LAB session 11,13

Writing Executable statements in PL/SQL ,Explicit cursors in PL SQL

PL SQL Block Syntax & Guidelines

An ***identifier*** is a name for a PL/SQL object, including any of the following:

Constant,Variable,Exception,Procedure,Function,Package,Record,PL/SQL table

Cursor, Reserved word.

Identifiers can contain up to 30 characters, but they must start with an alphabet character.

Do no use the same name of identifier as of the column. Oracle will reference the column name instead of identifier.

Reserved words cannot be used as identifiers unless they are enclosed in double quotation marks (for example, “SELECT”)

A literal is an explicit numeric, character, string, or Boolean value not represented by an identifier.

Character literals include all the printable characters in the PL/SQL character set:letters, numerals, spaces, and special symbols.

Numeric literals can be represented either by a simple value (for example, -32.5) or by scientific notation (for example, 2E5, meaning 2 * 10 to the power of 5 = 200000).

PL SQL Block Syntax & Guidelines

Refer to the code below

```
int salary = 10000;
boolean myValue = true;
```

'10000' is a literal whose identifier is 'salary' .
'true' is a literal whose identifier is 'myValue' .

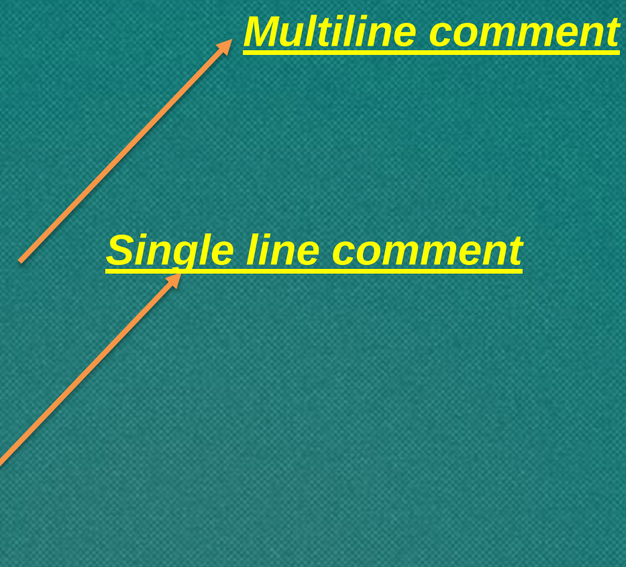
Commenting Code

Commenting Code

For single line comment we use **--**

For Multiple lines comments we will use **/* … */**

```
ACCEPT p_monthly_sal PROMPT ‘Please enter the
annual salary : ‘
v_sal NUMBER(9, 2);
BEGIN
/* Compute the annual salary based on the
monthly
salary input from the user */
v_sal := &p_monthly_sal * 12;
END; -- This is the end of the transaction
```



SQL Functions in PL/SQL

SQL Functions in PL/SQL

- Single-row number functions → Round,Trunc,MOD
- Single-row character functions → Upper,Lower,Instring,LPAD
- Datatype conversion functions → ToChar,ToDate,To_number
- Date functions
- GREATEST, LEAST → SELECT GREATEST (1, 2, 3) "Greatest" FROM DUAL ; O/P = 3

Following functions are not available in procedural statements

DECODE & Group functions like AVG, MIN, MAX, COUNT, SUM, STDDEV, and VARIANCE.

Group functions apply to groups of rows in a table and therefore they are available only in SQL statements in a PL/SQL block.

Data Type Conversions

Data type conversions

PL/SQL attempts to convert datatypes dynamically if they are mixed in a statement. For example, if a NUMBER value is assigned to a CHAR variable, then PL/SQL dynamically translates the number into a character representation, so that it can be stored in the CHAR variable.

we can also assign characters to DATE variables(with specified date format and vice versa.

Example:

```
V_DATE VARCHAR2(15);
V_DATE := TO_DATE(‘January 13, 1998’, ‘Month DD, YYYY’);
```

Nested Blocks

Nested Blocks

One of the advantages that PL/SQL has over SQL is the ability to nest statements. We can nest blocks wherever Required. Therefore, we can break down the executable part of a block into smaller blocks.
The exception section can also contain nested blocks.

Variable Scope

The scope of an object is the region or block where it has been declared . You can see scope of Y in following example.

Example

```
x BINARY_INTEGER;
BEGIN

…

    DECLARE
        Y NUMBER;

    BEGIN
    …
    END;

…
END;
```

scope of x

scope of y

Nested Block

Operators in PL/SQL

Operator in PL/SQL

- Logical
- Arithmetic
- Concatenation
- Parentheses to control order of operations
- Exponential operator (**)

The operations within an expression are done in a particular order depending on their precedence (priority). The following table shows the default order of operations from top to bottom:-

Operator	Operation
**, NOT	Exponentiation, logical navigation
+, -	Identity, negation
*, /	Multiplication, division
+, -, ||	Addition, subtraction, concatenation
=, !=, <, >, <=, >=, IS NULL, LIKE, BETWEEN, IN	Comparison
AND	Conjuction
OR	Inclusion

Using bind variables

Using Bind variables

To reference a bind variable colon(:) is used before its name let see its example

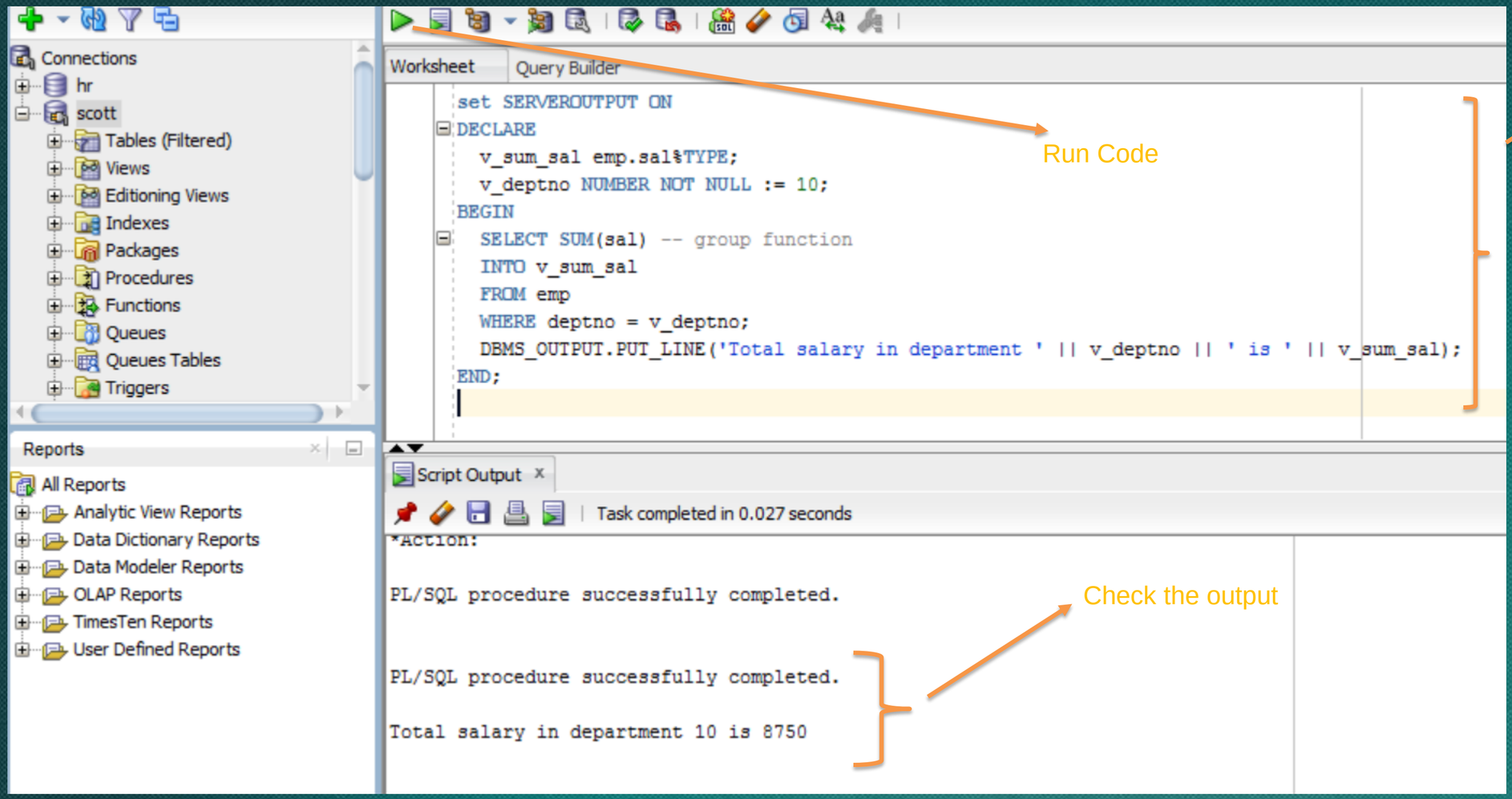
Example:

```
VARIABLE g_salary NUMBER
DECLARE
        v_sal emp.sal%TYPE
BEGIN
        SELECT sal
        INTO v_sal
        FROM emp
        WHERE empno = 7369;
        :g_salary := v_sal;
END;
/
```

We have seen practically in last lab that Binds variables can be print by using PRINT Command as :

```
SQL> PRINT g_salary
```

Using Bind variables



Interacting with oracle Database

Interacting with oracle database

This topic has already been covered as we studied the select statement and into clause in select statement in last lab.

We have seen an example of Retrieving data in PL/SQL via employee table as:

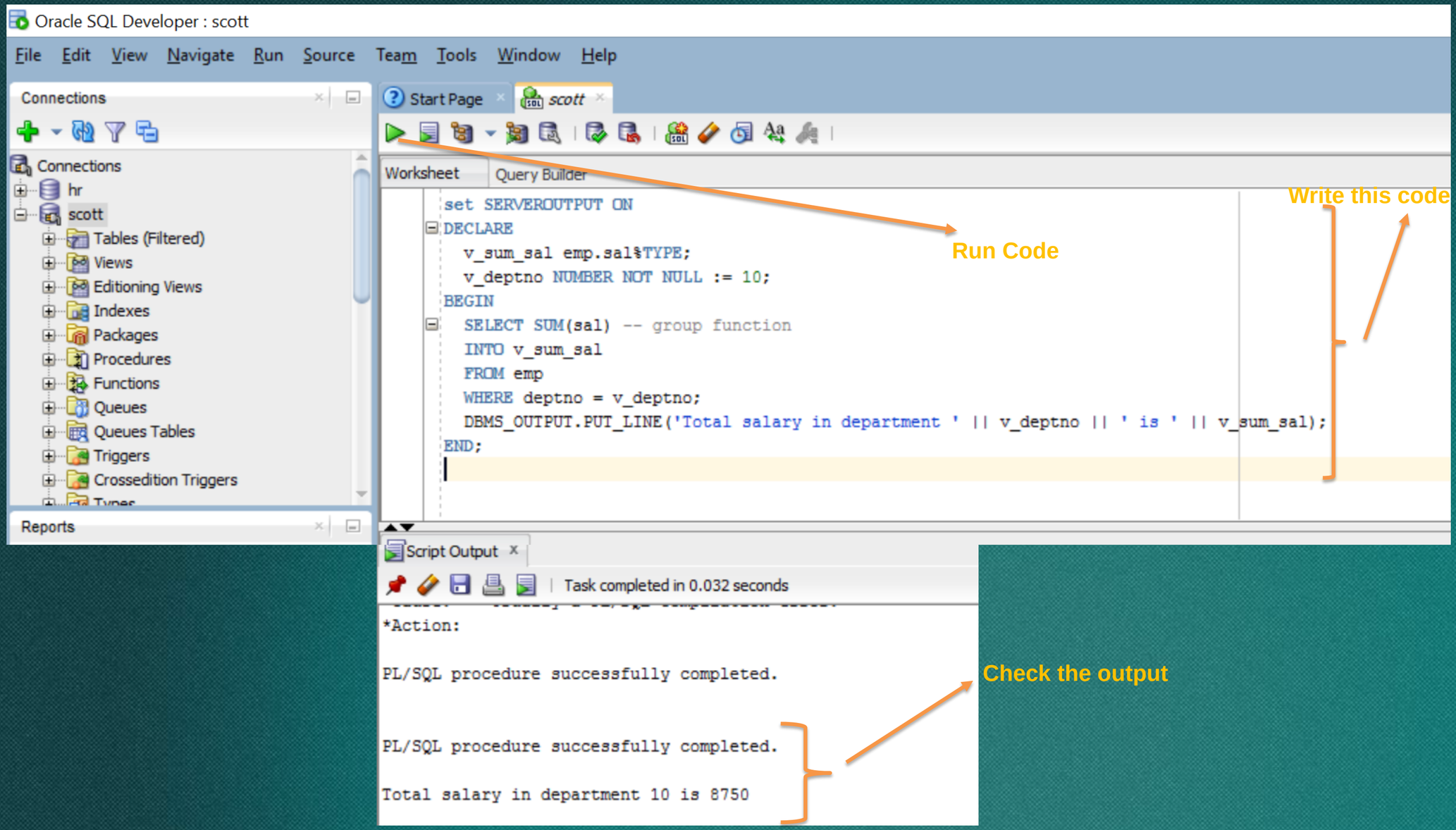
```
DECLARE
   v_EmpName  emp.ename%TYPE;
BEGIN
   SELECT ename INTO v_EmpName FROM emp WHERE ROWNUM = 1;
   DBMS_OUTPUT.PUT_LINE('Name = ' || v_EmpName);
END;
```

Let see how aggregate function can be applied in SQL section of PL/SQL with example

Example: Print the sum of all salaries in Accounting department (deptno = 10)

```
SET SERVEROUTPUT ON
DECLARE
        v_sum_sal emp.sal%TYPE;
        v_deptno NUMBER NOT NULL := 10;
BEGIN
        SELECT SUM(sal) -- group function
        INTO v_sum_sal
        FROM emp
        WHERE deptno = v_deptno;
        DBMS_OUTPUT.PUT_LINE('Total salary in department ' || v_deptno || ' is ' || v_sum_sal);
END;
```

Interacting with oracle database



SQL Cursor

SQL Cursor

A cursor is a private SQL work area of memory in which the SQL statement is parsed and executed. When the executable part of a block issues a SQL statement or DML statement, PL/SQL creates an *implicit cursor* automatically. PL/SQL manages *implicit cursor* automatically.

For queries that return more than one row. *Explicit cursors* are declared and named explicitly by programmers and manipulated through specific statements in the executable block(***Begin … End***).

Using Sql Cursor attributes we can test the outcomes of our SQL statements

SQL%ROWCOUNT : Number of rows affected by the most recent DML SQL statement (an integer value)

SQL%FOUND Boolean attribute that evaluates to TRUE if the most recent SQL statement affects one or more rows

SQL%NOTFOUND Boolean attribute that evaluates to TRUE if the most recent SQL statement does not affect any rows

SQL%ISOPEN Always evaluates to FALSE because PL/SQL closes implicit cursors immediately after they are executed.

SQL Cursor

Let see example of *implicit cursor* using SQL % Row Count (video)

Let see *Explicit cursor functions* in more details. They are basically used to track which record is currently being processed and programmer can manually control them row by row

for *controlling Explicit cursor* we use open,fetch and close statements.

The **OPEN** statement executes the query associated with the cursor, identifies the active set(*The set of rows the cursor holds is called the active set*), and positions the cursor (pointer) before the first row.

The **FETCH** statement retrieves the current row and advances the cursor to the next row.

the **CLOSE** statement disables the cursor after processing the last row in active set.

CURSOR WITH LOOPs using cursor with loops or for loops is basically shortcut because cursor is open and rows are fetched once for each iteration of loop and cursor will close automatically after processing of all the rows.

lets take few examples of **CURSOR WITH(For) LOOPs**

SQL Cursor

Example: To print employee number, name and salary of all employees having salary higher than average salary using PL/SQL Loops with cursor implementation

```
SET SERVEROUTPUT ON
DECLARE
    AVGSAL NUMBER;
    CURSOR emp_cursor IS SELECT empno, ename, sal FROM emp;
    emp_record emp_cursor%ROWTYPE;
BEGIN
    SELECT AVG(SAL) INTO AVGSAL FROM EMP;
    DBMS_OUTPUT.PUT_LINE('Average Salary : ' || AVGSAL);
    OPEN emp_cursor;
    LOOP
        FETCH emp_cursor INTO emp_record;
        EXIT WHEN emp_cursor%NOTFOUND;
        IF emp_record.sal > AVGSAL THEN
            DBMS_OUTPUT.PUT_LINE('Employee Number : ' || emp_record.empno);
            DBMS_OUTPUT.PUT_LINE('Employee Name : ' || emp_record.ename);
            DBMS_OUTPUT.PUT_LINE('Salary : ' || emp_record.sal);
        END IF;
    END LOOP;
    CLOSE emp_cursor; -- closes cursor
END;
```

Definition of a cursor

Declare a variable that has same records as emp_cursor have and cursor elements can only be accessed by means of this varable

fill AVGSAL variable via AVG salaries of emp table

This will fill all records of emp_cursor in emp_record variable
In second example we will perform these steps via for loop
With implicit open and implicit fetch.

SQL Cursor

```
SET SERVEROUTPUT ON
DECLARE
    AVGSAL NUMBER;
    CURSOR emp_cursor IS SELECT empno, ename,sal FROM emp;

BEGIN
    SELECT AVG(SAL) INTO AVGSAL FROM EMP;
    DBMS_OUTPUT.PUT_LINE('Average Salary : ' || AVGSAL);

    FOR emp_record IN emp_cursor LOOP
            -- implicit open and implicit fetch occur
        IF emp_record.sal > AVGSAL THEN
            DBMS_OUTPUT.PUT_LINE('Employee Number : ' || emp_record.empno);
            DBMS_OUTPUT.PUT_LINE('Employee Name : ' || emp_record.ename);
            DBMS_OUTPUT.PUT_LINE('Salary : ' || emp_record.sal);

        END IF;
    END LOOP; -- implicit close occurs
END;
```

Here **emp_record** has automatically (implicitly) defined of **Emp_cursor type** vairable which is accessing the elements of cursor(**emp_cursor**)

Video of Explicit Cursors in class

Finish