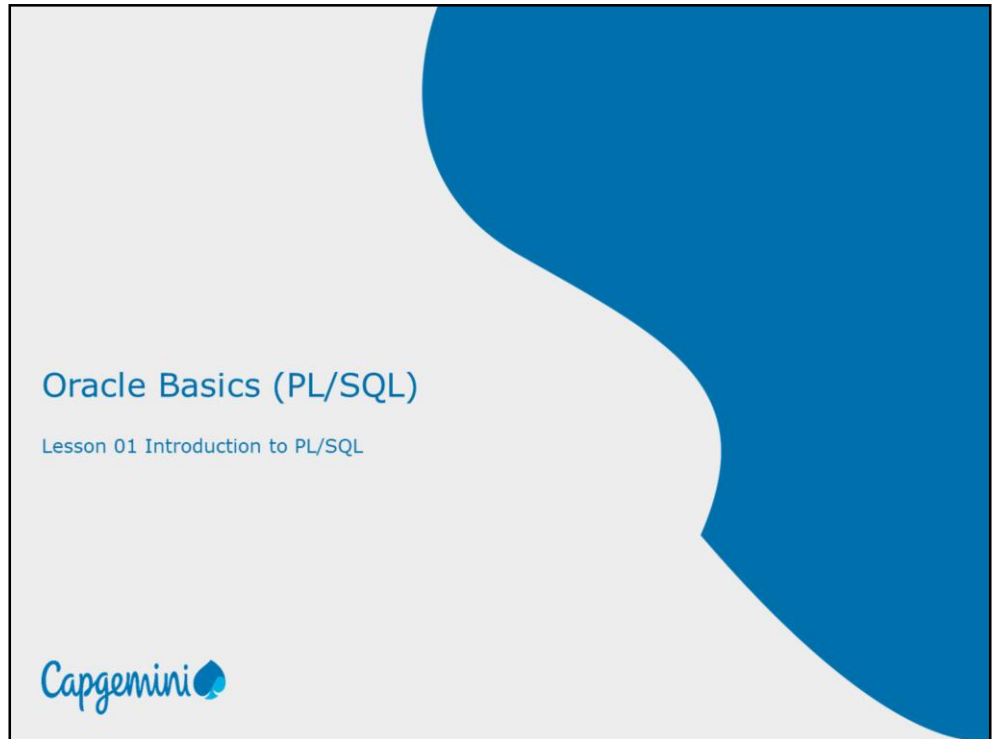


Instructor Notes:



Instructor Notes:

None

Lesson Objectives

To understand the following topics:

- Introduction to PL/SQL
- PL/SQL Block structure
- Handling variables in PL/SQL
- Variable scope and Visibility
- SQL in PL/SQL
- Programmatic Constructs

Instructor Notes:

None

1.1: 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).

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).

Instructor Notes:

None

1.1: Introduction to PL/SQL



Salient Features

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.

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.

Instructor Notes:

None

1.1: Introduction to PL/SQL

PL/SQL Block Structure

A PL/SQL block comprises of the following structures:

- 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;
```

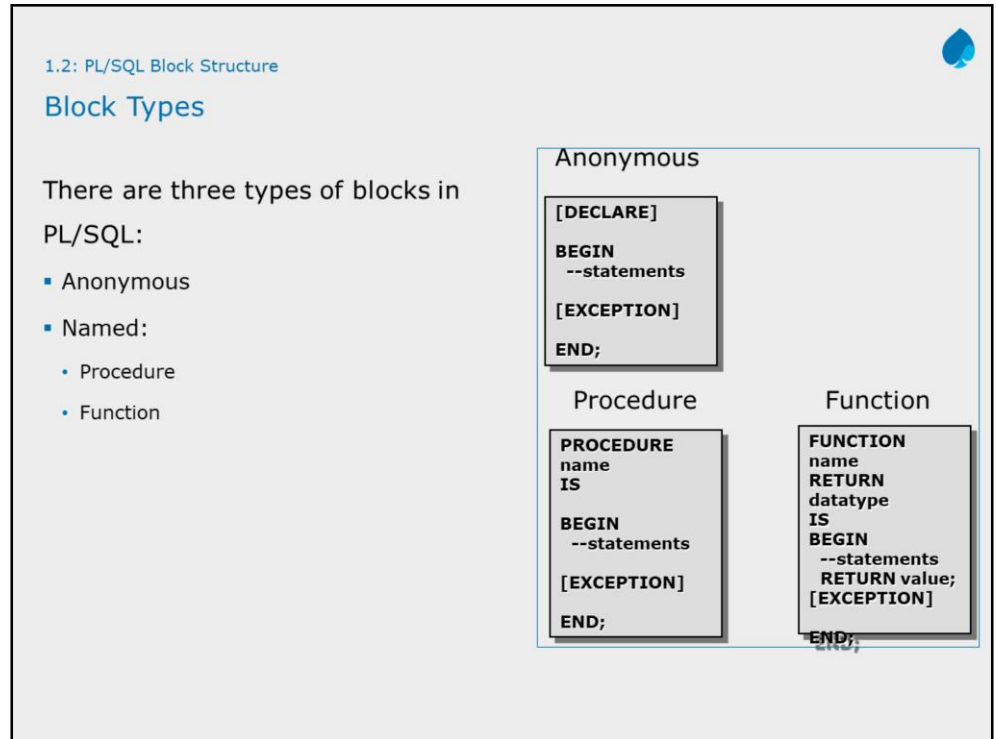
PL/SQL Block Structure:

- 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.

Instructor Notes:

None

**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”.

Instructor Notes:

None

1.3: 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

Instructor Notes:

None

1.3: Handling Variables in PL/SQL

**Guidelines for declaring variables**

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

Instructor Notes:

LOB datatypes are covered in detail later.
Mention this in the class.

1.3: Handling Variables in PL/SQL

**Types of Variables****PL/SQL variables**

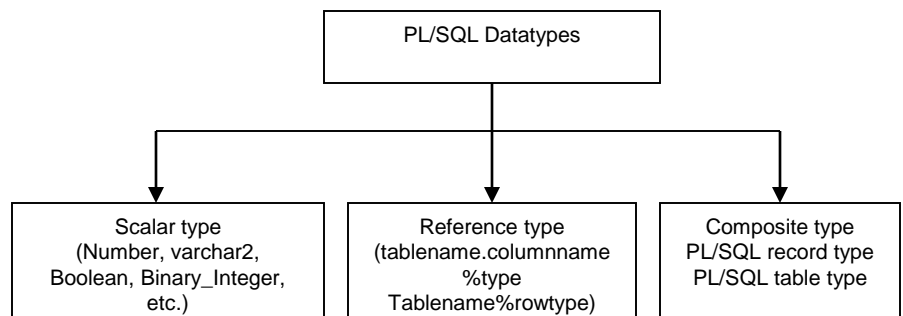
- Scalar
- Composite
- Reference
- LOB (large objects)

Non-PL/SQL variables

- Bind and host variables

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.



Instructor Notes:

None

1.3: 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;
```

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.

Instructor Notes:

None

1.3: 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)]
- BOOLEAN
- BINARY_INTEGER

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.

Instructor Notes:

None

1.3: 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;
```

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

```
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;  
/
```

Instructor Notes:

None

1.3: Handling Variables in PL/SQL

**Declaring Datatype by using %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

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.

Instructor Notes:

None

1.3: Handling Variables in PL/SQL



Declaring Datatype by using %TYPE Attribute

Example:

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

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;
```

Instructor Notes:

None

1.3: 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;
```

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:**
- V_Emprec emp%rowtype

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;

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

Instructor Notes:

None

1.3: Handling Variables in PL/SQL

**Composite Data Types**

Composite Datatypes in PL/SQL:

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

Instructor Notes:

None

1.3: 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.

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.

Instructor Notes:

None

1.3: Handling Variables in PL/SQL

**Record Data Types**

Syntax:

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

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.

Instructor Notes:

None

1.3: 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),
```

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);

Instructor Notes:

None

1.3: Handling Variables in PL/SQL

**Record Data Types - Example**

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;
```

Instructor Notes:

None

1.4: Scope and Visibility of variables

**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.

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.

Instructor Notes:

None

1.4: Scope and Visibility of variables

**Scope and Visibility of Variables****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.

Instructor Notes:

None

1.4: Scope and Visibility of variables

Scope and Visibility of Variables

Pictorial representation of visibility of a variable:

```
<<Outer>>
DECLARE
v_AvailableFlagBOOLEAN;
v_SSN
  NUMBER(9)
BEGIN
  À
  DECLARE
  v_SSN
    CHAR(11)
  v_StartDate  Date;
  BEGIN
  À
  END;
  À
  END;
```

V_AvailableFlag and the NUMBER(9) v_SSN are visible

v_AvailableFlag, v_StartDate and the CHAR(11) v_SSN are visible. But we can refer to the NUMBER(9)v_SSN with T_Outer.v_SSN

V_AvailableFlag and the NUMBER(9) v_SSN are visible

Instructor Notes:

None

1.4: Scope and Visibility of variables

**Scope and Visibility of Variables**

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


Instructor Notes:

None

1.5: 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;
```

Instructor Notes:

None

1.5: SQL in PL/SQL

**Types of Statements**

- DELETE statement

For Example:

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

Instructor Notes:

None

1.5: SQL in PL/SQL

**Types of Statements**▪ **SELECT statement**• **Syntax:**

```
SELECT Column_List INTO Variable_List  
FROM Table_List  
[WHERE expr1]  
GROUP BY expr4] [HAVING expr5]  
[ORDER BY expr | ASC | DESC]  
INTO Variable_List;
```

Instructor Notes:

None

1.5: SQL in PL/SQL

**Types of Statements**

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.

SELECT Statement:**Note:**

- The SELECT clause is used if the selected row must be modified through a DELETE or UPDATE command.

Instructor Notes:

None

1.5: SQL in PL/SQL

**Types of Statements**

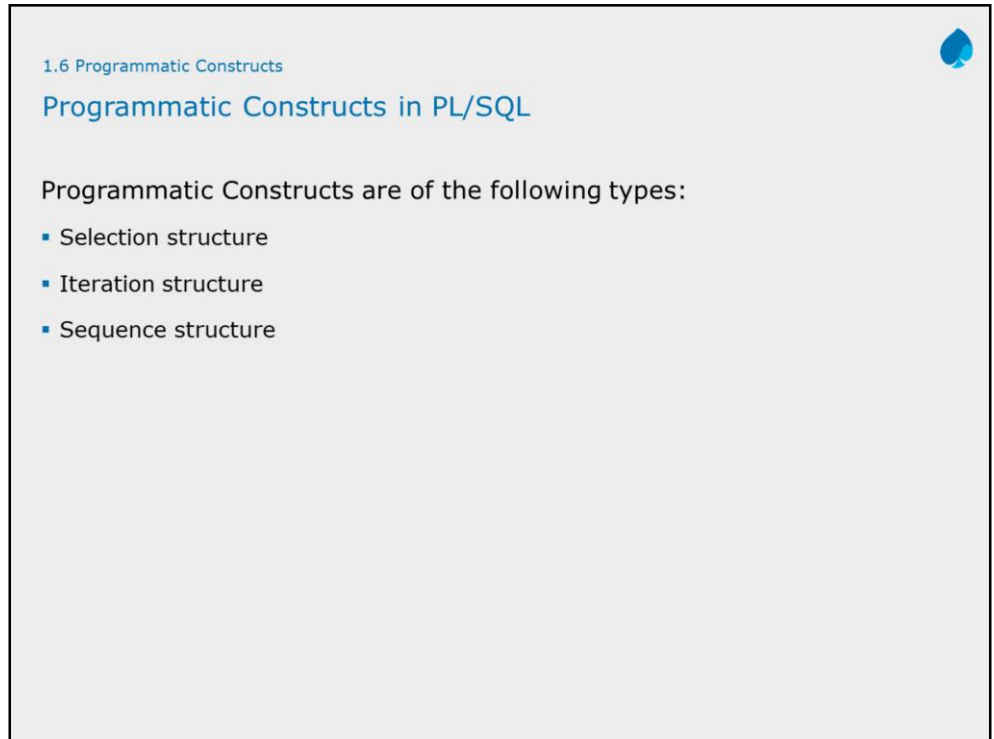
Example: <<BLOCK1>>

```
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;
```

SELECT statement (contd.):

Instructor Notes:

None



1.6 Programmatic Constructs

Programmatic Constructs in PL/SQL

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.

Instructor Notes:

None



1.6: 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;
```

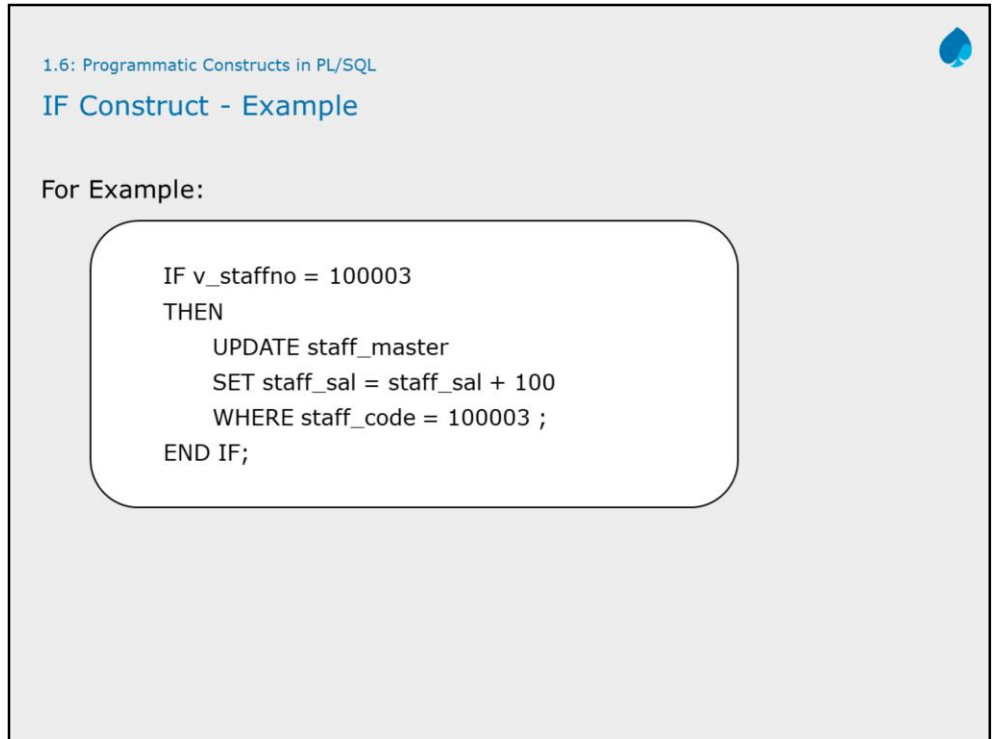
Programmatic Constructs (contd.)**Conditional Execution:**

- 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.

contd.

Instructor Notes:

None

A presentation slide with a light gray background. In the top right corner, there is a small blue Oracle logo. The slide title is "1.6: Programmatic Constructs in PL/SQL" in a small blue font, followed by "IF Construct - Example" in a larger blue font. Below the title, the text "For Example:" is displayed. A white rounded rectangle contains the following PL/SQL code:

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

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.

Instructor Notes:

None

1.6: Programmatic Constructs in PL/SQL

**IF Construct - Example**

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;
```

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.

Instructor Notes:

None

1.6: Programmatic Constructs in PL/SQL

**IF Construct - Example**

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;
```

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;
```

Instructor Notes:

None

1.6: Programmatic Constructs in PL/SQL

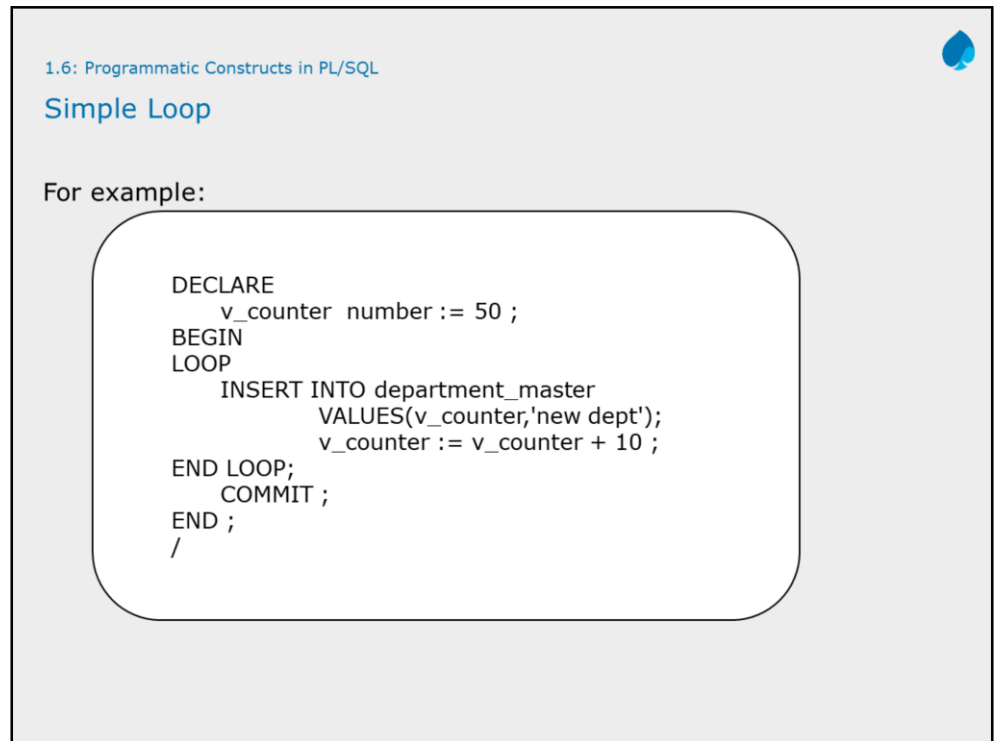
**Simple Loop****Looping**

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

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

Instructor Notes:

None



1.6: Programmatic Constructs in PL/SQL

Simple Loop

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 ;
/
```

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.

Instructor Notes:

None

1.6: 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.

contd.

Instructor Notes:

None

1.6: Programmatic Constructs in PL/SQL

**Simple Loop – EXIT statement****Syntax:**

```
BEGIN
.....
LOOP
    IF <Condition> THEN
        .....
        EXIT ;           -- Exits loop immediately
    END IF ;

    END LOOP;
LOOP
    .....
    .....
    EXIT WHEN <condition>
END LOOP;

.....
COMMIT ;
END ;
```

-- Control resumes here

Note:

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

Instructor Notes:

None

1.6: Programmatic Constructs in PL/SQL

**Simple Loop – EXIT statement**

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:

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

Instructor Notes:

None

1.6: Programmatic Constructs in PL/SQL



For Loop

FOR Loop:

- Syntax:

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

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.

Instructor Notes:

None

1.6: 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;
```

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;
/
```

Instructor Notes:

None

1.6: 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.

Instructor Notes:

None

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



Instructor Notes:

None

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



IGATE Sensitive

Instructor Notes:

None

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



Instructor Notes:**Answers for Review Questions:****Question 1:**

Answer: True

Question 2:

Answer: False

Question 3: True

Review & Question

Question 1: A record is a collection of individual fields that represents a row in the table.

- True/ False

Question 2: %ROWTYPE is used to declare a variable with the same datatype as a column of a specific table.

- True / False

Question 3: While using FOR loop, Upper_Bound, and Lower_Bound must be integers.

- True / False

