6. Discuss three tier structure of DBMS.
7. Who is DBA and what are the different roles and responsibilities of DBA ?
8. What do you mean by SQL ? What are the characteristics of SQL ?
9. What are different data types in SQL ? Explain.
10. Differentiate between char and varchar data types.
11. What do you mean by constraints ? Explain.
12. What is Domain integrity constraint ?
13. What is referential integrity constraint ?
14. What do you mean by DML and DDL ? Explain
15. What do you mean by DCL and TCL ? Explain.
16. What do you mean by table ? How can you create a table in SQL ?
17. What is the function of SELECT command ? Explain.
18. How can you insert rows in table ?
19. What is the use of DISTINCT clause in SELECT statement ?
20. What is the purpose of WHERE clause with SELECT statement ?
21. What is the function of ORDER BY clause ? Explain.
22. Discuss the use of AND, OR and NOT operators in SQL.
23. Why BETWEEN operator is used ?
24. How can you remove a table in database ?
25. How can you create users in SQL ?
26. How can you delete user accounts in SQL ?

—End—

CHAPTER - 6

PL/SQL

6.1 Introduction

PL/SQL is an extension of Structured Query Language (SQL) that is used in Oracle. Unlike SQL, PL/SQL allows the programmer to write code in a procedural format. Full form of PL/SQL is "Procedural Language extensions to SQL".PL/SQL is superset of SQL.

It combines the data manipulation power of SQL with the processing power of procedural language to create super powerful SQL queries.

Architecture of PL/SQL

The PL/SQL architecture mainly consists of following three components:

1. PL/SQL block
2. PL/SQL Engine
3. Database Server

Def : PL/SQL is a block structured language that enables developers to combine the power of SQL with procedural statements .All the statements of a block are passed to oracle engine all at once which increases processing speed and decreases the traffic.

Features of PL/SQL :

1. PL/SQL is basically a procedural language, which provides the functionality of decision making, iteration and many more features of procedural programming languages.
2. PL/SQL can execute a number of queries in one block using single command.

3. One can create a PL/SQL unit such as procedures, functions, packages, triggers, and types, which are stored in the database for reuse by applications.
4. PL/SQL provides a feature to handle the exception which occurs in PL/SQL block known as exception handling block.
5. Applications written in PL/SQL are portable to computer hardware or operating system where Oracle is operational.
6. PL/SQL Offers extensive error checking.

1. Better performance, as SQL is executed in bulk rather than a single statement
2. High Productivity
3. Tight integration with SQL
4. Full Portability
5. Tight Security
6. Support Object Oriented Programming concepts.

SQL doesn't provide the programmers with a technique of condition checking, looping and branching.

SQL statements are passed to Oracle engine one at a time which increases traffic and decreases speed.

SQL has no facility of error checking during manipulation of data.

6.2 PL/SQL Basics

Like other programming languages, PL/SQL has a character set, reserved words, punctuation, datatypes, and fixed syntax rules.

PL/SQL programs are written as lines of text using a specific set of characters:

- Upper- and lower-case letters A .. Z and a ..z
- Numerals 0 .. 9
- Symbols ( ) + – * / <>= ! ~ ^ ; : . ' @ % , " # $ & _ | { } ? [ ]
- Tabs, spaces, and carriage returns

PL/SQL keywords are not case-sensitive, so lower-case letters are equivalent to corresponding upper-case letters except within string and character literals.

6.3 PL/SQL Block Structure:

PL/SQL extends SQL by adding constructs found in procedural languages, resulting in a structural language that is more powerful than SQL. The basic unit in PL/SQL is a block. All PL/SQL programs are made up of blocks, which can be nested within each other.

Typically, each block performs a logical action in the program. A block has the following structure :

```
DECLARE
declaration statements;
BEGIN
executable statements
EXCEPTIONS
exception handling statements
END;
```

- Declare section starts with **DECLARE** keyword in which variables, constants, records as cursors can be declared which stores data temporarily. It basically consists definition of PL/SQL identifiers. This part of the code is optional.
- Execution section starts with **BEGIN** and ends with **END** keyword. This is a mandatory section and here the program logic is written to perform any task like loops and conditional statements. It supports all DML commands, DDL commands and SQL*PLUS built-in functions as well.

- Exception section starts with **EXCEPTION** keyword. This section is optional which contains statements that are executed when a run-time error occurs. Any exceptions can be handled in this section.

6.4 SQL VS PL/SQL

SQL	PL/SQL
1. **No Procedural Capabilities :** SQL does not have procedural capabilities i.e. SQL does not provide the programming techniques of conditional checking, looping and branching.	1. **Procedural Capabilities :** PL/SQL is a procedural extension of to SQL. PL/SQL supports programming techniques of conditional checking, looping and branching.
2. **Time Consuming Processing :** SQL statements are passed to Oracle Engine one at a time. Each time a SQL statement is executed, a call is made to engine's resources. This adds to the traffic on the network and thereby decreasing the speed of processing.	2. **Reduced Network Traffic :** PL/SQl statements sends an entire block of statements at one time. This reduces network time considerably.
3. **No Error handling Procedures :** While processing an SQL statement, if an error occurs, the Oracle engine displays its own error messages. SQL has no facility for programmed handling of errors that arise during manipulation of data.	3. **Error Handling Procedures :** PL/SQL supports error handling routines.
4. **No facilities Sharing :** SQL does not allow sharing of facilities.	4. **Facilties Sharing :** PL/SQL allows in database thereby providing access and sharing of same subprogram by multiple applications.

6.5 PL/SQL identifiers

There are several PL/SQL identifiers such as variables, constants, procedures, cursors, triggers etc.

1. **Variables** : Like several other programming languages, variables in PL/SQL must be declared prior to its use. They should have a valid name and data type as well.

Syntax for declaration of variables :

variable_namedatatype [NOT NULL := value];

Example to show how to declare variables in PL/SQL :

```
SQL> SET SERVEROUTPUT ON;
SQL> DECLARE
    var1 INTEGER;
    var2 REAL;
    var3 varchar2(20) ;
BEGIN
    null;
END;
/
```

Output :

```
PL/SQL procedure successfully completed.
```

Explanation :

- **SET SERVEROUTPUT ON** : It is used to display the buffer used by the dbms_output.
- **var1 INTEGER :** It is the declaration of variable, named ***var1*** which is of integer type. There are many other data types that can be used like float, int, real, smallint, long etc. It also supports variables used in SQL as well like NUMBER(prec, scale), varchar, varchar2 etc.
- **PL/SQL procedure successfully completed :** It is displayed when the code is compiled and executed successfully.

- **Slash (/) after END;**: The slash (/) tells the SQL*Plus to execute the block.

6.6 Control flow statement in PL SQL

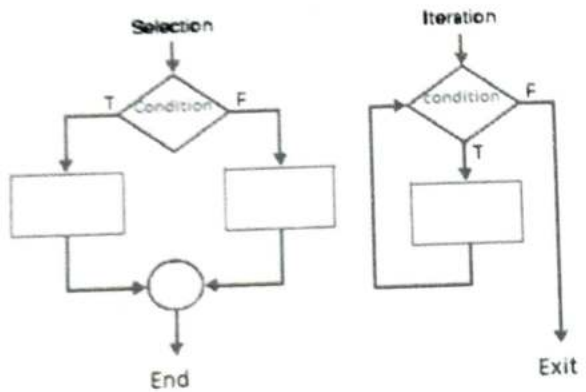
Control Structures in **PL/SQL**

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.

PL/SQL supports groups of execution control statements :

- IF Statements - Conditionally executes a block of statements.
- LOOP Statements - Repeatedly executes a block of statements

Control Structures



6.7 IF Statements :

The IF statement executes a sequence of statements depending on the value of a condition.

There are three forms of IF statements :

- IF-THEN
- IF-THEN-ELSE
- IF-THEN-ELSIF.

IF type	Characteristics
IF THEN END IF;	This is the simplest form of the IF statement. The condition between IF and THEN determines whether the set of statements between THEN and END IF should be executed. If the condition evaluates to FALSE, the code is not executed.
IF THEN ELSE END IF;	This combination implements an either/or logic: based on the condition between the IF and THEN keywords, execute the code either between THEN and ELSE or between ELSE and END IF. One of these two sections of executable statements is performed.
IF THEN ELSIF ELSE END IF;	This last and most complex form of the IF statement selects a condition that is TRUE from a series of mutually exclusive conditions and then executes the set of statements associated with that condition. If you're writing IF statements like this using any release from Oracle9*i* Database Release 1 onwards, you should consider using searched CASE statements instead.

6.7.1 Using the IF-THEN Statement

The simplest form of IF statement associates a condition with a sequence ofstatements enclosed by the keywords THEN and END IF (not ENDIF)

The sequence of statements is executed only if the condition is TRUE. If the condition is FALSE or NULL, the IF statement does nothing. In either case, control passes to the next statement.

Syntax :

IF < Condition > THEN
    < Action >
END IF;

OR

IF *condition*
THEN
... *sequence of executable statements* ...
END IF;

Write a program in PL/SQL to check the given value is greater than 50 by using IF-THEN Statement.

```
DECLARE
a NUMBER :=60;
BEGIN
dbms_output.put_line('Program started.' );
IF( a > 50 ) THEN
dbms_output.put_line('a is greater than 50');
END IF;
dbms_output.put_line('Program completed.');
END;
/
```

Output:

a is greater than 50

6.7.2 Using the IF-THEN-ELSE Statement

Use the IF-THEN-ELSE format when you want to choose between two mutually exclusive actions. The format of this either/or version of the IF statement is as follows:

Syntax :

IF *condition*
THEN
... *TRUE sequence of executable statements* ...
ELSE
... *FALSE/NULL sequence of executable statements* ...
END IF;

The *condition* is a Boolean variable, constant, or expression. If *condition* evaluates to TRUE, the executable statements found after the

THEN keyword and before the ELSE keyword are executed (the "TRUE sequence of executable statements"). If *condition* evaluates to FALSE or NULL, the executable statements that come after the ELSE keyword and before the matching END IF keywords are executed (the "FALSE/NULL sequence of executable statements").Notice that the ELSE clause does not have a THEN associated with it.

The important thing to remember is that one of the two sequences of statements will *always* execute, because IF-THEN-ELSE is an either/or construct. Once the appropriate set of statements has been executed, control passes to the statement immediately following the END IF keyword.

Write a program in PL/SQL to check the given value is greater or less than 50 by using IF-THEN-ELSE Statement.

```
DECLARE
a NUMBER :=30;
BEGIN
dbms_output.put_line('Program started.' );
IF( a > 50 ) THEN
dbms_output.put_line('a is greater than 50');
ELSE
dbms_output.put_line('a is less than 50');
END IF;
dbms_output.put_line('Program completed.');
END;
/
```

Output :

a is less than 50

6.7.3 Using the IF-THEN-ELSIF Statement

This last form of the IF statement comes in handy when you have to implement logic that has many alternatives; it is not an either/or situation. The IF-ELSIF formulation provides a way to handle multiple conditions within a single IF statement. In general, you should use ELSIF with mutually exclusive alternatives (i.e., only one condition can be TRUE

for any execution of the IF statement). The general format for this variation of IF is :

Syntax :

IF *condition-1*
THEN
statements-1
ELSIF *condition-N*
THEN
statements-N
[ELSE
else_statements]
END IF;

Each ELSIF clause must have a THEN after its *condition*. Only the ELSE keyword does not need the THEN keyword. The ELSE clause in the IF-ELSIF is the "otherwise" of the statement. If none of the conditions evaluate to TRUE, the statements in the ELSE clause are executed. But the ELSE clause is optional. You can code an IF-ELSIF that has only IF and ELSIF clauses. In such a case, if none of the conditions are TRUE, no statements inside the IF block are executed.

The conditions in the IF-ELSIF are always evaluated in the order of first condition to last condition. If two conditions evaluate to TRUE, the statements for the first such condition are executed.

Write a program in PL/SQL to display value of variable by using IF-THEN-ELSIF Statement.

```
DECLARE
a number(3):=60;
BEGIN
  IF ( a=10) THEN
dbms_output.put_line('Value of a is 10');
  ELSIF ( a=20) THEN
dbms_output.put_line('Value of a is 20');
  ELSIF ( a=30) THEN
dbms_output.put_line('Value of a is 30');
  ELSE
```



```
dbms_output.put_line('None of the values is matching');
END IF;
dbms_output.put_line('Exact value of a is: '|| a );
END;
/
```

Output :

None of the values is matching

Exact value of a is: 60

6.8 LOOP Statements/Iterative Control :

LOOP statements/Iterative control executes a sequence of statements multiple times or indicates the ability to repeat or skip sections of a code block.

A **loop** marks a sequence of statements that has to be repeated. The keyword loop has to be placed before the first statement in the sequence of statements to be repeated, while the keyword end loop is placed immediately after the last statement in the sequence.

Once a loop begins to execute, it will go on forever. Hence a conditional statement that controls the number of times a loop is executed always accompanies loops.

Types of Loop in PL/SQL

PL/SQL provides following three types of loops

- Basic loop statement
- For loop statement
- While loop statement

6.8.1 Basic LOOP Statement

The simplest form of LOOP statement is the basic (or infinite) loop, its execution block starts with keyword 'LOOP' and ends with the keyword 'END LOOP'.which encloses a sequence of statements between the keywords LOOP and END LOOP.

Syntax :

```
LOOP
sequence_of_statements
END LOOP;
```

With each iteration of the loop, the sequence of statements is cuted, then control resumes at the top of the loop. If further cessing is undesirable or impossible, you can use an EXIT statement complete the loop. **You can place one or more EXIT statements where inside a loop, but not outside a loop.**

Loop Syntax with Exit Statement :

```
LOOP
<execution block starts>
<EXIT condition based on developer criteria>
<execution_block_ends>
END LOOP;
```

Syntax Explanation :

In the above syntax, key word 'LOOP' marks the beginning of the loop and 'END LOOP' marks the end of the loop.

The execution block contains all the code that needs to be executed including the EXIT condition.

The execution part can contain any execution statement.

Note : Basic loop statement with no EXIT keyword will be an ITE-LOOP that will never stop.

Example : Program in PL/SQL code to print number from 1 to 5 basic loop statement.

```
1.  DECLARE
2.  a NUMBER :=1;
3.  BEGIN
4.  dbms_output.put_line('Program started');
5.  LOOP
6.  dbms_output.put_line(a);
7.  a:=a+1;
8.  EXIT WHEN a>5;
9.  END LOOP;
10. dbms_output.put_line('Program completed');
11. END;
12. /
```

Basic Loop

Output:

```
Program started
1
2
3
4
```


Program Explanation :

- **Code line 2**: Declaring the variable 'a' as 'NUMBER' data type and initializing it with value '1'.
- **Code line 4**: Printing the statement "Program started".
- **Code line 5:** Keyword 'LOOP' marks the beginning of the loop.
- **Code line 6:** Prints the value of 'a'.
- **Code line 7:** Increments the value of 'a' by +1.
- **Code line 8:** Checks whether the value of 'a' is greater than 5.
- **Code line 9:** Keyword 'END LOOP' marks the end of execution block.
- The code from line 6 to line 8 will continue to execute till 'a' reaches the value 6, as the condition will return TRUE, and the control will EXIT from the loop.
- **Code line 10:** Printing the statement "Program completed"

6.8.2 For loop statement :

Simple **FOR loops** iterate over a specified range of integers. The number of iterations is known before the loop is entered. A double dot (..) serves as the range operator. The range is evaluated when the FOR loop is first entered and is never re-evaluated. If the lower bound equals the higher bound, the loop body is executed once.

In this loop, the lower limit and the higher limit will be specified and as long as the loop variable is in between this range, the loop will be executed.The loop variable is self-incremental, so no explicit increment operation is needed in this loop. The loop variable need not to be declared, as it is declared implicitly.

Syntax :

```
FOR <loop_variable> in <lower_limit> ..<higher_limit>
LOOP
<execution block starts>
.
.
.
<execution_block_ends>
 END LOOP;
```

Syntax Explanation

- In the above syntax, keyword 'FOR' marks beginning of the loop and 'END LOOP' marks the end of the loop.
- Loop variable is evaluated every time before executing the execution part.
- The execution block contains all the code that needs to be executed. The execution part can contain any execution statement.
- The loop variable is declared implicitly during the execution of the entire loop, and the scope of this loop variable will be only inside this loop.
- If the loop variable came out of the range, then control will exit from the loop.
- The loop can be made to work in the reverse order by adding the keyword 'REVERSE' before lower limit.

Example Program in PL/SQL code to print number from 1 to 5 using FOR loop statement.

```
1.  BEGIN
2.  dbms_output.put_line('Program started');
3.  FOR a IN 1..5
4.  LOOP
5.  dbms_output.put_line(a);
6.  END LOOP;
7.  dbms_output.put_line('Program completed');
8.  END;
9.  /
```

Range Specification

FOR Loop

Output:

```
Program started
1
2
3
4
5
Program completed
```

Program Explanation

- **Code line 2**: Printing the statement "Program started"

- **Code line 3**: Keyword 'FOR' marks the beginning of the loop and loop variable 'a' is declared. It now will have the value starting from 1 to 5.
- **Code line 5**: Prints the value of 'a'.
- **Code line 6**: Keyword 'END LOOP' marks the end of execution block.
- The code from line 5 will continue to execute till 'a' reaches the value 6, as the condition will fail, and the control will EXIT from the loop.
- **Code line 7**: Printing the statement "Program completed"

6.8.3 WHILE LOOP Statement

WHILE loop statement works similar to the Basic loop statement except **the EXIT condition is at the very beginning of the loop**.

It works like an entry-check loop in which execution block will not even be executed once if the condition is not satisfied, as the exit condition is checking before execution part. It does not require keyword 'EXIT' explicitly to exit from the loop since it is validating the condition implicitly each time of the loop.

Syntax :

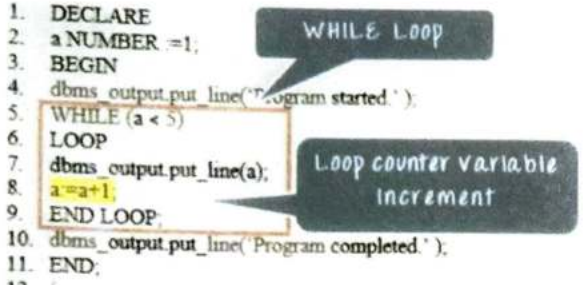
WHILE <EXIT condition>
LOOP
<execution block starts>

<execution_block_ends>
END LOOP;

- In the above syntax, keyword 'WHILE' marks beginning of the loop and 'END LOOP' marks the end of the loop.
- EXIT condition is evaluated each time before the execution part is starting executing.

- The execution block contains all the code that needs to be executed.
- The execution part can contain any execution statement.

Example : Program in PL/SQL code to print number from 1 to 5 using WHILE loop statement.




```
1.  DECLARE
2.  a NUMBER :=1;
3.  BEGIN
4.  dbms_output.put_line('Program started.');
5.  WHILE (a < 5)
6.  LOOP
7.  dbms_output.put_line(a);
8.  a:=a+1;
9.  END LOOP;
10. dbms_output.put_line('Program completed.');
11. END;
12. /
```

Output:

```
Program started
1
2
3
4
5
Program completed
```

- **Code line 2**: Declaring the variable 'a' as 'NUMBER' data type and initializing it with value '1'.
- **Code line 4**: Printing the statement "Program started".
- **Code line 5:** Keyword 'WHILE' marks the beginning of the loop, and it also checks whether the value of 'a' is greater than 5
- **Code line 7:** Prints the value of 'a'.
- **Code line 8:** Increments the value of 'a' by +1.
- **Code line 9:** Keyword 'END LOOP' marks the end of execution block.
- The code from line 7 and line 8 will continue to execute till 'a' reaches the value 6, as the condition will return TRUE, and the control will EXIT from the loop.
- **Code line 10:** Printing the statement "Program completed"

6.9 Trigger

A trigger is a PL/SQL block structure which is fired when a DML statements like INSERT, DELETE, UPDATE is executed on a database table. A trigger is triggered automatically when an associated DML statement is executed. Triggers are basically PL/SQL procedures that are associated with tables, and are called whenever a certain modification (event) occurs. The modification statements may include INSERT, UPDATE, and DELETE. Triggers are stored programs, which are automatically executed or fired when some events occur.

Triggers are, in fact, written to be executed in response to any of the following events :

- A database manipulation (DML) statement (DELETE, INSERT, or UPDATE).
- A database definition (DDL) statement (CREATES, ALTER, or DROP).
- A database operation (SERVERERROR, LOGON, LOGOFF, STARTUP, or SHUTDOWN).

Triggers could be defined on the table, view, schema, or database with which the event is associated.

Benefits of Triggers

Triggers can be written for the following purposes :

- Generating some derived column values automatically
- Enforcing referential integrity
- Event logging and storing information on table access
- Auditing
- Synchronous replication of tables
- Imposing security authorizations
- Preventing invalid transactions.

Different types of triggers in PL/SQL

PL/SQL has four types of triggers which are given below.

1. **Row level Triggers and Statement level Triggers :** A statement level trigger is fired once on behalf of the triggering statement, regardless of the number of rows in the table that the triggering statement affects. A row trigger fires once for each row affected by the triggering event.
2. **BEFORE and AFTER Triggers :** BEFORE triggers run the trigger action before the triggering statement is run. BEFORE trigger execute before the triggering DML statement (INSERT, UPDATE, DELETE) execute. AFTER triggers run the trigger action after the triggering statement is run. AFTER trigger execute after the triggering DML statement (INSERT, UPDATE, DELETE) executed.
3. **INSTEAD OF Triggers** : INSTEAD OF triggers describe how to perform insert, update, and delete operations against views that are too complex to support these operations natively. INSTEAD OF triggers allow applications to use a view as the sole interface for all SQL operations (insert, delete, update and select).

Triggers on System Events and User Events : You can use triggers to publish information about database events to subscribers. System events are for example Database startup and shutdown, Data Guard role transitions etc and User Events are User logon and logoff, DDL statements (CREATE, ALTER, and DROP) etc.

6.10 Cursor

We can manipulate the information within a SQL statement by means of assigning a name to a "select statement", this concept is known a **cursor**. A **cursor** is used for processing individual rows returned as a result for a query.Cursor is the work area which Oracle reserves for internal processing of SQL statements. This work area is private for oracles reserved are called cursor.A cursor holds the rows (one or more) returned by a SQL statement. The set of rows the cursor holds is referred to as the **active set**.

You can name a cursor so that it could be referred to in a program to fetch and process the rows returned by the SQL statement, one at a time. There are two types of cursors :

1. Implicit Cursors
2. Explicit Cursors

1. **Implicit Cursors :** Implicit cursors are automatically created by PL/SQL whenever an SQL statement is executed. Programmers cannot control the implicit cursors and the information in it.

Whenever a DML statement (INSERT, UPDATE and DELETE) is issued, an implicit cursor is associated with this statement. For INSERT operations, the cursor holds the data that needs to be inserted. For UPDATE and DELETE operations, the cursor identifies the rows that would be affected.

Different attributes of implicit cursor are

%FOUND, %ISOPEN, %NOTFOUND, and %ROWCOUNT.

Attribute	Description
%FOUND	Returns TRUE if an INSERT, UPDATE, or DELETE statement affected one or more rows or a SELECT INTO statement returned one or more rows. Otherwise, it returns FALSE.
%NOTFOUND	The logical opposite of %FOUND. It returns TRUE if an INSERT, UPDATE, or DELETE statement affected no rows, or a SELECT INTO statement returned no rows. Otherwise, it returns FALSE.
%ISOPEN	Always returns FALSE for implicit cursors, because Oracle closes the SQL cursor automatically after executing its associated SQL statement.
%ROWCOUNT	Returns the number of rows affected by an INSERT, UPDATE, or DELETE statement, or returned by a SELECT INTO statement.

Explicit Cursors

The Explicit cursors are defined by the programmers to gain more control over the context area. These cursors should be defined in the declaration section of the PL/SQL block. It is created on a SELECT statement which returns more than one row.

Syntax of explicit cursor

CURSOR cursor_name **IS** select_statement;

Steps :

You must follow these steps while working with an explicit cursor.

1. **Declare the cursor** : It is used to initialize in the memory anddefines the cursor with a name and the associated SELECT statement.
2. **Open the cursor** : It is used to allocate memory for the cursor and make it easy to fetch the rows returned by the SQL statements into it.
3. **Fetch the cursor** : It is used toretrieve data by access one row at a time.

   These are now called as active sets. Fetching data from the cursor is a record-level activity that means we can access the data in a record-by-record.
4. **Close the cursor** : It is used to release the allocated memory.Once all the record is fetched, we need to close the cursor so that the memory allocated to this context area will be released.

6.11 Trigger Vs. Cursor

TRIGGER	CURSOR
Triggers are named PL SQL blocks.	**Cursors** are temporary work areas.
Triggers are stored in database.	**Cursors**are not stored independently in the database.
Triggers can be invoked automatically.	We can create cursor both implicitly and explicitly.
Triggers cannot take parameters.	Cursors can take parameters.
Triggers are used to enforce data in dignity rules.	**Cursors** are used to process the result of query.

SUMMARY

- PL/SQL stands for Procedural Language/Structured Query Language.
- PL/SQL is a superset of SQL.
- PL/SQL is a Block Structure language that enables developer to combine the power of SQL with Procedural statement.
- Applications written in PL/SQL are portable to any computer hardware and operating system where oracle is operational.
- PL/SQL divided into logical block.
- PL/SQL Block Structure used four sections: Declare,Begin, Exceptionand End.
- The Declare Section start with Declare Keyword in which memory variable initialized.
- The Begin Section start with Begin Keyword in which consist set of SQL statement to manipulate data in the database.
- The Exception Section start with Exception Keyword, that handles error occur during execution..
- The End statement marks the end of PL/SQL Block.
- Trigger defines an action that database should take, when some database related events occur
- Triggers fired automatically when database events occur.
- A row level trigger is fired each time a row in the table is affected by the triggering statement.
- A Statement trigger is fired once on behalf of the triggering statements, independent of the number of rows, the triggering statements effected.
- Before trigger execute the trigger action before the triggering statement.