SEQUENCE:

A sequence is a database object, which can generate unique, sequential integer values. It can be used to automatically generate primary key or unique key values.
A sequence can be either in an ascending or descending order.

Syntax:

Create sequence <seq_name> [increment bty n] [start with n] [maxvalue n] [minvalue n] [cycle/nocycle] [cache/nocache];

By defalult the sequence starts with 1, increments by 1 with minvalue of 1 and with nocycle, nocache. Cache option pre-alloocates a set of sequence numbers and retains them in memory for faster access.

Ex:

```
SQL> create sequence s;
SQL> create sequence s1 increment by 10 start with 100 minvalue 5 maxvalue 200 cycle cache 20;
```

USING SEQUENCE

```
SQL> create table student(no number(2),name varchar(10));
SQL> insert into student values(s.nextval, 'saketh');
```

Initially currval is not defined and nextval is starting value.
After that nextval and currval are always equal.

CREATING ALPHA-NUMERIC SEQUENCE

```
SQL> create sequence s start with 111234;
SQL> Insert into student values (s.nextval || translate
(s.nextval,'1234567890','abcdefghij'));
```

ALTERING SEQUENCE

- We can alter the sequence to perform the following.
- Set or eliminate minvalue or maxvalue.
- Change the increment value.
- Change the number of cached sequence numbers.

Ex:

```
SQL> alter sequence s minvalue 5;
SQL> alter sequence s maxvalue 100;
SQL> alter sequence s increment by 2;
SQL> alter sequence s cache 10;
```

DROPPING SEQUENCE

```
SQL> drop sequence s;
```

SYNONYM:

A synonym is a database object, which is used as an alias for a table, view or sequence.

Types of Synonym

- Private
- Public

Private synonym is available to the particular user who creates.

Public synonym is created by DBA which is available to all the users.

ADVANTAGES

Hide the name and owner of the object.

Provides location transparency for remote objects of a distributed database.

CREATE AND DROP:

```
SQL> create synonym s1 for emp;
SQL> create public synonym s2 for emp;
SQL> drop synonym s1;
SQL> drop public synonym s2;
```

PL / SQL

Introduction of PL/SQL
PL/SQL Block structure
Control structures

- **Select control structure:-**
  - **If – then**
  - **If –then - else**
  - **If – then - elseif**
  - **Goto**
  - **Case**
- **Loop control structure:-**
  - **While**
  - **For**
  - **Loop**

PROCEDURES
FUNCTIONS
Exceptions

- **Pre-defined Exceptions**
- **User-defined Exceptions**

TRIGGERS
CURSORS
PACKAGES

Introduction of PL/SQL

PL/SQL is procedural language this is the component of ORACLE
PL/SQL is used to develop the programs to manipulate the data stored in the database.
PL/SQL Block:

Declare
Variable declaration;
Begin
Execution statement;
Exception
Exceptional statement;
End;

The PL/SQL block is divided into 3 section

- Declare
- Begin
- Exception

- Declare: - The variable for the program is declared in the Declare section
- Begin: - The actual logic for the program is declared in the Declare section
- Exception: - The statement for error handling is written in exception.

The declare section and exception section is optional
The editor foe PL/SQL program is notepad

The **syntax** for create new file
SQL> define_editor=notepad (windows) (or) SQL> define_editor=vi (linux)
SQL> ed filename .sql (or) SQL> edit filename .sql

To executor PL/SQL program the command is
SQL>start filename.sql (Or) SQL>@filename.sql

Number of line displays then press /

Output function in Pl/SQL is
Syntax:-
Dbms_output.put_line ('message or variable');

Ex:-

```
Dbms_output.put_line ('RAMU');
Dbms_output.put_line (a);
Dbms_output.put_line ('a is '|| a);
```

The input function in PL/SQL is '&'

Syntax :-

```
Var_name  <var_type>;
var_name:=&var_name;
```

Variable:- variable is a value that can be modified during the execution of program

Syntax:-

```
var_name datatype;
```

Ex:-

```
Rno number;
Name char(10);
Dob date;
```

- 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.

Ex 1:- WAP to Simple program

```
BEGIN
   DBMS_OUTPUT.PUT_LINE('Welcome to PL/SQL');
END;
/
```

Note :- Output Displaying parpous by using this command "set serveroutput on"

Ex 2:- WAP to display your name various input methods

```
Declare
        myname varchar2(12):='RAMU';
Begin
        dbms_output.put_line('My Name is : '||myname);
        myname:='Prasanna';
        dbms_output.put_line('My Name is : '||myname);
         myname:='&MyName';
        dbms_output.put_line('My Name is : '||myname);
End;
/
```

Ex 3:- :- WAP to display Addition of two number

```
Declare
        a number(3);
        b number(3);
        r number(3);
Begin
        a:=12;
        b:=15;
        r:=a+b;
        dbms_output.put_line('Addition is : '||r);
End;
/
```

PL/SQL Control structures

Control structures: control structures in PL/SQL are divided into two types

- Select control structure
- Loop control structure

Select control structure:-

Select control structure is used to execute the statements if the condition is true as fales.

Select control structure in PL/SQL is

- If – then
- If –then - else
- If – then - elseif
- Goto
- Case

IF-THEN:

Syntax:-

```
If <condition> then
        Statements;
End if;
```

EX:- WAP to Biggest of two number

```
Declare
        a number(3);
        b number(3);
Begin
        a:=&a;
        b:=&b;
        if  (a>b)  then
                dbms_output.put_line('Biggest number is : '||a);
        else
                dbms_output.put_line('Biggest number is : '||b);
        end if;
End;
/
```

IF-THEN-ELSE:

Syntax:-

```
If <condition> then
        True Statements;
Else
        False Statements;
```

```
End if;
```

Ex:- WAP to Biggest of THREE Number

```
Declare
        a number(3):=&a;
        b number(3):=&b;
        c number(3):=&c;
Begin
        if (a>b and a>c) then
                dbms_output.put_line('Biggest number is : '||a);
        elsif (b>c) then
                        dbms_output.put_line('Biggest number is : '||b);
                else
                        dbms_output.put_line('Biggest number is : '||c);
        end if;
End;
/
```

IF-THEN-ELSIF:

Syntax:-

```
If <condition> then
        True Statements;
ElsIf <condition> then
        True Statements;
Else
        False Statements;
End if;
```

GOTO:

Unconditional control structure. Where label is a label defined in the PL/SQL block. Labels are enclosed in double angle brackets. When a go to statement is evaluated, control immediately passes to the statement identified by the label.

Syntax:-

```
Goto label_name
<<label_name>>:
```

Ex:

```
BEGIN
        For i in 1..5 loop
                dbms_output.put_line('i = ' || i);
          if i = 4 then
            goto exit_loop;
          end if;
        end loop;
```

```
                <<exit_loop>>
                Null;
        END;
        /
```

CASE:-

Like the IF statement, the CASE statement selects one sequence of statements to execute. However, to select the sequence, the CASE statement uses a selector rather than multiple Boolean expressions. A selector is an expression whose value is used to select one of several alternatives.

Syntax:

```
CASE <variable>
        WHEN 'option 1' THEN Statements;
        WHEN 'option 2' THEN Statements;
        WHEN 'option 3' THEN Statements;
        . . .
        WHEN 'option n' THEN Statements;
ELSE
        Statements;
END CASE;
```

Ex:- WAP to print the grade by given request

```
DECLARE
grade CHAR(1);
BEGIN
        DBMS_OUTPUT.PUT_LINE('Enter Grade A/B/C/D/E');
    grade := '&GRADE';
    CASE grade
        WHEN 'A' THEN DBMS_OUTPUT.PUT_LINE('Excellent');
        WHEN 'B' THEN DBMS_OUTPUT.PUT_LINE('Very Good');
        WHEN 'C' THEN DBMS_OUTPUT.PUT_LINE('Good');
        WHEN 'D' THEN DBMS_OUTPUT.PUT_LINE('Fair');
        WHEN 'E' THEN DBMS_OUTPUT.PUT_LINE('Poor');
    ELSE
        DBMS_OUTPUT.PUT_LINE('No such grade');
    END CASE;
END;
/
```

Loop control structure:-

Loop control structure are used to execute statement number of times according to give

condition
Loop control structure are

- While
- For
- Loop

LOOP Statement:-

WHILE LOOP:-

Syntax:-

```
While <condition>
Loop
        Statements;
End loop;
```

Ex:

```
DECLARE
   i number := 1;
BEGIN
   While i <= 5 loop
        dbms_output.put_line('i = ' || i);
        i := i + 1;
    end loop;
END;
```

FOR LOOP:-

Syntax:

```
For <loop_counter_variable> in [reverse] low_bound..high_bound
loop
        Sequence of statements;
End loop;
```

Ex1:- WAP to display 1 to 5 numbers

```
BEGIN
   For i in 1..5 loop
        dbms_output.put_line('i = ' || i);
    end loop;
END;
/
```

Ex2:- WAP to display 1 to 5 numbers in reverse order

```
BEGIN
```

```
    For i in reverse 1..5 loop
        dbms_output.put_line('i = ' || i);
    end loop;
END;
/
```

Loop:-

Syntax:-

```
Loop
        Statements;
        Exit When <condition>
End loop;
        Statements;
```

Ex:- WAP to Display n^{th} number of natural

```
DECLARE
 i number := 1;
  num number:=&num;
BEGIN
   loop
     dbms_output.put_line('i = ' || i);
     i := i + 1;
     exit when i > num;
   end loop;
END;
```

PROCEDURES

PROCEDURES: the block of statements that can be used to perform the pacified task is called a procedure
A procedure should not return any value
The procedure & main program are executed in two separate notepad is a module that performs one or more actions.

Syntax:

```
Create or replace Procedure [schema.]< Procedure_name>(parameter1 [,parameter2 …])
is
        -- [declarations]
Begin
        -- executable statements
End ;
```

After executing the procedure can be called from anther program by using the **syntax**

Syntax:

```
Procedure_name(args)
```

Ex:

Write a Procedure to display your Name by using Procedure.

Procedure body :

```
SQL> ed nameout.sql
create or replace procedure msg (p_name varchar2(20))
IS
BEGIN
        dbms_output.put_line ('Welcome '|| p_name);
END;
/
SQL>@nameout.sql
```

Execution:-

```
SQL> exec msg (' RAMU ');
```

FUNCTIONS

FUNCTIONS: the collection of statement that can be used to perform a specific task is called As function. function is a module that returns a value.

The function and main program are create and executed in two separate files.

Syntax:

```
Create or replace Function [schema.]< function_name>(parameter1 [,parameter2 …])
return <data_type> is
        -- [variable declarations]
Begin
        -- executable statements;
Return var_name;
End ;
```

After executing the function can be called from anther program by using the **syntax**

Syntax:

```
Var_name:=function_name(args)
```

Ex1:- Write a program to display Addition of two numbers by using Function.

Function body:

```
SQL> ed sum.sql
Create or replace function sumout(x number,y number) return number is
        Total number;
Begin
        Total:=x+y;
Return total
End;
/
SQL>@sum.sql
```

Main body:

```
SQL> display.sql
Begin
        dbms_output.put_line(sumout(&a,&b));
End;
SQL>@display.sql
```

Eg2:- Write a function to accept a number and print the factorial of that number?

```
SQL>ed function.sql

create or replace function fac(num number) return number
is
        fact number(4):=1;
begin
        for i in reverse 1..num
        loop
                fact:=fact*i;
        end loop;
        return fact;
end;
/

SQL>@ function.sql;

SQL>select fac (5) from dual;
```

Exceptions

Exceptions: An errors in the PL/SQL program is called exception
The process of handling the error in a program is said to the exception handling

Exception in PL/SQL are divided into two types

1. Pre-defined exception
2. User-defined exception

Pre-defined exception: The exception having pre-defined in the oracle database.
Exception:

1) Zero-Divide
2) No-Data-found
3) Too-many-rows
4) Dup-val-on-index

The pre-defined exception are called automatically and displays the statement in the exception block.

Syntax:-

```
Exception
When <exp> then
Statements;
```

1) **Zero-Divide** :- This exception is raised automatically when we are dividing a number with zero
   Ex:- write a program to divide two numbers
2) **No-Data-found:-** this exception is raise automatically when the user trying to retrieve the record from the table which doesn't exist
3) **Too-many-rows**:- this exception is raise automatically when the user trying to retrieve more than one record from the table without using cursers
4) **Dup-val-on-index**:- this exception is raise automatically when the user insert a duplication record into the table

Ex 1:-

```
Declare
A number:=&a;
B number:=&b;
C number;
Begin
        C:=a/b
        dbms_output.put_line('C is '||C);
exception
        when zero-divide then
        dbms_output.put_line('dont enter zero in denominator');
end;
```

```
/
ex 2:-
    Declare
        Name varchar(20):='&name';
        s number:=&s;
        n number:=&n;
        Begin
            Insert into emp (empno,ename,sal) values(n,name,s);
        exception
            when Dup-val-on-index then
            dbms_output.put_line('record already exist');
        end;
/
```

Others: this exception is raised automatically when any error occurs in the PL/SQL

Pre-defined Exceptions

PL/SQL provides many pre-defined exceptions, which are executed when any database rule is violated by a program. For example, the predefined exception NO_DATA_FOUND is raised when a SELECT INTO statement returns no rows. The following table lists few of the important

Pre-defined exceptions –

Exception	Oracle Error	SQLCODE	Description
ACCESS_INTO_NULL	06530	-6530	It is raised when a null object is automatically assigned a value.
CASE_NOT_FOUND	06592	-6592	It is raised when none of the choices in the WHEN clause of a CASE statement is selected, and there is no ELSE clause.
COLLECTION_IS_NULL	06531	-6531	It is raised when a program attempts to apply collection methods other than EXISTS to an uninitialized nested table or varray, or the program attempts to assign values to the elements of an uninitialized nested table or varray.
DUP_VAL_ON_INDEX	00001	-1	It is raised when duplicate values are attempted to be stored in a column with unique index.

INVALID_CURSOR	01001	-1001	It is raised when attempts are made to make a cursor operation that is not allowed, such as closing an unopened cursor.
INVALID_NUMBER	01722	-1722	It is raised when the conversion of a character string into a number fails because the string does not represent a valid number.
LOGIN_DENIED	01017	-1017	It is raised when a program attempts to log on to the database with an invalid username or password.
NO_DATA_FOUND	01403	+100	It is raised when a SELECT INTO statement returns no rows.
NOT_LOGGED_ON	01012	-1012	It is raised when a database call is issued without being connected to the database.
PROGRAM_ERROR	06501	-6501	It is raised when PL/SQL has an internal problem.
ROWTYPE_MISMATCH	06504	-6504	It is raised when a cursor fetches value in a variable having incompatible data type.
SELF_IS_NULL	30625	-30625	It is raised when a member method is invoked, but the instance of the object type was not initialized.
STORAGE_ERROR	06500	-6500	It is raised when PL/SQL ran out of memory or memory was corrupted.
TOO_MANY_ROWS	01422	-1422	It is raised when a SELECT INTO statement returns more than one row.
VALUE_ERROR	06502	-6502	It is raised when an arithmetic, conversion, truncation, or sizeconstraint error occurs.
ZERO_DIVIDE	01476	1476	It is raised when an attempt is made to divide a number by zero.

User-defined exception: the exceptions create by the user are called user defined exception
When we are creating the user defined excetions we have to follow three steps

1. Declare exception variable
   Syntax:-
   Exp-var exception;

2. Call the exption defined in the exception block
   Syntax:-
   Raise exp-var;

3. The exception can be defined in the exception block by using the
   Syntax:-
   When exp-var the statement;

Ex:- W.A.P to raed the number and display the value of number if the value is positive otherwise call the exception and display the message "the number is negative"

```
declare
nagval exception;
num number;
is
begin
dbms_output.put_line('enter +ve value');
num:=&num
if n>0 then
     dbms_output.put_line('number is positive');
else
     dbms_output.put_line('raising nagval from nested block');
     raise nagval ;
end if;
exception
when nagval then
dbms_output.put_line ('exception raising to main block');
end;
/
```

Cursor

cursor:- cursor is a temporary table that stores the details of given table
cursor are user to retrieve the number of records from the table when we are using the explicit cursor we have follow five steps
step 1:- declare the cursor;
syntax:-

cursor < cursor_name> is select <cal_name> from <tab_name>
[where <condition]
[group by <col_name>]
[order by <col_name>]

Step 2:- open the cursor
Syntax :-

Open <cursor_name>

Step 3:- stares the values of the column in the cursor into the variable names.
Syntax :-

Fetch <cursor_name> into <var_name>;

Step 4:- read all the reareds in the table untill recard innot found
Syntax :-

exit when <cursor_name>%not found ;

step 5:- close cursbor
systax:-

close <`cursor_name>

Ex:- Write a program to display detil of emplayee (empno, ename)

```
Declare
Cursor xyz is select empno,ename from emp;
        N emp.empno%type;
        Name emp.ename%type;
Begin
        open xyz;
        Loop
                Fetch xyz into n,name;
                Dbms_output.put_line ('empno is'||n);
                Dbms_output.put_line ('emp name is'||name);
                Exit when xyz%notfound;
        End loop;
        Close xyz;
End;
/
```

Packages

Packages: the collection of function, procedures, cursors and exception is called as package when we are using the package in PL/SQL. We have to follow three steps.

1) Package specification
2) Package body
3) Main program

The above three parts are created and executed in three different files.

Package specification: In this part we declare the function and procedures whatever we want.

Syntax :-

```
Create or replace package<pack_name> is
        Function statements ;
        Procedure statements;
End;
```

Package body: In this part we write the actual definition of function and procedures which are first declared in package specification part.

Syntax :-

```
Create or replace package body<pack_name> is
        Function statements ;
        Procedure statements;
End;
```

Main program: in the main program to access package.

Syntax:-

<pack-name>.< Procedure_name>(arg);

The funcation in the package can be called by using the

(OR)-

<var_name>:=<pack-name>.<fun-name>(ages)

Ex:-

Create a package to store the following procedure for multiplication table,even-odd, function for factorial and function for palindrome?

Package specification:

```
Create or replace package data
Is
        Procedure mult(a number);
        Procedure even_odd(n number);
        Function fact(n number) return number;
        Function palen(srt varchar2) return varchar2;
End;
/
```

Package body:

```
Create or replace package body data
Is
        Procedure mult(a number)
        Is
                M number;
        Begin
                For i in 1..10
                Loop
                M:=a*i;
                Dbms_output.put_line(a||'x'||i||'='||m);
                End loop;
        End;
        Procedure even_odd(n number)
        Is
        Begin
                If mod(n,2)=0 then
                Dbms_output.put_line(n||' is even number');
                Else
                Dbms_output.put_line(n||' is not even number');
                End if;
        End;
        Function fact(n number) return number
        Is
                F number:=1;
        Begin
                For i in 1..n
                Loop
                F:=f*i; end loop;
                Return f;
        End;
        Function palen(srt varchar2) return varchar2
        Is
                S char;
```

```
                V varchar2(50);
        Begin
                For i in reverse 1..length(srt)
                Loop
                S:=substr(srt,i,1);
                V:=v||s;
                End loop;
                If v=srt then
                Return 'palindrome';
                Else
                Return 'not palindrome';
                End if;
        End;
End;
/
```

Main program:

```
        Begin
                Dbms_output.put_line(data.palen ('RAMUKUMAR'));
        End;
```

Trigger

Trigger :- trigger is an action that is executed automatically when the insert, delete, update is performed on the specified table

Syntax :-

```
Create or replace trigger <trigger_name> before/after insert or update or delete
on <table-name> for each row;
Begin
    Statements;
End;
```

Ex:- **WAP to create trigger that display the message "1 recores is inserntd into emp table " when the user is inserting the recoreds into emp table**

```
SQL> create or replace trigger trig_ins after insert on emp for each row
Begin
Dbms_output.put.line('record is inserted into emp table ');
End;
```

Drop triggers;

```
SQL> drop trigger <trig_name>;
```

Ex:- Write a database trigger store the deleted data of EMP table in EMPDEL table

```
SQL> create table empupdate (empno number,ename varchar(20),job varchar(20),mgr
number,hiredate date,sal number,comm number,deptno number,update_time
varchar(30));

SQL> ed trigger1.sql
create or replace trigger del_tri before insert or delete on emp for each row
begin
insert into empupdate values
(:old.empno,:old.ename,:old.job,:old.mgr,:old.hiredate,:old.sal,:old.comm,:old.deptno,to_
char(sysdate,'hh:mi:ss'));
end;
/
SQL> @trigger1.sql
```

Ex :- **Write a database trigger display the message when the inserting hiredate is greater than system date**

```
Create or replace trigger hiredate_over
After insert on emp
For each row
Begin
   If :new.hiredate > sysdate then
   Raise_application_error(-20009,'invalid hiredate.....');
   End if;
```

```
End;
/
```

Note :

":NEW" – it returns new value
":OLD" – it returns OLD value

Raise – application – error:-

This statement is used to display an error massage layer inserting as deleting as updating the records in the given table

Syntax :-

Raise – application – error(error number,'massage');

This statement takes two argumants

1) Error number
2) Massages

Error number: varies from 20,000 to 20,999
Massages: Massages is any message given by the user