Introduction to PL/SQL
UNIT-5

Introduction to PL/SQL

- PL/SQL Stands for Procedural Language for SQL.
- PL/SQL is not Flexible & not easy to learn.
- Data Manipulation not becomes very easy.
- PL/SQL is Procedural Language.
- PL/SQL is Extension of SQL.
- PL/SQL can be used as programming Language and as an application development tool.

Differences between SQL AND PL/SQL

SQL	PL/SQL
1. SQL Stands for Structured Query Language.	1. PL/SQL Stands for Procedural Language for SQL.
2. SQL is Flexible & easy to learn.	2. PL/SQL is not Flexible & not easy to learn.
3. Data Manipulation becomes very easy.	3. Data Manipulation not becomes very easy.
4. SQL is Non Procedural Language.	4. PL/SQL is Procedural Language.
5. SQL is Basic Language.	5. PL/SQL is Extension of SQL.
6. SQL can not be used as programming Language and as an application development tool.	6. PL/SQL can be used as programming Language and as an application development tool.

Features of PL/SQL

1. **Block Structure**
2. **Procedural Capabilities.**
3. **Exception Handling**
4. **Modularity**
5. **Cursors**
6. **Built in Function**
7. **Better Performance**
8. **Portability.**

Advantages of PL/SQL

1. It is not only supports SQL data manipulation but also provides facilities of conditional checking, branching and looping.

2. PL/SQL sends an entire block of statements to the oracle engine at one time.

3. Applications written in PL/SQL ae portable to any computer hardware and operating system.

4. PL/SQL provides facilities to display user friendly messages when errors are encountered.

5. PL/SQL performs all sorts of calculations very efficiently & quickly.

Block Structure of PL/SQL

With PL/SQL, you can use SQL statements to manipulate Oracle data and flow-of-control statements to process the data. You can also 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 Block Structured.**

  A block is the basic unit from which all PL/SQL programs are built. A block can be named (functions and procedures) or anonymous BLOCK and Stored Procedures.

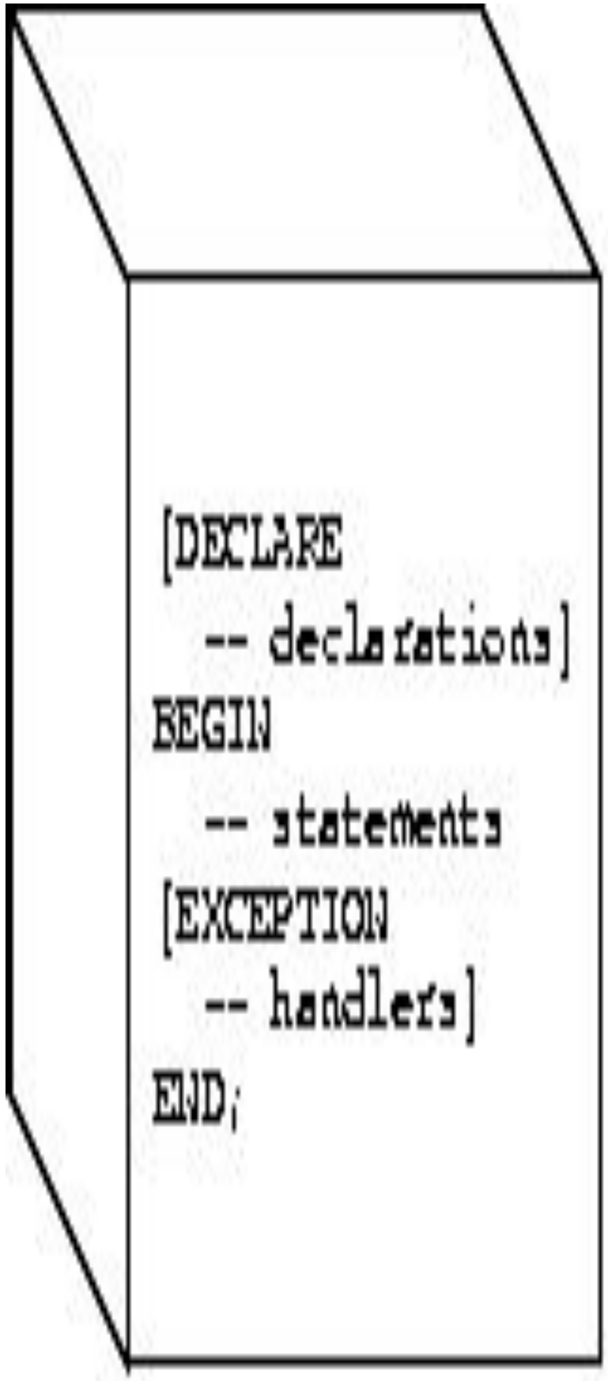
- **Sections of block :**

  1- Declare Section

  2- Begin Section

  3. Exception Section

  4. End Section



```
[DECLARE
  -- declarations]
BEGIN
  -- statements
[EXCEPTION
  -- handlers]
END;
```

Structure of PL/SQL

DECLARE

Declaration of memory Variables; Constants; Cursors;

BEGIN

SQL executable Statements & PL/SQL executable Statements

EXCEPTION

SQL or PL/SQL code to handle errors that may arise

during execution of block.

END;

Description of Blocks

1. **Declare Section :** Code block starts with a declaration section in which memory variables & other oracle objects can be declared & if required initialized. Once, declared they can be used in the SQL statements for data manipulation.

2- **Begin Section :** It contains of a set of SQL & PL/SQL statements, which describe processes that have to be applied to the data actual data manipulation, retrieval, looping & branching constructions are applied through this section.

3. **Exception Section :** It deals with handling of errors that arise during execution of the data manipulation statements, which make up the PL/SQL code block.

4. **End Section** : It is used to make the end of PL/SQL block.

Data Types of PL/SQL

Important Data Types in PL/SQL includes:

1. NUMBER Data Type
2. CHAR Data Type
3. VARCHAR2 Data Type
4. DATE Data Type
5. Boolean Data Type

Variables and Constants

- PL/SQL lets you declare constants and variables, then use them in SQL and procedural statements anywhere an expression can be used. However, forward references are not allowed. So, you must declare a constant or variable *before* referencing it in other statements, including other declarative statements.
- <u>Declaring Variables :</u>
- Variables can have any SQL datatype, such as CHAR, DATE, or NUMBER, or any PL/SQL datatype, such as BOOLEAN or BINARY_INTEGER.
- For example, assume that you want to declare a variable named part_no to hold 4-digit numbers and a variable named in_stock to hold the Boolean value TRUE or FALSE. You declare these variables as follows:
- part_no NUMBER(4);
- in_stock BOOLEAN;

State types of Attributes used in PL/SQL

PL/SQL variables and cursors have *attributes*, which are properties that let you reference the datatype and structure of an item without repeating its definition. Database columns and tables have similar attributes, which you can use to ease maintenance. A percent sign (%) serves as the attribute indicator.

- **<u>%TYPE Attributes</u>**

The %TYPE attribute provides the datatype of a variable or database column. This is particularly useful when declaring variables that will hold database values. For example, assume there is a column named title in a table named books. To declare a variable named my_title that has the same datatype as column title, use dot notation and the %TYPE attribute, as follows:

```
my_title books.title%TYPE;
```

- **%ROWTYPE Attribute:**

In PL/SQL, records are used to group data. A record consists of a number of related fields in which data values can be stored. The %ROWTYPE attribute provides a record type that represents a row in a table. The record can store an entire row of data selected from the table or fetched from a cursor or cursor variable.

Columns in a row and corresponding fields in a record have the same names and datatypes. In the example below, you declare a record named dept_rec. Its fields have the same names and datatypes as the columns in the dept table.

```
DECLARE
   dept_rec dept%ROWTYPE;  -- declare record variable
```

You use dot notation to reference fields, as the following example shows:

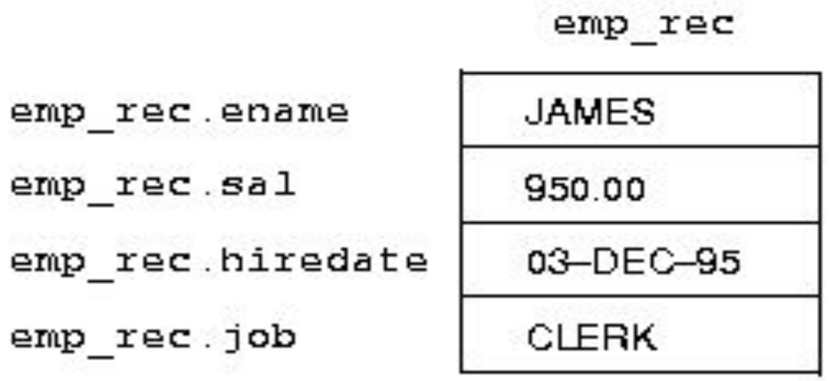
```
my_deptno := dept_rec.deptno;
```

If you declare a cursor that retrieves the last name, salary, hire date, and job title of an employee, you can use %ROWTYPE to declare a record that stores the same information, as follows:

```
DECLARE
  CURSOR c1 IS
    SELECT ename, sal, hiredate, job FROM emp;
  emp_rec c1%ROWTYPE;  -- declare record variable that represents
 a row fetched from the emp table.
```

When you execute the statement   FETCH c1 INTO emp_rec;

The value in the ename column of the emp table is assigned to the ename field of emp_rec, the value in the sal column is assigned to the sal field, and so on. It shows how the result might appear.



Assigning Values to a Variable

You can assign values to a variable in three ways. The first way uses the assignment operator (:=), a colon followed by an equal sign. You place the variable to the left of the operator and an expression A few examples:

- tax := price * tax_rate;
- valid_id := FALSE;
- bonus := current_salary * 0.10;
- wages := gross_pay(emp_id, st_hrs, ot_hrs) - deductions;
- <u>The second way</u> to assign values to a variable is by selecting (or fetching) database values into it.
- SELECT sal * 0.10 INTO bonus FROM emp WHERE empno = emp_id;
- <u>The third way to</u> assign values to a variable is by passing it as an OUT or IN OUT parameter to a subprogram.

  DECLARE

  my_sal REAL(7,2);
- PROCEDURE adjust_salary (emp_id INT, salary IN OUT REAL) IS ...
- BEGIN
- SELECT AVG(sal) INTO my_sal FROM emp;

Declaring Constants

- Declaring a constant is like declaring a variable except that you must add the keyword CONSTANT and immediately assign a value to the constant. Thereafter, no more assignments to the constant are allowed. In the following example, you declare a constant named credit_limit:

- credit_limit CONSTANT REAL := 5000.00;

Explain the basic SQL statements in PL/SQL

There are basically Four SQL statements in PL/SQL

1. INSERT Statement: This statement is used to insert tuple in the database.

Syntax: INSERT INTO (TABLE-NAME) VALUES (VALUE1, VALUE2……)

EXAMPLE : INSERT a record in the emp table.

INSERT INTO Emp values(1002,'JOHN',10000);

INSERT Statement used in PL/SQL is similar to statement used in SQL.

2. UPDATE Statement: This statement is used to update a particular value in the particular tuple of record.-

Syntax: UPDATE(Table-Name) SET(Column-Name) WHERE(condition);

Example:
```
UPDATE Emp SET sal = sal +100
        WHERE Empname='Ram';
```

This statement updates the salary of Ram by Rs. 100.

3. <u>SELECT-INTO Statement:</u> This statement of PL/SQL is little bit different than the SQL statement used in SQL. This statement consists of INTO clause which is used in reference with the user defined variable.

<u>Syntax:</u> SELECT (Column-Name1, Column-Name2……) INTO (User defined variable FROM (Table-Name)

<u>EXAMPLE</u> : SELECT Sum(Sal) INTO total_sal FROM emp;

In this example total_sal is user defined variable which is declared in the declare section of PL/SQL block structure. Here we are selecting the total salary from emp table & putting it in total_sal variable.

4. <u>DELETE Statement:</u> This statement of PL/SQL is similar to statement used in SQL. This statement is used delete particular tuple from table.

<u>Syntax:</u> DELETE FROM (Table-Name) WHERE (Condition)

<u>EXAMPLE</u> : DELETE FROM Emp WHERE Empname = ‘Sita’;

PL/SQL Control Structure

- **PL/SQL has a number of control structures which includes:**
- **Conditional controls**
- **Iterative or loop controls.**
- **Unconditional/Sequential controls**
- **It is these controls, used singly or together, that allow the PL/SQL developer to direct the flow of execution through the program.**

Control structures are the most important PL/SQL extension to SQL. Not only does PL/SQL let you manipulate Oracle data, it lets you process the data using conditional, iterative, and sequential flow-of-control statements such as IF-THEN-ELSE, CASE, FOR-LOOP, WHILE-LOOP, EXIT-WHEN, and GOTO. Collectively, these statements can handle any situation.

PL/SQL Control Structure

- **Conditional Controls :**

1. IF....THEN....END IF;

```
2. LOOP
         ...SQL Statements...
          EXIT;
END LOOP;
```

```
3. WHILE loops
   WHILE condition LOOP
                  ...SQL Statements...
END LOOP;
```

```
4. FOR loops
FOR <variable(numeric)> IN [REVERSE]
<lowerbound>..<upperbound> LOOP .... ..... END LOOP;
```

1.Conditional Control :

```
DECLARE
  acct_balance NUMBER(11,2);
  acct   CONSTANT NUMBER(4) := 3;
  debit_amt    CONSTANT NUMBER(5,2) := 500.00;
BEGIN
  SELECT bal INTO acct_balance FROM accounts
    WHERE account_id = acct  FOR UPDATE OF bal;
IF acct_balance >= debit_amt THEN  UPDATE accounts SET bal = bal -
debit_amt   WHERE account_id = acct;
  ELSE
     INSERT INTO temp VALUES  (acct, acct_balance, 'Insufficient funds');
END IF;
END;
```

Which processes a bank transaction. Before allowing you to withdraw $500 from account 3, it makes sure the account has sufficient funds to cover the withdrawal. If the funds are available, the program debits the account. Otherwise, the program inserts a record into an audit table.

2.Iterative Control:

LOOP statements let you execute a sequence of statements multiple times. You place the keyword LOOP before the first statement in the sequence and the keywords END LOOP after the last statement in the sequence. The example shows the simplest kind of loop, which repeats a sequence of statements continually:

```
LOOP
  -- sequence of statements
END LOOP;
```

The FOR-LOOP statement lets you specify a range of integers, then execute a sequence of statements once for each integer in the range. For example, the following loop inserts 500 numbers and their square roots into a database table:

```
FOR num IN 1..500 LOOP
  INSERT INTO roots VALUES (num, SQRT(num));
END LOOP;
```

3.Sequential/Unconditional Control:

The GOTO statement lets you branch to a label unconditionally. The label, an undeclared identifier enclosed by double angle brackets, must precede an executable statement or a PL/SQL block. When executed, the GOTO statement transfers control to the labeled statement or block, as the following example shows:

```
IF rating > 90 THEN
  GOTO calc_raise;  -- branch to label
END IF;
IF job_title = 'SALESMAN' THEN  -- control resumes here
  amount := commission * 0.25;
ELSE
  amount := salary * 0.10;
END IF;
```

Sample Programs on PL/SQL

1] Write a program to display message

```
create procedure dispmsg
as
begin
print ('Hello');
End
```

For running program type command : exec dispmsg

2] Write a program to Find largest Number

```
create procedure largest @num1 int,@num2 int
as
begin
if (@num1>@num2)
select concat(@num1,' is Greatest')
else
select concat(@num2,' is Greatest')
End
```

exec largest 12,10

3] Write a program to find Even or Odd Number

```
create procedure showeven @num1 int
as
begin
declare @res int
select @res=@num1%2
if(@res=0)
select concat(@num1,' is Even Number')
else
select concat(@num1,' is Odd Number')
End
exec showeven 11
```

4] Write a program to find Addition of Two Number

```
create procedure addnum @num1 int,@num2 int  as
begin
declare @res int
select @res=@num1+@num2
select concat('Addition is', @res)
end
exec addnum 12,13
```

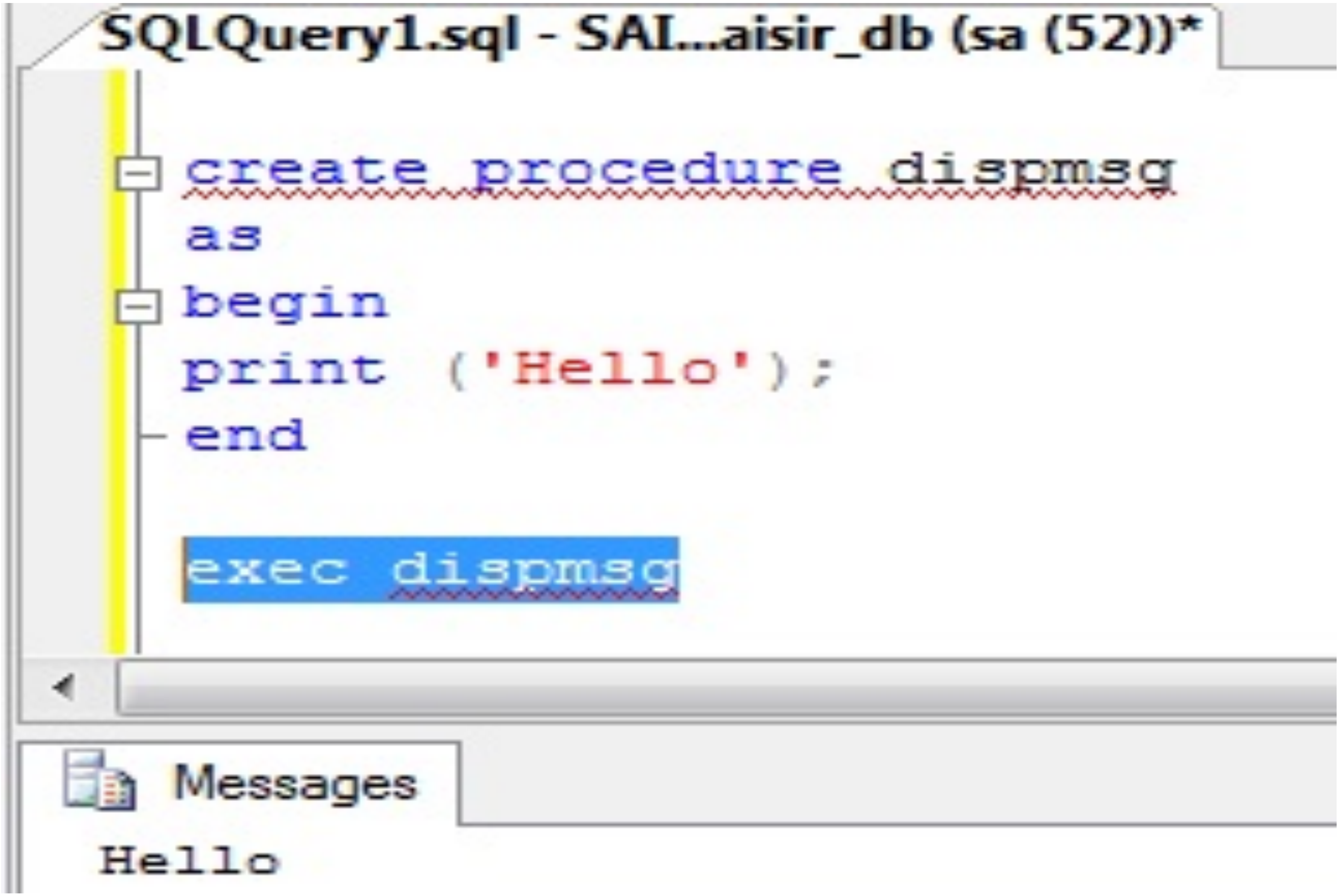
5] <u>Write a program to Calcalute Simple Interest</u>

```
create procedure siminterest @num1 int,@num2 int,@num3 int
as
begin
declare @res int
select @res=@num1*@num2*@num3
select concat('Simple Interest is', @res)
End
exec siminterest 12,3,3
```

6] <u>Write a program to find Circumference of Circle</u>

```
create procedure circumference @radius int
as
begin
declare @res int
select @res=2*3.14*@radius
select concat('circumference of circle is', @res)
End
exec circumference 3
```

Write a program to display simple message in PL/SQL

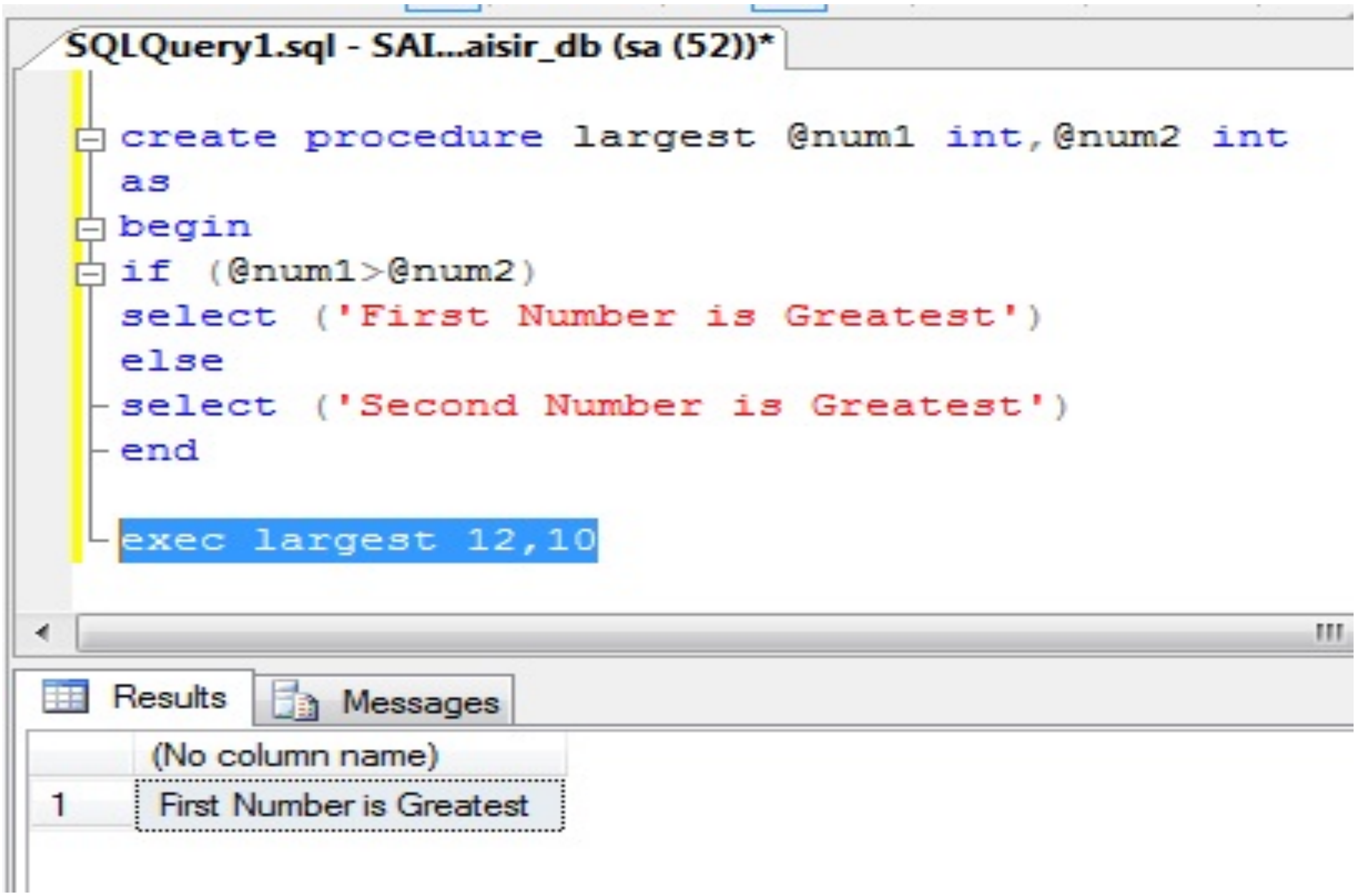


```
create procedure dispmsg
as
begin
print ('Hello');
end

exec dispmsg
```

```
Hello
```

Write a program greatest among two numbers in PL/SQL



SQLQuery1.sql - SAI...aisir_db (sa (52))*

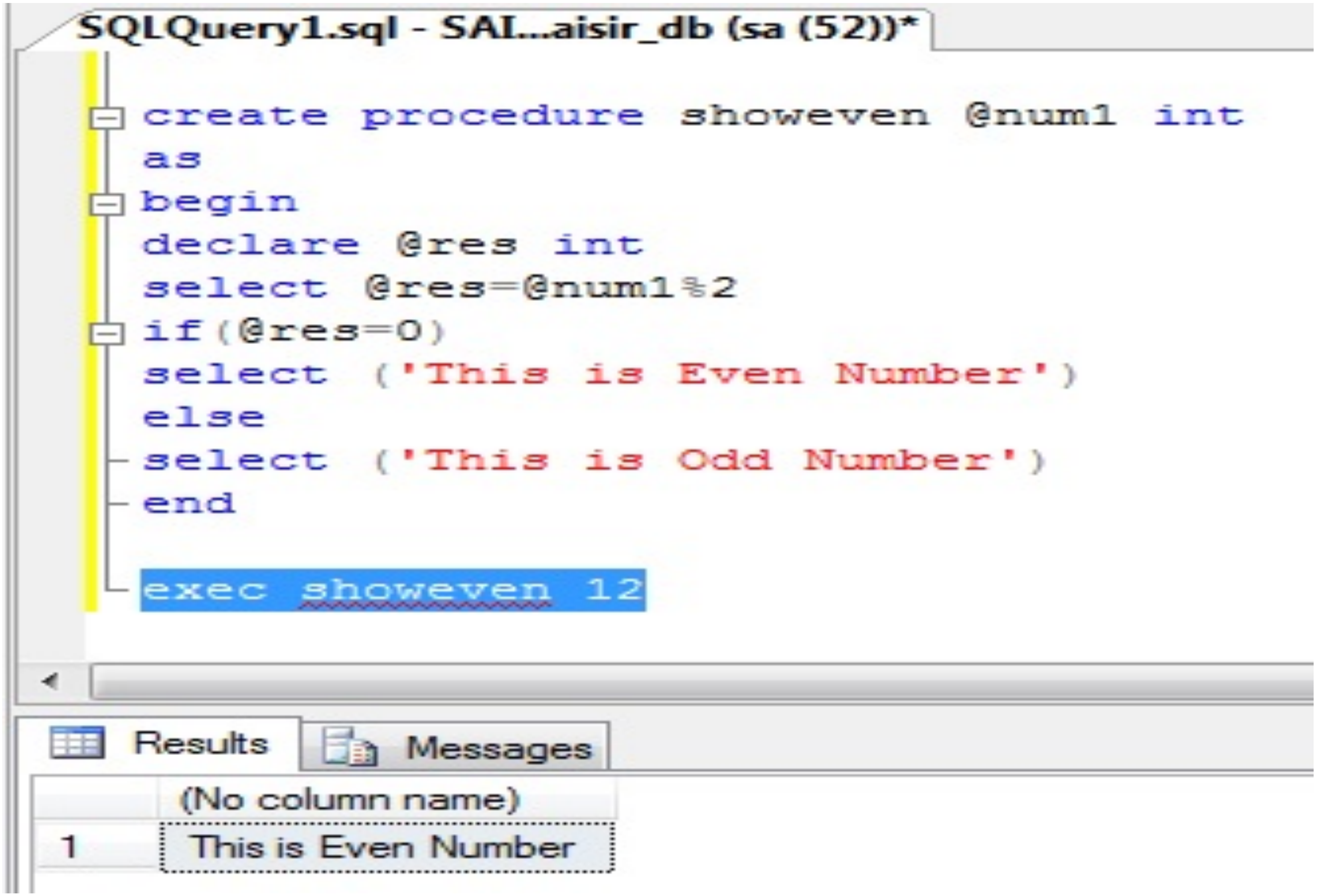
```
create procedure largest @num1 int,@num2 int
as
begin
if (@num1>@num2)
select ('First Number is Greatest')
else
select ('Second Number is Greatest')
end

exec largest 12,10
```

Results | Messages

```
   (No column name)
1  First Number is Greatest
```

Write a program to read a given number is even or odd in PL/SQL


SQLQuery1.sql - SAI...aisir_db (sa (52))*

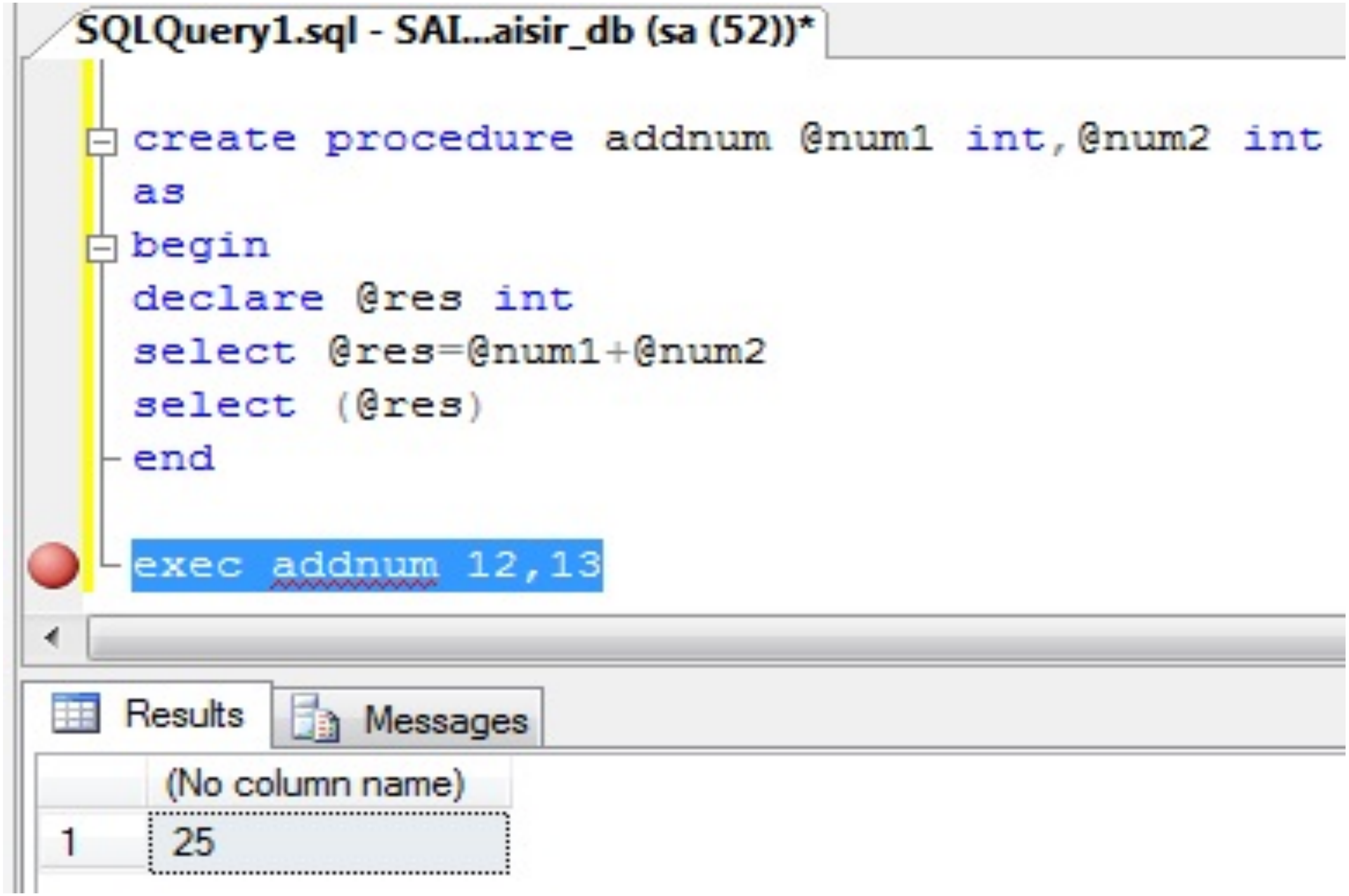
```
create procedure showeven @num1 int
as
begin
declare @res int
select @res=@num1%2
if(@res=0)
select ('This is Even Number')
else
select ('This is Odd Number')
end

exec showeven 12
```

Results | Messages

```
   (No column name)
1  This is Even Number
```

Write a program to Addition of two number in PL/SQL

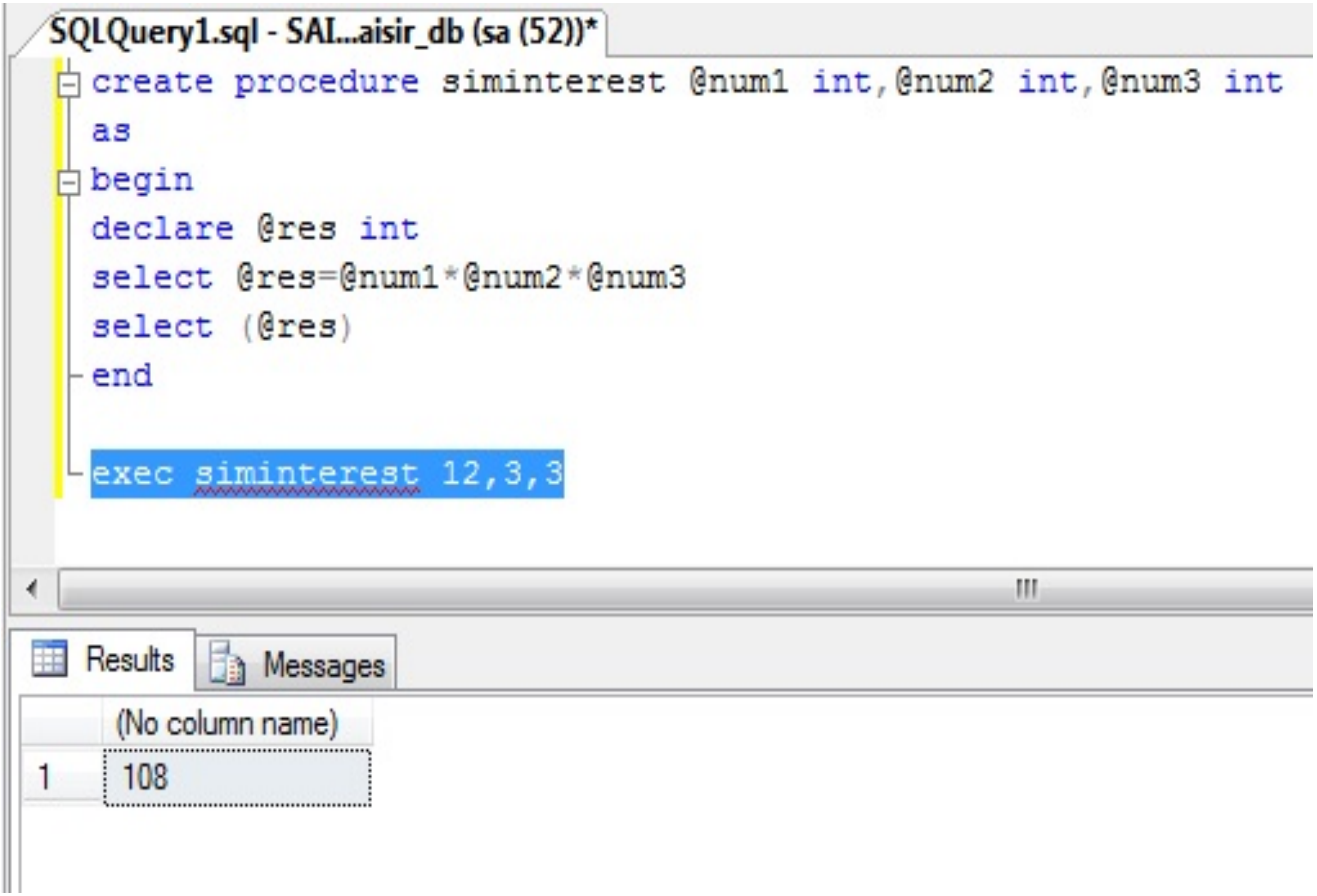


```
create procedure addnum @num1 int,@num2 int
as
begin
declare @res int
select @res=@num1+@num2
select (@res)
end

exec addnum 12,13
```

```
   (No column name)
1  25
```

Write a program to Calculate Simple Interest in PL/SQL



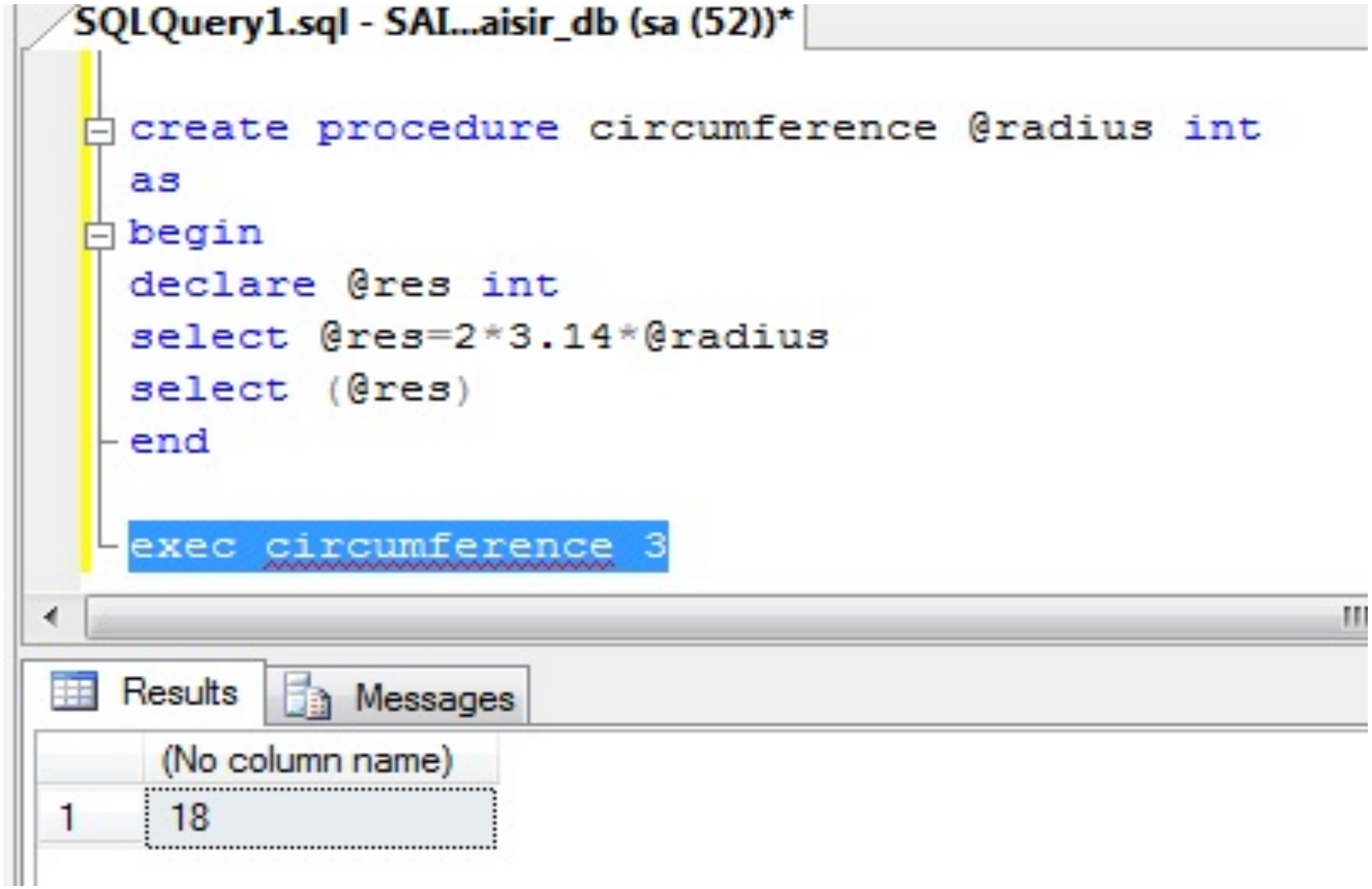
SQLQuery1.sql - SAI...aisir_db (sa (52))*

```
create procedure siminterest @num1 int,@num2 int,@num3 int
as
begin
declare @res int
select @res=@num1*@num2*@num3
select (@res)
end

exec siminterest 12,3,3
```

```
   (No column name)
1  108
```

Write a program to Calculate Circumference of Circle in PL/SQL



SQLQuery1.sql - SAI...aisir_db (sa (52))*

```
create procedure circumference @radius int
as
begin
declare @res int
select @res=2*3.14*@radius
select (@res)
end

exec circumference 3
```

```
   (No column name)
1  18
```

Cursors :

The Oracle engine uses a work area for its internal processing in order to execute an SQL statements. This work area is private to SQL'S operations and is called as Cursor.

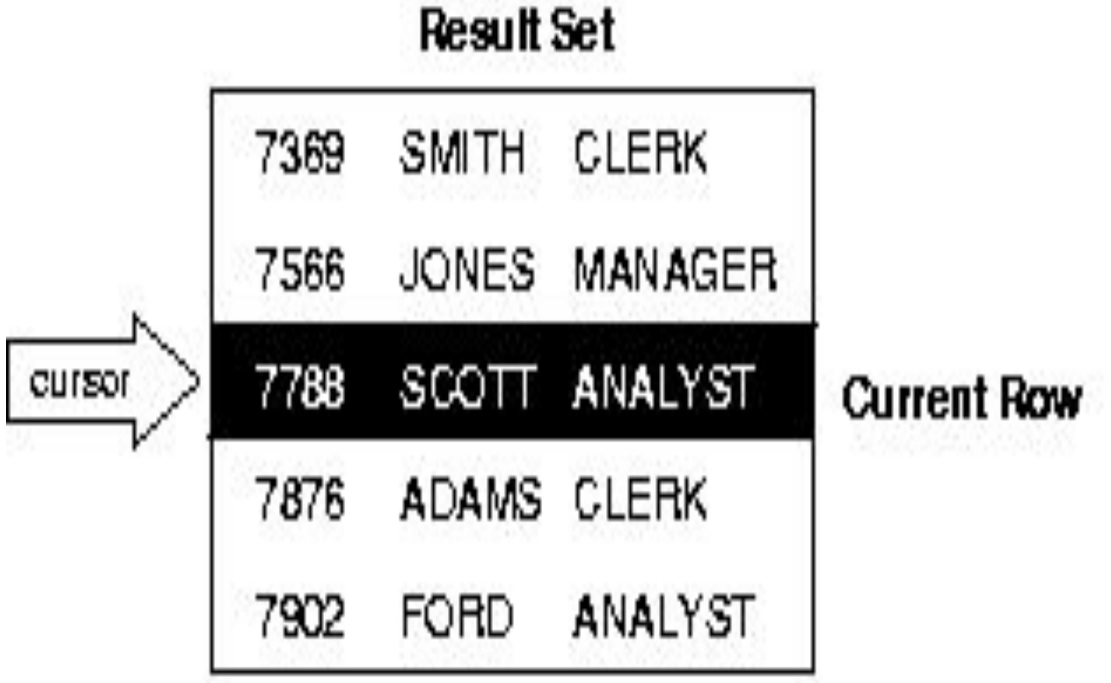
Oracle uses work areas to execute and store processing information. A PL/SQL construct called a *cursor* lets you name a work area and access its stored information.

The data that is stored in the cursor is called the. Conceptually the size of cursor in memory is the required to hold the number of rows in the Active Data Set.

Oracle has a pre-defined area in main memory set aside, within cursor are opened. When a cursor is loaded with multiple rows via a query Oracle engine opens and maintains a row pointer. Depending on user requests to view data the row pointer will be relocated within the cursor's active data set.

There are Two types of cursors which are classified depending on the circumstances under which they are opened.

- Multi-row query processing is somewhat like file processing. PL/SQL program opens a cursor, processes rows returned by a query, then closes the cursor. Just as a file pointer marks the current position in an open file, a cursor marks the current position in a result set.
- You use the <u>OPEN, FETCH, and CLOSE</u> statements to control a cursor. The <u>OPEN statement</u> executes the query associated with the cursor, identifies the result set, and positions the cursor before the first row. The <u>FETCH statement</u> retrieves the current row and advances the cursor to the next row. When the last row has been processed, the <u>CLOSE statement</u> disables the cursor.



Types of cursors:

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 can explicitly declare a cursor to process the rows individually.

Types of cursors: I*mplicit Cursor* & E*xplicit Curosor*.

1. Implicit Cursor : If oracle engine opens a cursor for its internal processing then they are known as Implicit Cursor. Implicit Cursor attributes can be used to access information about the status of last insert, update, delete or single row select statements.

2.Explicit Cursor: A user can also open a cursor for processing of data as required. Such user defined cursors are known as Explicit Cursor. When Explicit Cursor is declared, the oracle engine is informed of the said name needs to be opened.

An example follows:

```
DECLARE
    CURSOR c1 IS
SELECT empno, ename, job FROM emp
WHERE deptno = 20;
```

The set of rows returned by a multi-row query is called the *result set*. Its size is the number of rows that meet your search criteria. As Figure shows, an explicit cursor "points" to the *current row* in the result set. This allows your program to process the rows one at a time.

PL/SQL Control Structure for Cursor

- **Sample Program structure for writing Cursor**

```
DECLARE
          name varchar2(20);
          Cursor c1 is
     select t.name from table t
           where date is not null;
BEGIN
           OPEN c1;
       LOOP
             FETCH c1 into name;
            exit when c1%NOTFOUND;
        END LOOP;
       CLOSE c1;
END;
```

Sample Program on Implicit Cursor

Example 1:

```
Declare Var_rows number(5);
Begin
Update employee
Set salary = salary +100;
End;
```

Example 2:

```
Declare Var_student  number(5);
Begin
Select Name from student where rollno = 100;
End;
```

Sample Program on Explicit Cursor

It is defined in the declaration section of PL/SQL block. It is created on SELECT statement where returns more than one row.

For steps of Explicit cursor:

1. **Declare 2. Open 3. Fetch 4. Close**

Syntax: CURSOR Cursor_name IS Select statement;

Example 1:

```
Declare CURSOR emp_cur IS SELECT* FROM
   emp_tbl  Where Salary > 5000;
End;
```

Example 2:

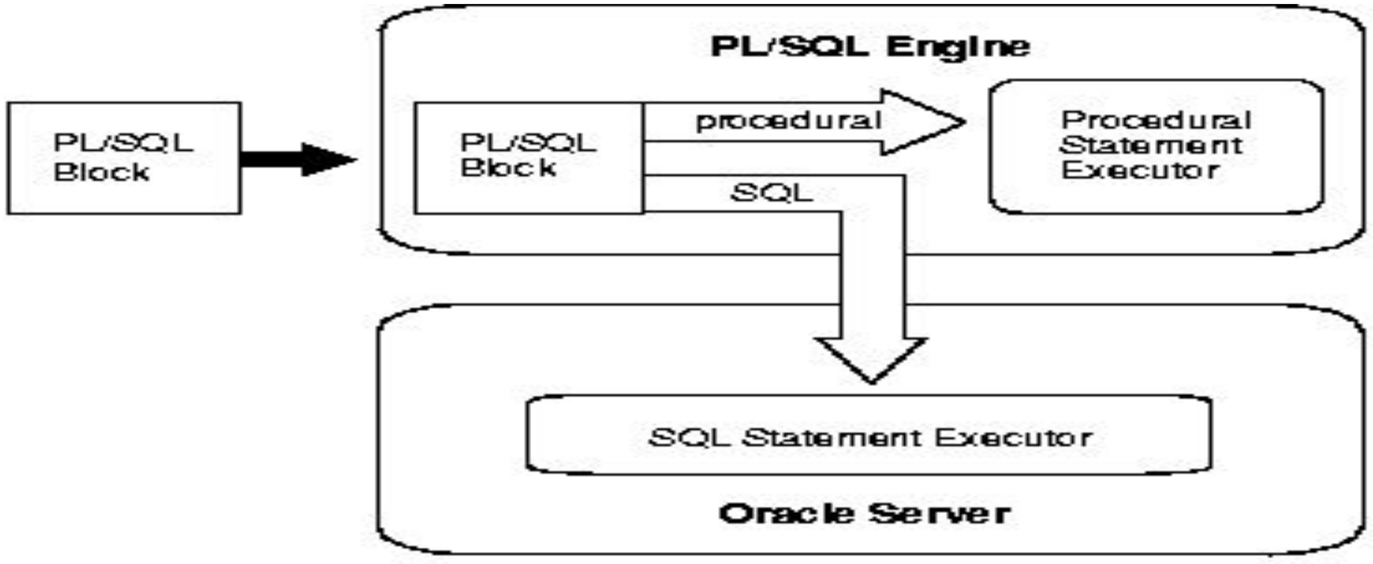
```
DECLARE
Emp_rec emp_tbl %rowtype;
CURSOR emp_cur IS SELECT * FROM
emp_tbl  where salary > 10000;
Begin
OPEN emp_rec;
FETCH emp_cur into emp_rec;
CLOSE emp_cur;
End;
```

PL/SQL Architecture

The PL/SQL compilation and run-time system is a technology, not an independent product. Think of this technology as an engine that compiles and executes PL/SQL blocks and subprograms. The engine can be installed in an Oracle server or in an application development tool such as Oracle Forms or Oracle Reports. So, PL/SQL can reside in two environments:

- The Oracle database server
- Oracle tools

These two environments are independent. PL/SQL is bundled with the Oracle server but might be unavailable in some tools. In either environment, the PL/SQL engine accepts as input any valid PL/SQL block or subprogram. Figure shows the PL/SQL engine processing an anonymous block. The engine executes procedural statements but sends SQL statements to the SQL Statement Executor in the Oracle server.



In the Oracle Database Server

Application development tools that lack a local PL/SQL engine must rely on Oracle to process PL/SQL blocks and subprograms. When it contains the PL/SQL engine, an Oracle server can process PL/SQL blocks & subprograms as well as single SQL statements. Oracle server passes the blocks and subprograms to its local PL/SQL engine.

Anonymous Blocks

Anonymous PL/SQL blocks can be embedded in an Oracle Precompiler or OCI program. At run time, the program, lacking a local PL/SQL engine, sends these blocks to the Oracle server, where they are compiled and executed. Likewise, interactive tools such as SQL*Plus and Enterprise Manager, lacking a local PL/SQL engine, must send anonymous blocks to Oracle.

Assignment Questions Based on UNIT-5

Q.1.. What is PL/SQL? Explain with their features. 5

Q.2. Write ANY FIVE differences between SQL & PL/SQL. 5

Q.3. Write any Five Advantages of PL/SQL. 5

Q.4. State types of attributes used in PL/SQL. 5

Q.5. Explain Data types used in PL/SQL. 5

Q.6. Explain PL/SQL block structures in detail . 5

Q.7. Explain PL/SQL Control structures in detail . 5

Q.8. Explain PL/SQL basics? And explain procedure with their types. 6

Q.9. Draw & Explain Architecture of PL/SQL. 5

Q.10. What is Cursor? Explain with Implicit cursor & Explicit cursor With proper example. 6

Q.11. Write a program to find area of circle in PL/SQL. 5