 2015. All Rights Reserved 47

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.

3.5: Programmatic Constructs in PL/SQL

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.



Copyright © Capgemini 2015. All Rights Reserved 48

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;
```


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:

```
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;
```

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

3.5: Programmatic Constructs in PL/SQL

Simple Loop

- Looping

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

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

3.5: Programmatic Constructs in PL/SQL

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 ;
/
```



Copyright © Capgemini 2015. All Rights Reserved 51

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.

3.5: Programmatic Constructs in PL/SQL

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.

3.5: Programmatic Constructs in PL/SQL

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;

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



Copyright © Capgemini 2015. All Rights Reserved 53

Note:

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

3.5: Programmatic Constructs in PL/SQL

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.



Copyright © Capgemini 2015. All Rights Reserved 54

Note:

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

3.5: Programmatic Constructs in PL/SQL

For Loop

- FOR Loop:

- Syntax:

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



Copyright © Capgemini 2015. All Rights Reserved 55

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:
 - $\text{Upper_Bound} - \text{Lower_Bound} + 1$.
- 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.

3.5: Programmatic Constructs in PL/SQL

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 ;
```



Copyright © Capgemini 2015. All Rights Reserved 56

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.

3.5: Programmatic Constructs in PL/SQL

While Loop

- **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.



Copyright © Capgemini 2015. All Rights Reserved 57

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;
/
```

3.5: Programmatic Constructs in PL/SQL

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 ;
```

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:
 - PL/SQL is a procedural extension to SQL.
 - PL/SQL exhibits a block structure, different block types being: Anonymous, Procedure, and Function.
 - While declaring variables in PL/SQL:
 - declare and initialize variables within the declaration section
 - assign new values to variables within the executable section
 - pass values into PL/SQL blocks through parameters
 - view results through output variables



Summary

- Different types of PL/SQL Variables are: Scalar, Composite, Reference, LOB
- Scope of a variable: It is the portion of a program in which the variable can be accessed.
- Visibility of a variable: It is the portion of the program, where the variable can be accessed without having to qualify the reference.
- Different programmatic constructs in PL/SQL are Selection structure, Iteration structure, Sequence structure



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



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



Review Question

- Question 5: While using FOR loop, Upper_Bound, and Lower_Bound must be integers.
 - True / False

