Data Base Management System Lab Manual

Ex. No : 1 Creation of a database and writing SQL queries to retrieve information from the database.

AIM

To create a database and to retrieve the information from the database using SQL queries.

COMMANDS

SQL> create table stud (sname varchar2(30), sid varchar2(10), sage number(2), sarea varchar2(20));
Table created.

SQL> desc stud;

Name	Null?	Type
SNAME		VARCHAR2(30)
SID		VARCHAR2(10)
SAGE		NUMBER(2)
SAREA		VARCHAR2(20)

SQL>alter table stud modify (sage number(10));
Table altered.

SQL> alter table stud add (sdept varchar2(20));
Table altered.

SQL> desc stud;

Name	Null?	Type
SNAME		VARCHAR2(30)
SID		VARCHAR2(10)
SAGE		NUMBER(10)
SAREA		VARCHAR2(20)
SDEPT		VARCHAR2(20)

SQL> alter table stud drop (sdept varchar2(20));
Table altered.

SQL> desc studs;

Name	Null?	Type
SNAME		VARCHAR2(30)
SID		VARCHAR2(10)
SAGE		NUMBER(10)
SAREA		VARCHAR2(20)

SQL> truncate table studs;
Table truncated.

SQL> desc studs;

Name	Null?	Type
SNAME		VARCHAR2(30)
SID		VARCHAR2(10)
SAGE		NUMBER(10)
SAREA		VARCHAR2(20)
SDEPT		VARCHAR2(20)

SQL> drop table studs;

Table dropped.

RESULT

Thus the creation of database and the SQL queries to retrieve information from the database has been implemented and the output was verified.

Ex. No: 2 Performing Insertion, Deletion, Modifying, Altering, Updating and Viewing records based on conditions.

A. **AIM**

To study the various categories of DML commands such as logical operations, aggregate functions, string functions, numeric functions, date functions, conversion functions, group functions and set operations.

DESCRIPTION

- THE ORACLE TABLE – DUAL - Dual is a small oracle table which consists of only one row and one column and contains the value X in that column.
- INSERT - This command is used to insert values into the table.
- SELECT - This command is used to display the contents of the table or those of a particular column.
- RENAME - This command renames the name of the table.
- ARITHMETIC OPERATIONS - Various operations such as addition, multiplication, subtraction and division can be performed using the numbers available in the table.
- DISTINCT - This keyword is used along with select keyword to display unique values from the specified column. It avoids duplicates during display.
- ORDER BY CLAUSE - The order by clause arranges the contents of the table in ascending order (by default) or in descending order (if specified explicitly) according to the specified column.
- CONCATENATION OPERATOR - This combines information from two or more columns in a sentence according to the format specified.

LOGICAL OPERATORS

- AND : The oracle engine will process all rows in a table and displays the result only when all of the conditions specified using the AND operator are specified.
- OR : The oracle engine will process all rows in a table and displays the result only when any of the conditions specified using the OR operators are satisfied.
- NOT : The oracle engine will process all rows in a table and displays the result only when none of the conditions specified using the NOT operator are specified.
- BETWEEN : In order to select data that is within a range of values, the between operator is used. (AND should be included)

PATTERN MATCH

- LIKE PREDICATE : The use of like predicate is that it allows the comparison of one string value with another string value, which is not identical. This is achieved by using wildcard characters which are % and _. The purpose of % is that it matches any string and _ matches any single character.
- IN AND NOT IN PREDICATE : The arithmetic operator = compares a single value to another single value. In case a value needs to be compared to a list of values then the in predicate is used. The not in predicate is the opposite of the in predicate. This will select all the rows whose values do not match all of the values in the list.

NUMERIC FUNCTIONS

- ABS: It returns the absolute value of ‘n’.
- POWER: It returns m raised to nth power. n must be an integer else an error is returned.
- ROUND: It returns n rounded to m places right of the decimal point. If m is omitted, n is rounded to zero places. m must be an integer.
- SQRT: It returns square root of n. n should be greater than zero.

STRING FUNCTIONS

- LOWER: It returns char with letters in lower case.
- INITCAP: It returns char with the first letter in upper case.
- UPPER: It returns char with all letters forced to upper case.
- SUBSTR: It returns a portion of char beginning at character m, exceeding up to n characters. If n is omitted result is written up to the end character. The 1stposition of char is one.
- LENGTH: It returns the length of char
- LTRIM: It removes characters from the left of char with initial characters removed up to the 1st character not in set.
- RTRIM: It returns char with final characters removed after the last character not in the set. Set is optional. It defaults to spaces.
- LPAD: It returns char1, left padded to length n with the sequence of characters in char2. char2 defaults to blanks.
- RPAD: It returns char1, right padded to length n with the characters in char2, replicated as many times as necessary. If char2 is omitted, it is padded with blanks.

AGGREGATE FUNCTIONS

- AVG (N): It returns average value of n ignoring null values.
- MIN (EXPR): It returns minimum value of the expression.
- COUNT (EXPR): It returns the number of rows where expression is not null.
- COUNT (*): It returns the number of rows in the table including the duplicates and those with null values.
- MAX (EXPR): It returns maximum value of the expression.
- SUM(N): It returns sum of values of n.

CONVERSION FUCTIONS

- TO_NUMBER(CHAR): It converts the char value containing a number to a value of number data type.
- TO_CHAR(N,FMT): It converts a value of number data type to a value of char data type, using the optional format string. It accepts a number n and a numeric format fmt in which the number has to appear. If fmt is omitted, n is converted to a char value exactly long enough to hold significant digits.
- TO_CHAR(DATE, FMT): It converts a value of data type to char value. It accepts a date as well as the format in which the date has to appear. Fmt must be a date format. If fmt is omitted, date is the default date format.

DATE FUNCTIONS

- SYSDATE : The sysdate is a pseudo column that contains the current date and time. It requires no arguments when selected from the table dual and returns the current date.

- ADD_MONTHS(D,N): It returns date after adding the number of months specified with the function.
- LAST_DAY(D): It returns the last date of the month specified with the function
- MONTHS_BETWEEN(D1,D2): It returns number of months between D1 and D2.
- NEXT_DAY(DATE, CHAR): It returns the date of the first week day named by char . char must be a day of the week.

GROUP BY CLAUSE

The group by clause is another section of the select statement. This optional class tells oracle to group rows based on distinct values that exists for specified columns.

HAVING CLAUSE

The having clause can be used in conjunction with the group by clause. Having imposes a condition on the group by clause, which further filters the groups created by the group by clause.

SET OPERATIONS

- UNION CLAUSE: Multiple queries can be put together and their output combined using the union clause. The union clause merges the output of two or more queries into a single set of rows and columns.
- INTERSECT CLAUSE: Multiple queries can be put together and their output can be combined using the intersect clause. The intersect clause outputs only rows produced by both the queries intersected. The output in an intersect clause will include only those rows that are retrieved by both the queries.

COMMANDS

CREATION OF TABLE

```
SQL>create table stud (sname varchar2(30), sid varchar2(10), sage number(10), sarea varchar2(20), sdept varchar2(20));
Table created.
```

INSERTION OF VALUES INTO THE TABLE

```
SQL> insert into stud values ('ashwin',101,19,'anna nagar','aeronautical');
1 row created.
SQL> insert into stud values ('bhavesh',102,18,'nungambakkam','marine');
1 row created.
SQL> insert into stud values ('pruthvik',103,20,'anna nagar','aerospace');
1 row created.
SQL> insert into stud values ('charith',104,20,'kilpauk','mechanical');
1 row created.
SQL> select * from stud;
```

SNAME	SID	SAGE	SAREA	SDEPT
ashwin	101	19	anna nagar	aeronautical
bhavesh	102	18	nungambakkam	marine
pruthvik	103	20	anna nagar	aerospace
charith	104	20	kilpauk	mechanical

RENAMING THE TABLE 'STUD'

```
SQL> rename stud to studs;
```

Table renamed.

ARITHMETIC OPERATION

```
SQL> select sname, sid+100 "stid" from studs;
SNAME                          stid
------------------------------ ---------
ashwin                          201
bhavesh                         202
pruthvik                        203
charith                         204
```

CONCATENATION OPERATOR

```
SQL> select sname || ' is a ' || sdept || ' engineer. ' AS  "PROFESSION" from studs;
PROFESSION
----------------------------------------------------------------------
ashwin is a aeronautical engineer.
bhavesh is a marine engineer.
pruthvik is a aerospace engineer.
charith is a mechanical engineer.
```

DISPLAY ONLY DISTINCT VALUES

```
SQL> select distinct sarea from studs;
SAREA
--------------------
anna nagar
kilpauk
nungambakkam
```

USING THE WHERE CLAUSE

```
SQL> select sname,sage from studs where sage<=19;
SNAME                          SAGE
------------------------------ ---------
ashwin                          19
bhavesh                         18
```

BETWEEN OPERATOR

```
SQL> select sname,sarea, sid from studs where sid between 102 and 104;
SNAME                          SAREA                SID
------------------------------ -------------------- ----------
bhavesh                        nungambakkam          102
pruthvik                       anna nagar            103
charith                        kilpauk               104
```

IN PREDICATE

```
SQL> select sname,sarea , sid from studs where sid in(102,104);
SNAME                          SAREA                SID
------------------------------ -------------------- ----------
bhavesh                        nungambakkam          102
charith                        kilpauk               104
```

PATTERN MATCHING

```
SQL> select sname, sarea from studs where sarea like '%g%';

SNAME                          SAREA
------------------------------ --------------------
ashwin                         anna nagar
bhavesh                        nungambakkam
pruthvik                       anna nagar
```

LOGICAL AND OPERATOR

```
SQL> select sname ,sid from studs where sid>102 and sarea='anna nagar';
SNAME                          SID
------------------------------ ----------
pruthvik                       103
```

LOGICAL OR OPERATOR

```
SQL> select sname ,sid from studs where sid>102 or sarea='anna nagar';
SNAME                          SID
------------------------------ ----------
ashwin                         101
pruthvik                       103
charith                        104
```

NOT IN PREDICATE

```
SQL> select sname, sid from studs where sid not in(102,104);
SNAME                          SID
------------------------------ ----------
ashwin                         101
pruthvik                       103
```

UPDATING THE TABLE

```
SQL> alter table studs add ( spocket varchar2(20) );
Table altered.

SQL> update studs set spocket=750 where sid=101;
1 row updated.
SQL> update studs set spocket=500 where sid=102;
1 row updated.
SQL> update studs set spocket=250 where sid=103;
1 row updated.
SQL> update studs set spocket=100 where sid=104;
1 row updated.
SQL> select * from studs;
SNAME                          SID        SAGE  SAREA              SDEPT
------------------------------ ---------- --------- -------------------- --------------------
SPOCKET
```

```
--------------------
ashwin          101        19    anna nagar          aeronautical
750
bhavesh         102        18    nungambakkam        marine
500
pruthvik        103        20    anna nagar          aerospace
250
charith         104        20    kilpauk             mechanical
100
```

AGGREGATE FUNCTIONS

```
SQL> select avg( spocket ) result from studs;
  RESULT
---------
   400
SQL>  select min(spocket) result  from studs;
RESULT
--------------------
100
SQL> select count(spocket) result from studs;
  RESULT
---------
       4
SQL> select count(*) result from studs;
  RESULT
---------
       4
SQL> select count(spocket) result from studs where sarea='anna nagar';
  RESULT
---------
       2
SQL> select max(spocket) result from studs;
RESULT
--------------------
750
SQL> select sum(spocket) result from studs;
  RESULT
---------
    1600
```

NUMERIC FUNCTIONS

```
SQL> select abs(-20) result from dual;
  RESULT
---------
      20
SQL> select power (2,10) result from dual;
```

```
  RESULT
---------
    1024
SQL> select round(15.359,2) result from dual;
  RESULT
---------
   15.36
SQL> select sqrt (36) result from dual;
  RESULT
---------
       6
```

STRING FUNCTIONS

```
SQL> select lower('ORACLE') result from dual;
RESULT
------
oracle
SQL> select upper('oracle') result from dual;
RESULT
------
ORACLE
SQL> select initcap('Oracle') result from dual;
RESULT
------
Oracle
SQL> select substr('oracle' ,2 ,5) result from dual;
RESULT
-----
racle
SQL> select lpad('oracle',10,'#') result from dual;
RESULT
----------
####oracle
SQL> select rpad ('oracle',10,'^') result from dual;
RESULT
----------
oracle^^^^
```

CONVERSION FUNCTIONS

```
SQL> update studs set sage=to_number(substr(118,2,3));
4 rows updated.
SQL> select * from studs;
SNAME              SID       SAGE      SAREA              SDEPT       SPOCKET
------------------- ---------- --------- -------------------- --------------------
ashwin              101          18        anna nagar         aeronautical   750
bhavesh             102          18        nungambakkam       marine          500
```

pruthvik 103 18 anna nagar aerospace 250
charith 104 18 kilpauk mechanical 100

```
SQL> select to_char( 17145, '099,999')  result from dual;
RESULT
--------
 017,145
SQL> select to_char(sysdate,'dd-mon-yyyy')  result from dual;
RESULT
-----------
16-jul-2008
```

DATE FUNCTIONS

```
SQL> select sysdate from dual;
SYSDATE
---------
16-JUL-08
SQL> select  sysdate,add_months(sysdate,4) result from dual;
SYSDATE   RESULT
---------          ---------
16-JUL-08 16-NOV-08

SQL> select sysdate, last_day(sysdate) result from dual;
SYSDATE   RESULT
---------          ---------
16-JUL-08 31-JUL-08

SQL> select  sysdate, next_day(sysdate,'sunday') result from dual;
SYSDATE   RESULT
---------        ---------
16-JUL-08   20-JUL-08

SQL> select months_between('09-aug-91','11-mar-90') result from dual;
  RESULT
    ---------
 16.935484
```

GROUP BY CLAUSE

```
SQL> select sarea, sum(spocket) result from studs group by sarea;
SAREA                    RESULT
--------------------          ------------
anna nagar                  1000
nungambakkam                 500
kilpauk                      100
```

HAVING CLAUSE

```
SQL> select sarea, sum(spocket) result from studs group by sarea having spocket<600;
```

```
SAREA                RESULT
-------------------  ------------
nungambakkam          500
kilpauk               100
```

DELETION

```
SQL> delete from studs where sid=101;
1 row deleted.
SQL> select * from studs;
SNAME                SID        SAGE       SAREA              SDEPT              SPOCKET
---------------------------- ---------- --------- ------------------- ------------------- ------------------
bhavesh              102          18       nungambakkam       marine             500
pruthvik             103          20        anna nagar         aerospace          250
charith              104          20       kilpauk             mechanical         100
```

CREATING TABLES FOR DOING SET OPERATIONS

TO CREATE PRODUCT TABLE

```
SQL> create table product(prodname varchar2(30), prodno varchar2(10));
Table created.
SQL> insert into product values('table',10001);
1 row created.
SQL> insert into product values('chair',10010);
1 row created.
SQL> insert into product values('desk',10110);
1 row created.
SQL> insert into product values('cot',11110);
1 row created.
SQL> insert into product values('sofa',10010);
1 row created.
SQL>
SQL> insert into product values('tvstand',11010);
1 row created.
SQL> select * from product;
PRODNAME                PRODNO
------------------------------ ----------
table                   10001
chair                   10010
desk                    10110
cot                     11110
sofa                    10010
tvstand                 11010
```

TO CREATE SALE TABLE

```
SQL> create table sale(prodname varchar2(30),orderno number(10),prodno varchar2(10));
Table created.
SQL>  insert into sale values('table',801,10001);
```

1 row created.
SQL> insert into sale values('chair',805,10010);
1 row created.
SQL> insert into sale values('desk',809,10110);
1 row created.
SQL> insert into sale values('cot',813,11110);
1 row created.
SQL> insert into sale values('sofa',817,10010);
1 row created.
SQL> select * from sale;

PRODNAME	ORDERNO	PRODNO
table	801	10001
chair	805	10010
desk	809	10110
cot	813	11110
sofa	817	10010

SET OPERATIONS

SQL> select prodname from product where prodno=10010 union select prodname from sale where prodno=10010;

PRODNAME
---
chair
sofa

SQL> select prodname from product where prodno=11110 intersect select prodname from sale where prodno=11110;

PRODNAME
---
cot

RESULT

The DML commands were executed and the output was verified.

B. **AIM**

To study the nested queries using DML commands.

TO CREATE SSTUD1 TABLE

```
SQL> create table sstud1 ( sname varchar2(20) , place varchar2(20));
Table created.
SQL> insert into sstud1 values ( 'prajan','chennai');
1 row created.
SQL> insert into sstud1 values ( 'anand','chennai');
1 row created.
SQL> insert into sstud1 values ( 'kumar','chennai');
1 row created.
SQL> insert into sstud1 values ( 'ravi','chennai');
1 row created.
SQL> select * from sstud1;
SNAME               PLACE
------------------- --------------------
prajan              chennai
anand               chennai
kumar               chennai
ravi                Chennai
```

TO CREATE SSTUD2 TABLE

```
SQL> create table sstud2 ( sname varchar2(20), dept varchar2(10), marks number(10));
Table created.
SQL> insert into sstud2 values ('prajan','cse',700);
1 row created.
SQL> insert into sstud2 values ('anand','it',650);
1 row created.
SQL> insert into sstud2 values ('vasu','cse',680);
1 row created.
SQL> insert into sstud2 values ('ravi','it',600);
1 row created.
SQL> select * from sstud2;
SNAME               DEPT          MARKS
------------------- ---------- ---------
prajan               cse           700
anand                it            650
vasu                 cse           680
ravi                 it            600
```

NESTED QUERIES

```
SQL> select sname from sstud1 where sstud1.sname in ( select sstud2.sname from  sstud2 );
SNAME
--------------------
anand
prajan
```

ravi
SQL> select sname from sstud1 where sstud1.sname not in (select sstud2.sname from sstud2);
SNAME
---
kumar
SQL> select sname from sstud2 where marks > some(select marks from sstud2 where dept='cse');
SNAME
---
prajan
SQL> select sname from sstud2 where marks >= some (select marks from sstud2 where dept='cse');
SNAME
---
prajan
vasu
SQL> select sname from sstud2 where marks > any (select marks from sstud2 where dept='cse');
SNAME
---
prajan
SQL> select sname from sstud2 where marks >= any (select marks from sstud2 where dept='cse');
SNAME
---
prajan
vasu
SQL> select sname from sstud2 where marks > all (select marks from sstud2 where dept='cse');
no rows selected
SQL> select sname from sstud2 where marks < all (select marks from sstud2 where dept='cse');
SNAME
---
anand
ravi
SQL> select sname from sstud1 where exists (select sstud2.sname from sstud2 where sstud1.sname=sstud2.sname);
SNAME
---
prajan
anand
ravi
SQL> select sname from sstud1 where not exists (select sstud2.sname from sstud2 where sstud1.sname=sstud2.sname);
SNAME
---
kumar

RESULT

The Nested Queries using DML commands were executed and the output was verified.

Ex. No 3: Creation of Views, Synonyms, Sequence, Indexes, Save point.

AIM:

To create views, synonyms, sequences, indexes and save points using DDL, DML and DCL statements

DESCRIPTION:

Views

- A database view is a *logical* or *virtual table* based on a query. It is useful to think of a *view* as a stored query. Views are queried just like tables.
- A DBA or view owner can drop a view with the DROP VIEW command.

TYPES OF VIEWS

- Updatable views – Allow data manipulation
- Read only views – Do not allow data manipulation

TO CREATE THE TABLE 'FVIEWS'

```
SQL> create table fviews( name varchar2(20),no number(5), sal number(5), dno number(5));
Table created.
SQL> insert into fviews values('xxx',1,19000,11);
1 row created.
SQL> insert into fviews values('aaa',2,19000,12);
1 row created.
SQL> insert into fviews values('yyy',3,40000,13);
1 row created.
SQL> select * from fviews;
NAME                 NO       SAL       DNO
-------------------- --------- --------- ---------
xxx                   1     19000        11
aaa                   2     19000        12
yyy                   3     40000        13
```

TO CREATE THE TABLE 'DVIEWS'

```
SQL> create table dviews( dno number(5), dname varchar2(20));
Table created.
SQL> insert into dviews values(11,'x');
1 row created.
SQL> insert into dviews values(12,'y');
1 row created.
SQL> select * from dviews;
    DNO DNAME
--------- --------------------
     11 x
     12 y
```

CREATING THE VIEW 'SVIEW' ON 'FVIEWS' TABLE

```
SQL> create view sview as select name,no,sal,dno from fviews where dno=11;
View created.
```

SQL> select * from sview;

NAME	NO	SAL	DNO
xxx	1	19000	11

Updates made on the view are reflected only on the table when the structure of the table and the view are not similar -- proof

SQL> insert into sview values ('zzz',4,20000,14);

1 row created.

SQL> select * from sview;

NAME	NO	SAL	DNO
xxx	1	19000	11

SQL> select * from fviews;

NAME	NO	SAL	DNO
xxx	1	19000	11
aaa	2	19000	12
yyy	3	40000	13
zzz	4	20000	14

Updates made on the view are reflected on both the view and the table when the structure of the table and the view are similar – proof

CREATING A VIEW 'IVIEW' FOR THE TABLE 'FVIEWS'

SQL> create view iview as select * from fviews;

View created.

SQL> select * from iview;

NAME	NO	SAL	DNO
xxx	1	19000	11
aaa	2	19000	12
yyy	3	40000	13
zzz	4	20000	14

PERFORMING UPDATE OPERATION

SQL> insert into iview values ('bbb',5,30000,15);

1 row created.

SQL> select * from iview;

NAME	NO	SAL	DNO
xxx	1	19000	11
bbb	5	30000	15

SQL> select * from fviews;

NAME	NO	SAL	DNO
xxx	1	19000	11
aaa	2	19000	12

```
yyy                  3    40000       13
zzz                  4    20000       14
bbb                  5    30000       15
```

CREATE A NEW VIEW 'SSVIEW' AND DROP THE VIEW

```
SQL> create view ssview( cusname,id) as select name, no from fviews where dno=12;
View created.
SQL> select * from ssview;
CUSNAME                 ID
-------------------- ---------
aaa                  2
SQL> drop view ssview;
View dropped.
```

TO CREATE A VIEW 'COMBO' USING BOTH THE TABLES 'FVIEWS' AND 'DVIEWS'

```
SQL> create view combo as select name,no,sal,dviews.dno,dname from fviews,dviews where
fviews.dno=dviews.dno;
View created.
SQL> select * from combo;
NAME                 NO       SAL       DNO DNAME
-------------------- --------- --------- --------- --------------------
xxx                  1    19000       11 x
aaa                  2    19000       12 y
```

TO PERFORM MANIPULATIONS ON THIS VIEW

```
SQL> insert into combo values('ccc',12,1000,13,'x');
insert into combo values('ccc',12,1000,13,'x')
*
ERROR at line 1:
ORA-01779: cannot modify a column which maps to a non key-preserved table
```

This shows that when a view is created from two different tables no manipulations can be performed using that view and the above error is displayed.

Synonyms

- A *synonym* is an *alias*, that is, a form of shorthand used to simplify the task of referencing a database object.
- There are two categories of synonyms, *public* and *private.*
- A public synonym can be accessed by any system user.
- The individual creating a public synonym does not own the synonym – rather, it will belong to the PUBLIC user group that exists within Oracle.
- Private synonyms, on the other hand, belong to the system user that creates them and reside in that user's schema.
- A system user can grant the privilege to use private synonyms that they own to other system users.
- In order to create synonyms, we will need to have the CREATE SYNONYM privilege.
- This privilege will be granted to us by the DBA.

- We must have the CREATE PUBLIC SYNONYM privilege in order to create public synonyms.
- If we own a synonym, we have the right to drop (delete) the synonym. The DROP SYNONYM command is quite simple.
- DROP SYNONYM synonym_name;
- In order to drop a public synonym we must include the PUBLIC keyword in the DROP SYNONYM command.
- In order to drop a public synonym, we must have the DROP PUBLIC SYNONYM privilege.
- DROP PUBLIC SYNONYM synonym_name;

Examples:

```
SQL> select * from class;

NAME            ID
---------- ----------
anu              1
brindha          2
chinthiya        3
divya            4
ezhil            5
fairoz           7
hema             9

7 rows selected.
```

Create synonym:

```
SQL> create synonym c1 for class;

Synonym created.
SQL> insert into c1 values('kalai',20);

1 row created.

SQL> select * from class;

NAME            ID
---------- ----------
anu              1
brindha          2
```

```
chinthiya          3
divya              4
ezhil              5
fairoz             7
hema               9
kalai             20

8 rows selected.

SQL> select * from c1;

NAME                ID
---------- ----------
anu                 1
brindha             2
chinthiya           3
divya               4
ezhil               5
fairoz              7
hema                9
kalai              20

8 rows selected.
SQL> insert into class values('Manu',21);

1 row created.

SQL> select * from c1;

NAME                ID
---------- ----------
anu                 1
brindha             2
chinthiya           3
divya               4
ezhil               5
fairoz              7
hema                9
kalai              20
Manu               21
```

9 rows selected.

Drop Synonym:

```
SQL> drop synonym c1;

Synonym dropped.

SQL> select * from c1;
select * from c1
              *
ERROR at line 1:
ORA-00942: table or view does not exist
```

Sequences

- Oracle provides the capability to generate sequences of unique numbers, and they are called **sequences.**
- Just like tables, views, indexes, and synonyms, a sequence is a type of database object.
- Sequences are used to generate unique, sequential integer values that are used as primary key values in database tables.
- The sequence of numbers can be generated in either ascending or descending order.

Creation of table:

```
SQL> create table class(name varchar(10),id number(10));
Table created.
```

Insert values into table:

```
SQL> insert into class values('&name',&id);
Enter value for name: anu
Enter value for id: 1
old   1: insert into class values('&name',&id)
new   1: insert into class values('anu',1)

1 row created.

SQL> /
Enter value for name: brindha
Enter value for id: 02
old   1: insert into class values('&name',&id)
new   1: insert into class values('brindha',02)

1 row created.
```

```
SQL> /
Enter value for name: chinthiya
Enter value for id: 03
old   1: insert into class values('&name',&id)
new   1: insert into class values('chinthiya',03)

1 row created.

SQL> select * from class;

NAME              ID
----------      ----------
anu                  1
brindha              2
chinthiya            3
```

Create Sequence:

```
SQL> create sequence s_1
  2  start with 4
  3  increment by 1
  4  maxvalue 100
  5  cycle;

Sequence created.

SQL> insert into class values('divya',s_1.nextval);

1 row created.

SQL> select * from class;

NAME              ID
---------- ----------
anu                1
brindha            2
chinthiya          3
divya              4
```

Alter Sequence:

```
SQL> alter sequence s_1
  2  increment by 2;

Sequence altered.

SQL> insert into class values('fairoz',s_1.nextval);

1 row created.

SQL> select * from class;

NAME            ID
---------- ----------
anu              1
brindha          2
chinthiya        3
divya            4
ezhil            5
fairoz           7
```

Drop Sequence:

```
SQL> drop sequence s_1;

Sequence dropped.
```

Indexes

- An index can be created in a table to find data more quickly and efficiently.
- The users cannot see the indexes; they are just used to speed up searches/queries.
- Updating a table with indexes takes more time than updating a table without; because the indexes also need an update. So we should only create indexes on columns (and tables) that will be frequently searched against.

Syntax:

Create Index:

CREATE INDEX index_name ON table_name (column_name)

```
SQL> create table splr(sname varchar(10),sid number(10),scity varchar(10));

Table created.
```

```
SQL> insert into splr values('hcl',01,'chennai');

1 row created.

SQL> insert into splr values('dell',04,'madurai');

1 row created.

SQL> insert into splr values('HP',02,'kovai');

1 row created.

SQL> insert into splr values('Lenovo',03,'trichy');

1 row created.

SQL> select * from splr;

SNAME           SID SCITY
---------- ---------- ----------
hcl              1 chennai
dell             4 madurai
HP               2 kovai
Lenovo           3 trichy

SQL> create index sp1 on splr(sid);

Index created.

SQL> create index sp2 on splr(sid,scity);

Index created.
```

Drop Index:

```
SQL> drop index sp1;

Index dropped.

SQL> drop index sp2;

Index dropped.
```

DCL statements

DESCRIPTION

The DCL language is used for controlling the access to the table and hence securing the database. DCL is used to provide certain privileges to a particular user. Privileges are rights to be allocated. The privilege commands are namely,

- Grant
- Revoke
- Commit
- Savepoint
- Rollback

GRANT COMMAND: It is used to create users and grant access to the database. It requires database administrator (DBA) privilege, except that a user can change their password. A user can grant access to their database objects to other users.

REVOKE COMMAND: Using this command , the DBA can revoke the granted database privileges from the user.

COMMIT : It is used to permanently save any transaction into database.

SAVEPOINT: It is used to temporarily save a transaction so that you can rollback to that point whenever necessary

ROLLBACK: It restores the database to last commited state. It is also use with savepoint command to jump to a savepoint in a transaction

SYNTAX

GRANT COMMAND

Grant < database_priv [database_priv…..] > to <user_name> identified by <password> [,<password…..];

Grant <object_priv> | All on <object> to <user | public> [With Grant Option];

REVOKE COMMAND

Revoke <database_priv> from <user [, user] >;

Revoke <object_priv> on <object> from < user | public >;

<database_priv> -- Specifies the system level priveleges to be granted to the users or roles. This includes create / alter / delete any object of the system.

<object_priv> -- Specifies the actions such as alter / delete / insert / references / execute / select / update for tables.

<all> -- Indicates all the priveleges.

[With Grant Option] – Allows the recipient user to give further grants on the objects.

The priveleges can be granted to different users by specifying their names or to all users by using the “Public” option.

COMMIT:

Commit;

SAVEPOINT:

Savepoint savapoint_name;

ROLLBACK:

Rollback to savepoint_name;

EXAMPLES

Consider the following tables namely “DEPARTMENTS” and “EMPLOYEES”
Their schemas are as follows ,
Departments (dept _no , dept_ name , dept_location);
Employees (emp_id , emp_name , emp_salary);
SQL> Grant all on employees to abcde;
Grant succeeded.
SQL> Grant select , update , insert on departments to abcde with grant option;
Grant succeeded.
SQL> Revoke all on employees from abcde;
Revoke succeeded.
SQL> Revoke select , update , insert on departments from abcde;
Revoke succeeded.

COMMIT, ROLLBACK and SAVEPOINT:

SQL> select * from class;

NAME ID
---------- ----------
anu 1
brindha 2
chinthiya 3
divya 4
ezhil 5
fairoz 7

SQL> insert into class values('gayathri',9);

1 row created.

SQL> commit;

Commit complete.

SQL> update class set name='hema' where id='9';

1 row updated.

SQL> savepoint A;

Savepoint created.

SQL> insert into class values('indu',11);

1 row created.

SQL> savepoint B;

Savepoint created.

```
SQL> insert into class values('janani',13);

1 row created.

SQL> select * from class;

NAME                 ID
---------- ----------
anu                   1
brindha               2
chinthiya             3
divya                 4
ezhil                 5
fairoz                7
hema                  9
indu                 11
janani               13

9 rows selected.

SQL> rollback to B;

Rollback complete.

SQL> select * from class;

NAME                 ID
---------- ----------
anu                   1
brindha               2
chinthiya             3
divya                 4
ezhil                 5
fairoz                7
hema                  9
indu                 11

8 rows selected.

SQL> rollback to A;

Rollback complete.

SQL> select * from class;
```

NAME	ID
anu	1
brindha	2
chinthiya	3
divya	4
ezhil	5
fairoz	7
hema	9

RESULT:

Thus the Views, Synonyms, and Sequences, indexes and save points has been executed using DDL, DML and DCL statements.

Ex.No :4 Creating an Employee Database to set various Constraints.

AIM

To study the various constraints available in the SQL query language.

DOMAIN INTEGRITY CONSTRAINTS

NOT NULL CONSTRAINT

```
SQL> create table empl (ename varchar2(30) not null, eid varchar2(20) not null);
Table created.
SQL> insert into empl values ('abcde',11);
1 row created.
SQL> insert into empl values ('fghij',12);
1 row created.
SQL> insert into empl values ('klmno',null);
insert into empl values ('klmno',null)
*
ERROR at line 1:
ORA-01400: cannot insert NULL into ("ITA"."EMPL"."EID")
SQL> select * from empl;
ENAME                          EID
------------------------------ --------------------
abcde                          11
fghij                          12
```

CHECK AS A COLUMN CONSTRAINT

```
SQL> create table depts ( dname varchar2(30) not null, did number(20) not null check (did<10000));
Table created.
SQL> insert into depts values ('sales ',9876);
1 row created.
SQL> insert into depts values ('marketing',5432);
1 row created.
SQL> insert into depts values ('accounts',789645);
insert into depts values ('accounts',789645)
*
ERROR at line 1:
ORA-02290: check constraint (ITA.SYS_C003179) violated
SQL> select * from depts;
DNAME                            DID
------------------------------ ---------
sales                           9876
marketing                       5432
```

CHECK AS A TABLE CONSTRAINT

```
SQL> create table airports (aname varchar2(30) not null , aid number(20) not null, acity varchar2(30)
check( acity in ('chennai','hyderabad','bangalore')));
Table created.
SQL> insert into airports values( 'abcde', 100,'chennai');
1 row created.
SQL> insert into airports values( 'fghij', 101,'hyderabad');
```

```
1 row created.
SQL> insert into airports values( 'klmno', 102,'bangalore');
1 row created.
SQL> insert into airports values( 'pqrst', 103,'mumbai');
insert into airports values( 'pqrst', 103,'mumbai')
*
ERROR at line 1:
ORA-02290: check constraint (ITA.SYS_C003187) violated
SQL> select * from airports;
ANAME                          AID ACITY
------------------------------ --------- ------------------------------
abcde                           100      chennai
fghij                           101      hyderabad
klmno                           102      bangalore
```

ENTITY INTEGRITY CONSTRAINTS

UNIQUE AS A COLUMN CONSTRAINT

```
SQL> create table book (bname varchar2(30) not null, bid number(20) not null unique);
Table created.
SQL> insert into book values ('fairy tales',1000);
1 row created.
SQL> insert into book values ('bedtime stories',1001);
1 row created.
SQL> insert into book values ('comics',1001);
insert into book values ('comics',1001)
*
ERROR at line 1:
ORA-00001: unique constraint (ITA.SYS_C003130) violated
SQL> select * from book;
BNAME                           BID
------------------------------ ---------
fairy tales                     1000
bedtime stories                 1001
```

UNIQUE AS A TABLE CONSTRAINT

```
SQL> create table orders( oname varchar2(30) not null , oid number(20) not null , unique(oname,oid));
Table created.
SQL> insert into orders values ('chair', 2005);
1 row created.
SQL> insert into orders values ('table',2006);
1 row created.
SQL> insert into orders values ('chair',2007);
1 row created.
SQL> insert into orders values ('chair', 2005);
insert into orders values ('chair', 2005)
```

*
ERROR at line 1:
ORA-00001: unique constraint (ITA.SYS_C003152) violated
SQL> select * from orders;
ONAME OID
------------------------------ ---------
chair 2005
table 2006
chair 2007

PRIMARY KEY AS A COLUMN CONSTRAINT

SQL> create table custo (cname varchar2(30) not null , cid number(20) not null primary key);
Table created.
SQL> insert into custo values ('jones', 506);
1 row created.
SQL> insert into custo values ('hayden',508);
1 row created.
SQL> insert into custo values ('ricky',506);
insert into custo values ('ricky',506)
*
ERROR at line 1:
ORA-00001: unique constraint (ITA.SYS_C003165) violated
SQL> select * from custo;
CNAME CID
------------------------------ ---------
jones 506
hayden 508

PRIMARY KEY AS A TABLE CONSTRAINT

SQL> create table branches(bname varchar2(30) not null , bid number(20) not null , primary key(bnam e,bid));
Table created.
SQL> insert into branches values ('anna nagar', 1005);
1 row created.
SQL> insert into branches values ('adyar',1006);
1 row created.
SQL> insert into branches values ('anna nagar',1007);
1 row created.
SQL> insert into branches values ('anna nagar', 1005);
insert into branches values ('anna nagar', 1005)
*
ERROR at line 1:
ORA-00001: unique constraint (ITA.SYS_C003173) violated
SQL> select * from branches;
BNAME BID

------------------------------ ---------
anna nagar 1005
adyar 1006
anna nagar 1007

REFERENTIAL INTEGRITY CONSTRAINTS

TO CREATE 'DEPTS' TABLE

SQL> create table depts(city varchar2(20), dno number(5) primary key);
Table created.
SQL> insert into depts values('chennai', 11);
1 row created.
SQL> insert into depts values('hyderabad', 22);
1 row created.

TO CREATE 'SEMP' TABLE

SQL> create table semp(ename varchar2(20), dno number(5) references depts(dno));
Table created.
SQL> insert into semp values('x', 11);
1 row created.
SQL> insert into semp values('y', 22);
1 row created.
SQL> select * from semp;

ENAME DNO
-------------------- ---------
x 11
y 22

ALTER TABLE

SQL> alter table semp add(eddress varchar2(20));
Table altered.
SQL> update semp set eddress='10 gandhi road' where dno=11;
1 row updated.
SQL> update semp set eddress='12 m.g. road' where dno=22;
1 row updated.
SQL> select * from semp;

ENAME DNO EDDRESS
-------------------- --------- --------------------
x 11 10 gandhi road
y 22 12 m.g. road

SQL> select city, ename from depts, s2emp where depts.dno = s2emp.dno;

CITY ENAME
-------------------- --------------------
chennai x
hyderabad y

RESULT

The various constraints were implemented and the tables were created using the respective constraints.

Ex. No: 5 Creating relationship between the databases.

AIM

To create databases and implement the relationship between databases using joinoperation.

DESCRIPTION:

JOIN OPERATIONS

- INNER JOIN/ NATURAL JOIN/ JOIN: It is a binary operation that allows us to combine certain selections and a Cartesian product into one operation.
- OUTER JOIN: It is an extension of join operation to deal with missing information.
  - Left Outer Join: It takes tuples in the left relation that did not match with any tuple in the right relation, pads the tuples with null values for all other attributes from the right relation and adds them to the result of the natural join.
  - Right Outer Join: It takes tuples in the right relation that did not match with any tuple in the left relation, pads the tuples with null values for all other attributes from the left relation and adds them to the result of the natural join.
  - Full Outer Join: It combines tuples from both the left and the right relation and pads the tuples with null values for the missing attributes and hem to the result of the natural join.

CREATING TABLES FOR DOING JOIN AND NESTED QUERY OPERATIONS

Creating Dept table:

```
SQL> create table dept(dno number(10),dname varchar(10),loc varchar(10));
Table created.
SQL> insert into dept values(10,'inventory','hyd');
1 row created.
SQL> insert into dept values(20,'finance','bglr');
1 row created.
SQL> insert into dept values(30,'HR','mumbai');
1 row created.
SQL> select * from dept;
      DNO DNAME      LOC
---------- ---------- ----------
        10 inventory  hyd
        20 finance    bglr
        30 HR         mumbai
```

Creating emp2 table:

```
SQL> create table emp2(eno number(10),ename varchar(10),job varchar(10),M
er(10),dno number(10));
Table created.
SQL> insert into emp2 values(111,'saketh','analyst',444,10);
1 row created.
SQL> insert into emp2 values(222,'sandeep','clerk',333,20);
1 row created.
SQL> insert into emp2 values(333,'jagan','manager',111,10);
1 row created.
SQL> insert into emp2 values(444,'madhu','engineer',222,40);
1 row created.
SQL> select * from emp2;
      ENO ENAME      JOB             MGR        DNO
---------- ---------- ---------- ---------- ----------
      111 saketh     analyst         444         10
```

222 sandeep clerk 333 20
333 jagan manager 111 10
444 madhu engineer 222 40

1. **Equijoin:**
   A join which contains an equal to '=' operator in this joins condition

   SQL> select eno,ename,job,dname,loc from emp2 e,dept d where e.dno=d.dno;

```
    ENO ENAME      JOB        DNAME      LOC
---------- ---------- ---------- ---------- ----------
    111 saketh     analyst    inventory  hyd
    222 sandeep    clerk      finance    bglr
    333 jagan      manager    inventory  hyd
```

   Using Clause:
   SQL> select eno,ename,job,dname,loc from emp2 e join dept d using(dno);

```
    ENO ENAME      JOB        DNAME      LOC
---------- ---------- ---------- ---------- ----------
    111 saketh     analyst    inventory  hyd
    222 sandeep    clerk      finance    bglr
    333 jagan      manager    inventory  hyd
```

   On Clause:
   SQL> select eno,ename,job,dname,loc from emp2 e join dept d on(e.dno=d.dno);

```
    ENO ENAME      JOB        DNAME      LOC
---------- ---------- ---------- ---------- ----------
    111 saketh     analyst    inventory  hyd
    222 sandeep    clerk      finance    bglr
    333 jagan      manager    inventory  hyd
```

2. **Non-Equijoin:**
   A join which contains an operator other than equal to '=' in the join condition.
   SQL> select eno,ename,job,dname,loc from emp2 e,dept d where e.dno>d.dno;

```
    ENO ENAME      JOB        DNAME      LOC
---------- ---------- ---------- ---------- ----------
    222 sandeep    clerk      inventory  hyd
    444 madhu      engineer   inventory  hyd
    444 madhu      engineer   finance    bglr
    444 madhu      engineer   HR         Mumbai
```

3. **Self Join:**
   Joining the table itself is called self join.

   SQL> select e1.eno,e2.ename,e1.job,e2.dno from emp2 e1,emp2 e2 where e1.eno=e2 gr;

```
       ENO ENAME      JOB              DNO
---------- ---------- ---------- ----------
       444 saketh     engineer          10
       333 sandeep    manager           20
       111 jagan      analyst           10
       222 madhu      clerk             40
```

4. **Natural Join:**
   It compares all the common columns.

   SQL> select eno,ename,job,dname,loc from emp2 natural join dept;

```
       ENO ENAME      JOB        DNAME      LOC
---------- ---------- ---------- ---------- ----------
       111 saketh     analyst    inventory  hyd
       222 sandeep    clerk      finance    bglr
       333 jagan      manager    inventory  hyd
```

5. **Cross Join:**
   This will give the cross product.

   SQL> select eno,ename,job,dname,loc from emp2 cross join dept;

```
       ENO ENAME      JOB        DNAME      LOC
---------- ---------- ---------- ---------- ----------
       111 saketh     analyst    inventory  hyd
       222 sandeep    clerk      inventory  hyd
       333 jagan      manager    inventory  hyd
       444 madhu      engineer   inventory  hyd
       111 saketh     analyst    finance    bglr
       222 sandeep    clerk      finance    bglr
       333 jagan      manager    finance    bglr
       444 madhu      engineer   finance    bglr
       111 saketh     analyst    HR         mumbai
       222 sandeep    clerk      HR         mumbai
       333 jagan      manager    HR         mumbai

       ENO ENAME      JOB        DNAME      LOC
```

---------- ---------- ---------- ---------- ----------
444 madhu engineer HR mumbai

12 rows selected.

6. Outer Join:

It gives the non matching records along with matching records.

6.1 Left Outer Join:

This will display the all matching records and the records which are in left hand side table those that are in right hand side table.

SQL> select eno,ename,job,dname,loc from emp2 e left outer join dept d on(e.dno= d.dno);

(OR)

SQL> select eno,ename,job,dname,loc from emp2 e,dept d where e.dno=d.dno(+);

ENO ENAME JOB DNAME LOC
---------- ---------- ---------- ---------- ----------
333 jagan manager inventory hyd
111 saketh analyst inventory hyd
222 sandeep clerk finance bglr
444 madhu engineer

6.2 Right Outer Join:

This will display the all matching records and the records which are in right hand side table those that are not in left hand side table.

SQL> select eno,ename,job,dname,loc from emp2 e right outer join dept d on(e.dno =d.dno);

(OR)

SQL> select eno,ename,job,dname,loc from emp2 e,dept d where e.dno(+)=d.dno;

ENO ENAME JOB DNAME LOC
---------- ---------- ---------- ---------- ----------
111 saketh analyst inventory hyd
222 sandeep clerk finance bglr
333 jagan manager inventory hyd
HR mumbai

6.3 Full Outer Join:

This will display the all matching records and the non matching records from both tables.

SQL> select eno,ename,job,dname,loc from emp2 e full outer join dept d on(e.dno= d.dno);

ENO ENAME JOB DNAME LOC
---------- ---------- ---------- ---------- ----------
333 jagan manager inventory hyd

111	saketh	analyst	inventory	hyd
222	sandeep	clerk	finance	bglr
444	madhu	engineer		
			HR	Mumbai

RESULT

Thus the relationship between databases has been implemented using join operation.

Ex. No: 6 Study of PL/SQL block.

AIM

To study about PL/SQL block in database management systems

DESCRIPTION

PL/SQL PROGRAMMING

Procedural Language/Structured Query Language (PL/SQL) is an extension of SQL.

Basic Syntax of PL/SQL

```
DECLARE
/* Variables can be declared here */
BEGIN
/* Executable statements can be written here */
EXCEPTION
/* Error handlers can be written here. */
END;
```

Steps to Write & Execute PL/SQL

- As we want output of PL/SQL Program on screen, before Starting writing anything type (Only Once per session)

```
SQL> SET SERVEROUTPUT ON
```

- To write program, use Notepad through Oracle using ED command.

```
SQL> ED ProName
```

Type the program Save & Exit.

- To Run the program

```
SQL> @ProName
```

Decision making with IF statement :-

The general syntax for the using IF—ELSE statement is

```
IF(TEST_CONDITION) THEN
        SET OF STATEMENTS
ELSE
        SET OF STATEMENTS
END IF;
```

For Nested IF—ELSE Statement we can use IF--ELSIF—ELSE as follows

```
IF(TEST_CONDITION) THEN
        SET OF STATEMENTS
ELSIF (CONDITION)
        SET OF STATEMENTS
END IF;
```

LOOPING STATEMENTS:-

For executing the set of statements repeatedly we can use loops. The oracle supports number of looping statements like GOTO, FOR, WHILE & LOOP.
Here is the syntax of these all the types of looping statements.

GOTO STATEMENTS

```
<<LABEL>>
SET OF STATEMENTS
GOTO LABEL;
```

FOR LOOP

```
FOR <VAR> IN [REVERSE] <INI_VALUE>..<END_VALUE>
SET OF STATEMENTS
```

END LOOP;

WHILE LOOP
WHILE (CONDITION) LOOP
SET OF STATEMENTS
END LOOP;

LOOP STATEMENT
LOOP
SET OF STATEMENTS
IF (CONDITION) THEN
EXIT
SET OF STATEMENTS
END LOOP;
While using LOOP statement, we have take care of EXIT condition, otherwise it may go into infinite loop.

1. **TO DISPLAY HELLO MESSAGE**

```
SQL> set serveroutput on;
SQL> declare
  2  a varchar2(20);
  3  begin
  4  a:='Hello';
  5  dbms_output.put_line(a);
  6  end;
  7  /
Hello
PL/SQL procedure successfully completed.
```

2. **Insert the record into Sailors table by reading the values from the Keyboard.**

```
SQL> create table sailors(sid number(10),sname varchar(10),rating number(10),age
 number(10));

Table created.

SQL> set serveroutput on
SQL> declare
  2  sid number(10):=&sid;
  3  sname varchar(10):='&sname';
  4  rating number(10):=&rating;
  5  age number(10):=&age;
  6  begin
  7  insert into sailors values(sid,sname,rating,age);
  8  end;
  9  /
Enter value for sid: 02
old   2: sid number(10):=&sid;
```

```
new   2: sid number(10):=02;
Enter value for sname: lavanya
old   3: sname varchar(10):='&sname';
new   3: sname varchar(10):='lavanya';
Enter value for rating: 01
old   4: rating number(10):=&rating;
new   4: rating number(10):=01;
Enter value for age: 25
old   5: age number(10):=&age;
new   5: age number(10):=25;

PL/SQL procedure successfully completed.

SQL> /
Enter value for sid: 03
old   2: sid number(10):=&sid;
new   2: sid number(10):=03;
Enter value for sname: vani
old   3: sname varchar(10):='&sname';
new   3: sname varchar(10):='vani';
Enter value for rating: 02
old   4: rating number(10):=&rating;
new   4: rating number(10):=02;
Enter value for age: 25
old   5: age number(10):=&age;
new   5: age number(10):=25;

PL/SQL procedure successfully completed.

SQL> select * from sailors;

      SID SNAME         RATING       AGE
---------- ---------- ---------- ----------
         2 lavanya             1         25
         3 vani                2         25
```

RESULT

Thus the PL/SQL block has been studied and implemented.

Ex. No: 7 Write a PL/SQL block to satisfy some conditions by accepting input from the user.

AIM

To implement various programs using PL/SQL language.

PROGRAMS

TO INPUT A VALUE FROM THE USER AND DISPLAY IT

```
SQL> set serveroutput on;
SQL> declare
  2  a varchar2(20);
  3  begin
  4  a:=&a;
  5  dbms_output.put_line(a);
  6  end;
  7  /
Enter value for a: 5
old   4: a:=&a;
new   4: a:=5;
5
PL/SQL procedure successfully completed.
```

GREATEST OF THREE NUMBERS

```
SQL> set serveroutput on;
SQL> declare
  2  a number(7);
  3  b number(7);
  4  c number(7);
  5  begin
  6  a:=&a;
  7  b:=&b;
  8  c:=&c;
  9  if(a>b and a>c) then
 10  dbms_output.put_line (' The greatest of the three is ' || a);
 11  else if (b>c) then
 12  dbms_output.put_line (' The greatest of the three is ' || b);
 13  else
 14  dbms_output.put_line (' The greatest of the three is ' || c);
 15  end if;
 16  end if;
 17  end;
 18  /
Enter value for a: 5
old   6: a:=&a;
new   6: a:=5;
Enter value for b: 7
old   7: b:=&b;
new   7: b:=7;
Enter value for c: 1
old   8: c:=&c;
new   8: c:=1;
The greatest of the three is 7
```

PL/SQL procedure successfully completed.

PRINT NUMBERS FROM 1 TO 5 USING SIMPLE LOOP

```
SQL> set serveroutput on;
SQL> declare
  2  a number:=1;
  3  begin
  4  loop
  5  dbms_output.put_line (a);
  6  a:=a+1;
  7  exit when a>5;
  8  end loop;
  9  end;
 10  /
1
2
3
4
5
```

PL/SQL procedure successfully completed.

PRINT NUMBERS FROM 1 TO 4 USING WHILE LOOP

```
SQL> set serveroutput on;
SQL> declare
  2  a number:=1;
  3  begin
  4  while(a<5)
  5  loop
  6  dbms_output.put_line (a);
  7  a:=a+1;
  8  end loop;
  9  end;
 10  /
1
2
3
4
```

PL/SQL procedure successfully completed.

PRINT NUMBERS FROM 1 TO 5 USING FOR LOOP

```
SQL> set serveroutput on;
SQL> declare
  2  a number:=1;
  3  begin
  4  for a in 1..5
  5  loop
  6  dbms_output.put_line (a);
```

```
  7  end loop;
  8  end;
  9  /
1
2
3
4
5
PL/SQL procedure successfully completed.
```

TO CREATE SACCOUNT TABLE

```
SQL> create table saccount  ( accno number(5), name varchar2(20), bal number(10));
Table created.
SQL> insert into saccount values ( 1,'mala',20000);
1 row created.
SQL> insert into saccount values (2,'kala',30000);
1 row created.
SQL> select * from saccount;
   ACCNO NAME                 BAL
--------- -------------------- ---------
       1 mala                 20000
       2 kala                 30000
SQL> set serveroutput on;
SQL> declare
  2  a_bal number(7);
  3  a_no varchar2(20);
  4  debit number(7):=2000;
  5  minamt number(7):=500;
  6  begin
  7  a_no:=&a_no;
  8  select bal into a_bal from saccount where accno= a_no;
  9  a_bal:= a_bal-debit;
 10  if (a_bal > minamt) then
 11  update  saccount  set bal=bal-debit where accno=a_no;
 12  end if;
 13  end;
 14
 15  /
Enter value for a_no: 1
old   7: a_no:=&a_no;
new   7: a_no:=1;
PL/SQL procedure successfully completed.
SQL> select * from saccount;
  ACCNO NAME                 BAL
--------- -------------------- ---------
```

```
1 mala                 18000
2 kala                 30000
```

TO CREATE TABLE SROUTES

```
SQL> create table sroutes ( rno number(5), origin varchar2(20), destination varchar2(20), fare numbe
r(10), distance number(10));
Table created.
SQL> insert into sroutes values ( 2, 'chennai', 'dindugal', 400,230);
1 row created.
SQL> insert into sroutes values ( 3, 'chennai', 'madurai', 250,300);
1 row created.
SQL> insert into sroutes values ( 6, 'thanjavur', 'palani', 350,370);
1 row created.
SQL> select * from sroutes;
    RNO ORIGIN               DESTINATION               FARE  DISTANCE
--------- -------------------- -------------------- --------- ---------
       2 chennai              dindugal                  400       230
       3 chennai              madurai                   250       300
       6 thanjavur            palani                    350       370
SQL> set serveroutput on;
SQL> declare
  2  route sroutes.rno % type;
  3  fares sroutes.fare % type;
  4  dist sroutes.distance % type;
  5  begin
  6  route:=&route;
  7  select fare, distance into fares , dist from sroutes where rno=route;
  8  if (dist < 250) then
  9  update sroutes set fare=300 where rno=route;
 10  else if dist between 250 and 370 then
 11  update sroutes set fare=400 where rno=route;
 12  else if (dist > 400) then
 13  dbms_output.put_line('Sorry');
 14  end if;
 15  end if;
 16  end if;
 17  end;
 18  /
Enter value for route: 3
old   6: route:=&route;
new   6: route:=3;
PL/SQL procedure successfully completed.
SQL> select * from sroutes;
    RNO ORIGIN               DESTINATION               FARE  DISTANCE
```

--------- ---------- ---------- --------- ---------

2	chennai	dindugal	400	230
3	chennai	madurai	400	300
6	thanjavur	palani	350	370

RESULT

The various programs in PL/SQL were implemented and their output was verified.

Ex. No: 8 Write a PL/SQL block that handles all types of Exceptions.

AIM:

To write a PL/SQL program with exception handling mechanisms.

DESCRIPTION:

PL/SQL provides a feature to handle the Exceptions which occur in a PL/SQL Block known as exception Handling. Using Exception Handling we can test the code and avoid it from exiting abruptly.

When an exception occurs amessages which explains its cause is recieved. PL/SQL Exception message consists of three parts.
1) Type of Exception
2) An Error Code
3) A message

General Syntax for coding the exception section

```
DECLARE
  Declaration section
BEGIN
  Exception section
EXCEPTION
WHEN ex_name1 THEN
   -Error handling statements
WHEN ex_name2 THEN
   -Error handling statements
WHEN Others THEN
  -Error handling statements
END;
```

Program with user defined exception:

```
SQL> DECLARE
  2  N INTEGER:=&N;
  3  A EXCEPTION;
  4  B EXCEPTION;
  5  BEGIN
  6  IF MOD(N,2)=0 THEN
  7  RAISE A;
  8  ELSE
  9  RAISE B;
 10  END IF;
 11  EXCEPTION
 12  WHEN A THEN
 13  DBMS_OUTPUT.PUT_LINE('THE INPUT IS EVEN.....')
 14  WHEN B THEN
 15  DBMS_OUTPUT.PUT_LINE('THE INPUT IS ODD.....');
 16  END;
 17  /
Enter value for n: 20
old   2: N INTEGER:=&N;
new   2: N INTEGER:=20;
```

```
THE INPUT IS EVEN.....

PL/SQL procedure successfully completed.

SQL> /
Enter value for n: 21
old   2: N INTEGER:=&N;
new   2: N INTEGER:=21;
THE INPUT IS ODD.....

PL/SQL procedure successfully completed.
```

Program with system defined exception:
Divide by zero exception:

```
SQL> DECLARE
  2   L_NUM1 NUMBER;
  3   L_NUM2 NUMBER;
  4
  5  BEGIN
  6   L_NUM1 := 10;
  7   L_NUM2 := 0;
  8   DBMS_OUTPUT.PUT_LINE('RESULT:'||L_NUM1/L_NUM2);
 10  EXCEPTION
 11  WHEN ZERO_DIVIDE THEN
 12   DBMS_OUTPUT.PUT_LINE(SQLCODE);
 13   DBMS_OUTPUT.PUT_LINE(SQLERRM);
 14
 15  END;
 16  /
-1476
ORA-01476: divisor is equal to zero

PL/SQL procedure successfully completed.
```

Handling the Exceptions on 'no data found'

```
SQL> create table employee1 (
  2   id                    number,
  3   employee_type_id          number,
  4   external_id             varchar2(30),
  5   first_name              varchar2(30),
  6   middle_name               varchar2(30),
  7   last_name               varchar2(30),
  8   name                  varchar2(100),
```

```
  9   birth_date                  date  ,
 10   gender_id                   number );
Table created.
SQL>
SQL> create table gender (
  2   id                        number,
  3   code                      varchar2(30),
  4   description               varchar2(80),
  5   active_date               date        default SYSDATE  not null,
  6   inactive_date             date );
Table created.
SQL> insert into gender ( id, code, description ) values ( 1, 'F', 'Female' );
1 row created.
SQL> insert into gender ( id, code, description ) values ( 2, 'M', 'Male' );
1 row created.
SQL> insert into gender ( id, code, description ) values ( 3, 'U', 'Unknown' );
1 row created.
SQL> set serveroutput on size 1000000;
SQL> declare
  2
  3     d_birth_date                  employee1.birth_date%TYPE;
  4     n_gender_id                   employee1.gender_id%TYPE;
  5     n_selected                    number := -1;
  6     n_id                          employee1.id%TYPE;
  7     v_first_name                  employee1.first_name%TYPE;
  8     v_last_name                   employee1.last_name%TYPE;
  9     v_middle_name                    employee1.middle_name%TYPE;
 10     v_name                        employee1.name%TYPE;
 11
 12     begin
 13       v_first_name  := 'JOHN';
 14      v_middle_name := 'J.';
 15       v_last_name   := 'DOUGH';
 16       v_name        := rtrim(v_last_name||', '||v_first_name||' '||v_middle_name);
 17       d_birth_date  := to_date('19800101', 'YYYYMMDD');
 18
 19       begin
 20         select id into n_gender_id from gender where code = 'M';
 21       exception
 22         when OTHERS then
```

```
23          raise_application_error(-20001, SQLERRM||' on select gender');
24        end;
25
26        begin
27          select id
28         into   n_id
29          from   employee1
30          where  name       = v_name
31          and    birth_date = d_birth_date
32          and    gender_id  = n_gender_id;
34          n_selected := sql%rowcount;
35        exception
36          when NO_DATA_FOUND then
37            n_selected := sql%rowcount;
38            DBMS_OUTPUT.PUT_LINE('Caught raised exception NO_DATA_FOUND');
39          when OTHERS then
40            raise_application_error(-20002, SQLERRM||' on select employee');
41        end;
43        DBMS_OUTPUT.PUT_LINE(to_char(n_selected)||' row(s) selected.');
44      end;
45  /
Caught raised exception NO_DATA_FOUND
0 row(s) selected.
PL/SQL procedure successfully completed.
```

RESULT

Thus the PL/SQL program that handles exception has been implemented and output was verified.

Ex. No: 9 Creation of Procedures.

AIM

To write PL/SQL programs that executes the concept of procedures.

DEFINITION

A procedure or function is a logically grouped set of SQL and PL/SQL statements that perform a specific task. They are essentially sub-programs. Procedures and functions are made up of,

- Declarative part
- Executable part

- Optional exception handling part

These procedures and functions do not show the errors.

KEYWORDS AND THEIR PURPOSES

REPLACE: It recreates the procedure if it already exists.

PROCEDURE: It is the name of the procedure to be created.

ARGUMENT: It is the name of the argument to the procedure. Paranthesis can be omitted if no arguments are present.

IN: Specifies that a value for the argument must be specified when calling the procedure ie. used to pass values to a sub-program. This is the default parameter.

OUT: Specifies that the procedure passes a value for this argument back to it's calling environment after execution ie. used to return values to a caller of the sub-program.

INOUT: Specifies that a value for the argument must be specified when calling the procedure and that procedure passes a value for this argument back to it's calling environment after execution.

RETURN: It is the datatype of the function's return value because every function must return a value, this clause is required.

PROCEDURES – SYNTAX

```
create or replace procedure <procedure name> (argument {in,out,inout} datatype ) {is,as}
variable declaration;
constant declaration;
begin
PL/SQL subprogram body;
exception
exception PL/SQL block; end;
```

CREATING THE TABLE 'ITITEMS' AND DISPLAYING THE CONTENTS

```
SQL> create table ititems(itemid number(3), actualprice number(5), ordid number(4), prodid number(4));
Table created.
SQL> insert into ititems values(101, 2000, 500, 201);
1 row created.
SQL> insert into ititems values(102, 3000, 1600, 202);
1 row created.
SQL> insert into ititems values(103, 4000, 600, 202);
1 row created.
SQL> select * from ititems;
ITEMID  ACTUALPRICE    ORDID    PRODID
---------  -----------  --------  ---------
    101        2000          500       201
    102        3000         1600       202
    103        4000          600       202
```

PROGRAM FOR GENERAL PROCEDURE – SELECTED RECORD'S PRICE IS INCREMENTED BY 500 , EXECUTING THE PROCEDURE CREATED AND DISPLAYING THE UPDATED TABLE

```
SQL> create procedure itsum(identity number, total number) is price number;
  2  null_price exception;
  3  begin
```

```
 4  select actualprice into price from ititems where itemid=identity;
 5  if price is null then
 6  raise null_price;
 7  else
 8  update ititems set actualprice=actualprice+total where itemid=identity;
 9  end if;
10  exception
11  when null_price then
12  dbms_output.put_line('price is null');
13  end;
14  /
Procedure created.
SQL> exec itsum(101, 500);
PL/SQL procedure successfully completed.

SQL> select * from ititems;
 ITEMID   ACTUALPRICE     ORDID     PRODID
---------  -----------  ---------  ---------
     101         2500        500        201
     102         3000       1600        202
     103         4000        600        202
```

PROCEDURE FOR 'IN' PARAMETER – CREATION, EXECUTION

```
SQL> set serveroutput on;
SQL> create procedure yyy (a IN number) is price number;
 2  begin
 3  select actualprice into price from ititems where itemid=a;
 4  dbms_output.put_line('Actual price is ' || price);
 5  if price is null then
 6  dbms_output.put_line('price is null');
 7  end if;
 8  end;
 9  /
Procedure created.
SQL> exec yyy(103);
Actual price is 4000
PL/SQL procedure successfully completed.
```

PROCEDURE FOR 'OUT' PARAMETER – CREATION, EXECUTION

```
SQL> set serveroutput on;
SQL> create procedure zzz (a in number, b out number) is identity number;
 2  begin
 3  select ordid into identity from ititems where itemid=a;
 4  if identity<1000 then
 5   b:=100;
```

```
  6  end if;
  7  end;
  8  /
Procedure created.
SQL> declare
  2  a number;
  3  b number;
  4  begin
  5  zzz(101,b);
  6  dbms_output.put_line('The value of b is '|| b);
  7  end;
  8  /
The value of b is 100
PL/SQL procedure successfully completed.
```

PROCEDURE FOR 'INOUT' PARAMETER – CREATION, EXECUTION

```
SQL> create procedure itit ( a in out number) is
  2  begin
  3  a:=a+1;
  4  end;
  5  /
  Procedure created.

SQL> declare
  2  a number:=7;
  3  begin
  4  itit(a);
  5  dbms_output.put_line('The updated value is '||a);
  6  end;
  7  /
The updated value is 8
PL/SQL procedure successfully completed.
```

RESULT

The PL/SQL programs were executed and their respective outputs were verified.

Ex. No: 10 Creation of database Triggers and Functions

AIM

To study and implement the concepts of triggers and functions.

DEFINITION

- A trigger is a statement that is executed automatically by the system as a side effect of a modification to the database. The parts of a trigger are,
- Trigger statement: Specifies the DML statements and fires the trigger body. It also specifies the table to which the trigger is associated.
- Trigger body or trigger action: It is a PL/SQL block that is executed when the triggering statement is used.
- Trigger restriction: Restrictions on the trigger can be achieved

The different uses of triggers are as follows,

- To generate data automatically
- To enforce complex integrity constraints
- To customize complex securing authorizations
- To maintain the replicate table
- To audit data modifications

TYPES OF TRIGGERS

The various types of triggers are as follows,

- Before: It fires the trigger before executing the trigger statement.
- After: It fires the trigger after executing the trigger statement.
- For each row: It specifies that the trigger fires once per row.
- For each statement: This is the default trigger that is invoked. It specifies that the trigger fires once per statement.

VARIABLES USED IN TRIGGERS

- :new
- :old

These two variables retain the new and old values of the column updated in the database. The values in these variables can be used in the database triggers for data manipulation

TRIGGERS - SYNTAX

```
create or replace trigger triggername [before/after] {DML statements}
on [tablename] [for each row/statement]
begin
------------------------
```

```
 ------------------------
 ------------------------
exception
end;
```

USER DEFINED ERROR MESSAGE

The package “raise_application_error” is used to issue the user defined error messages

Syntax: raise_application_error(error number,‘error message‘);

The error number can lie between -20000 and -20999.

The error message should be a character string.

TO CREATE A SIMPLE TRIGGER THAT DOES NOT ALLOW INSERT UPDATE AND DELETE OPERATIONS ON THE TABLE

```
SQL> create trigger ittrigg before insert or update or delete on itempls for each row
  2  begin
  3  raise_application_error(-20010,'You cannot do manipulation');
  4  end;
  5
  6  /
Trigger created.
SQL> insert into itempls values('aaa',14,34000);
insert into itempls values('aaa',14,34000)
            *
ERROR at line 1:
ORA-20010: You cannot do manipulation
ORA-06512: at "STUDENT.ITTRIGG", line 2
ORA-04088: error during execution of trigger 'STUDENT.ITTRIGG'
SQL> delete from itempls where ename='xxx';
delete from itempls where ename='xxx'
            *
ERROR at line 1:
ORA-20010: You cannot do manipulation
ORA-06512: at "STUDENT.ITTRIGG", line 2
ORA-04088: error during execution of trigger 'STUDENT.ITTRIGG'
SQL> update itempls set eid=15 where ename='yyy';
update itempls set eid=15 where ename='yyy'
       *
ERROR at line 1:
ORA-20010: You cannot do manipulation
ORA-06512: at "STUDENT.ITTRIGG", line 2
ORA-04088: error during execution of trigger 'STUDENT.ITTRIGG'
```

TO DROP THE CREATED TRIGGER

```
SQL> drop trigger ittrigg;
Trigger dropped.
```

TO CREATE A TRIGGER THAT RAISES AN USER DEFINED ERROR MESSAGE AND DOES NOT ALLOW UPDATION AND INSERTION

```
SQL> create trigger ittriggs before insert or update of salary on itempls for each row
  2  declare
  3  triggsal itempls.salary%type;
  4  begin
  5  select salary into triggsal from itempls where eid=12;
  6  if(:new.salary>triggsal or :new.salary<triggsal) then
  7  raise_application_error(-20100,'Salary has not been changed');
  8  end if;
  9  end;
 10  /
Trigger created.
SQL> insert into itempls values ('bbb',16,45000);
insert into itempls values ('bbb',16,45000)
            *
ERROR at line 1:
ORA-04098: trigger 'STUDENT.ITTRIGGS' is invalid and failed re-validation
SQL> update itempls set eid=18 where ename='zzz';
update itempls set eid=18 where ename='zzz'
       *
ERROR at line 1:
ORA-04298: trigger 'STUDENT.ITTRIGGS' is invalid and failed re-validation
```

FUNCTIONS – SYNTAX

```
create or replace function <function name> (argument in datatype,……) return datatype {is,as}
variable declaration;
constant declaration;
begin
PL/SQL subprogram body;
exception
exception PL/SQL block;
end;
```

CREATE THE TABLE 'ITTRAIN' TO BE USED FOR FUNCTIONS

```
SQL>create table ittrain ( tno number(10), tfare number(10));
Table created.
SQL>insert into ittrain values (1001, 550);
1 row created.
SQL>insert into ittrain values (1002, 600);
1 row created.
SQL>select * from ittrain;
     TNO       TFARE
  ---------    ------------
    1001         550
    1002         600
```

TO CREATE THE TABLE 'ITEMPLS'

```
SQL> create table itempls (ename varchar2(10), eid number(5), salary number(10));
Table created.
SQL> insert into itempls values('xxx',11,10000);
1 row created.
SQL> insert into itempls values('yyy',12,10500);
1 row created.
SQL> insert into itempls values('zzz',13,15500);
1 row created.
SQL> select * from itempls;
ENAME          EID    SALARY
---------- --------- ---------
xxx             11     10000
yyy             12     10500
zzz             13     15500
```

PROGRAM FOR FUNCTION AND IT'S EXECUTION

```
SQL> create function aaa (trainnumber number) return number is
  2  trainfunction ittrain.tfare % type;
  3  begin
  4  select tfare into trainfunction from ittrain where tno=trainnumber;
  5  return(trainfunction);
  6  end;
  7  /
Function created.
SQL> set serveroutput on;
SQL> declare
  2  total number;
  3  begin
  4  total:=aaa (1001);
  5  dbms_output.put_line('Train fare is Rs. '||total);
  6  end;
  7  /
Train fare is Rs.550
PL/SQL procedure successfully completed.
```

FACTORIAL OF A NUMBER USING FUNCTION — PROGRAM AND EXECUTION

```
 SQL>  create function itfact (a number) return number is
  2   fact number:=1;
  3   b number;
  4   begin
  5   b:=a;
  6   while b>0
  7   loop
  8   fact:=fact*b;
  9   b:=b-1;
 10   end loop;
```

```
 11  return(fact);
 12  end;
 13  /
Function created.
 SQL> set serveroutput on;
SQL> declare
  2  a number:=7;
  3  f number(10);
  4  begin
  5  f:=itfact(a);
  6  dbms_output.put_line('The factorial of the given number is'||f);
  7 end;
  8  /
The factorial of the given number is 5040
PL/SQL procedure successfully completed.
```

RESULT

The triggers and functions were created, executed and their respective outputs were verified.

Ex. No: 11 SIMPLE CALCULATOR

AIM

To implement a simple calculator by using Visual Basic front end tools.

PROCEDURE:

Step1: create a new project in visual basic using the option file---> new project.

Step2: In the form use the front end tools in the toolbox like textbox, label,command button and create a front end Design for the simple calculator.

Step3: Open the properties window for the tool sand select properties. Now the properties window is opened.

Step4: Set properties for each tool in the form like caption, name, etc.

Step5: Double click each and every tool to open the project code window.

Step6: write the code for the events of the tools.

Step7: write the code for the simple operations in the calculator like
Addition, subtraction, multiplication and division.

Step7: The code is Automatically compiled at the end of each line while pressing the Enter key.

Step7: now execute the code by click the F5 button in the keyboard or select
Run--->start.

Step8: after successfully executing the project create the executable file by
Select the option file---> make file.exe.

CODING:

```
Dim a, b, c, d As Integer
Private Sub button0_Click()
display.Text = display.Text + button0.Caption
End Sub

Private Sub button1_Click()
display.Text = display.Text + button1.Caption
End Sub

Private Sub button2_Click()
display.Text = display.Text + button2.Caption
End Sub

Private Sub button3_Click()
display.Text = display.Text + button3.Caption
End Sub

Private Sub button4_Click()
```

```
display.Text = display.Text + button4.Caption
End Sub

Private Sub button5_Click()
display.Text = display.Text + button5.Caption
End Sub

Private Sub button6_Click()
display.Text = display.Text + button6.Caption
End Sub

Private Sub button7_Click()
display.Text = display.Text + button7.Caption
End Sub

Private Sub button8_Click()
display.Text = display.Text + button8.Caption
End Sub
Private Sub button9_Click()
display.Text = display.Text + button9.Caption
End Sub

Private Sub add_Click()
a = Val(display.Text)
display.Text = ""
d = 1
End Sub

Private Sub sub_Click()
a = Val(display.Text)
display.Text = ""
d = 2
End Sub
Private Sub mul_Click()
a = Val(display.Text)
display.Text = ""
d = 3
End Sub
Private Sub div_Click()
a = Val(display.Text)
display.Text = ""
d = 4
End Sub
Private Sub equalto_Click()
```

```
b = Val(display.Text)
If d = 1 Then
c = a + b
display.Text = c
ElseIf d = 2 Then
c = a - b
display.Text = c
ElseIf d = 3 Then
c = a * b
display.Text = c
ElseIf d = 4 Then
c = a / b
display.Text = c
End If
End Sub
Private Sub clear_Click()
a = 0
b = 0
c = 0
display.Text = ""
End Sub
Private Sub off_Click()
MSG = MsgBox("THANKS FOR USING FX990ES FROM NASA COPY RIGHTS RESERVED", vbOKOnly, "BYE")
End
End Sub
Private Sub decimalpoint_Click()
display.Text = display.Text + decimalpoint.Caption
End Sub
```

RESULT:

Thus the simple calculator created by using the front end tools was executed successfully.