NAME	ABHINAV RANJAN
REGISTRATION NUMBER	RA1911003010003
DEPARTMENT	B. Tech CSE
SECTION	A1
SUBJECT	DATABASE MANAGEMENT SYSTEMS
SEMESTER	VI

DBMS LAB RECORD

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

FACULTY OF ENGINEERING & TECHNOLOGY

(Formerly SRM University, Under section 3 of UGC Act, 1956)

S.R.M. NAGAR, KATTANKULATHUR –603 203, KANCHEEPURAM DISTRICT

SCHOOL OF COMPUTING
AND DEPARTMENT OF COMPUTER SCIENCE

Course Code: 18CSC303J

Course Name:Database Management Systems

Faculty Incharge: Dr.B.Muruganantham

LAB REPORT

ABHINAV RANJAN

RA1911003010003
CSE A1 SECTION
SRMIST , KTR

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

FACULTY OF ENGINEERING & TECHNOLOGY

(Formerly SRM University, Under section 3 of UGC Act, 1956)

S.R.M. NAGAR, KATTANKULATHUR –603 203, KANCHEEPURAM DISTRICT

BONAFIDE CERTIFICATE

Register No: RA1911003010003

This is to certify that this DBMS Lab Experiment Record is the bonafide work done by **Abhinav Ranjan**

of B.Tech CSE Department, SRM Institute of Science and Technology during the academic year 2021-2022 under my supervision.

SIGNATURE

Dr. B. Muruganantham
Designation
Department
SRM IST , SRM Nagar
Potheri , Kattankulathur
Tamil Nadu 603203

SIGNATURE

Course Coordinator
Associate Professor,
SRM IST , SRM Nagar
Potheri , Kattankulathur
Tamil Nadu 603203

ACKNOWLEDGEMENT

I express my heartfelt thanks to the honorable **Vice Chancellor Dr. C. MUTHAMIZHCHELVAN**, for being the beacon in all my endeavors.

I would like to extend my gratitude to the **Registrar Dr. S. Ponnusamy**, for his encouragement

I express my sincere thanks to the **Dean (College of Engineering and Technology) Dr. T. V.Gopal**, for bringing out novelty in all executions.

I also wish to thank the **Chairperson, School of Computing Dr. Revathi Venkataraman**, for imparting confidence to complete the lab experiments and this report

I am deeply grateful to the **Course project Internal guide Dr. B. Muruganantham , Associate Professor ,Department of Computing Technologies,** , for his assistance, timely suggestion and guidance throughout the duration of the lab sessions

I extend my gratitude to **Dr.M.Pushpalatha of the Department of Computing Technologies** and Departmental colleagues for their Support.

Finally, I thank my parents and friends near and dear ones who directly and indirectly contributed to the successful completion of the report. Above all, I thank the almighty for showering his blessings on me to complete the lab experiments and this report

INDEX

TITLE OF EXPERIMENT : DDL Commands

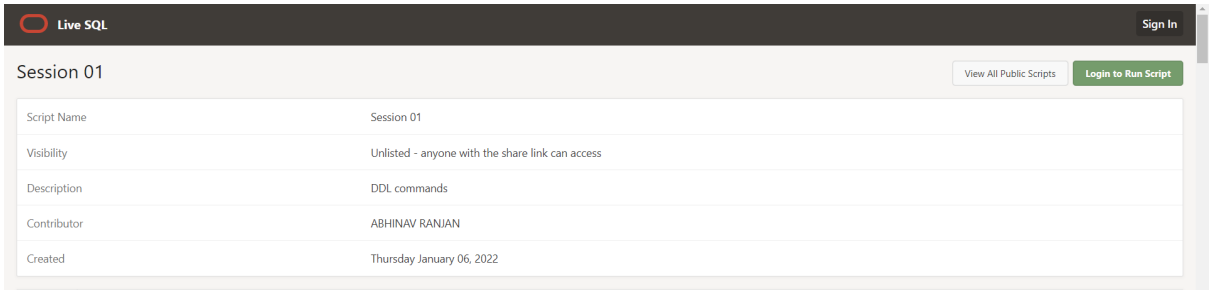
DATE OF EXPERIMENT : 6 January 2022
EXPERIMENT NO
: 01

AIM:
To perform basic Data Definition Language commands.

INTRODUCTION:
Data Definition Language or DDL commands in standard query language(SQL) are used to describe/define the database schema. These commands deal with database schema creation and its further modifications. Some popularly known DDL commands are CREATE, ALTER, DROP, TRUNCATE, and COMMENT.

Command	Description
CREATE	Used for creating database objects like a database and a database table.
ALTER	Used for modifying and renaming elements of an existing database table.
DROP	Used for removing an entire database or a database table.
TRUNCATE	Used to remove all the records from a database table.
COMMENT	Used to write comments within SQL queries.

COMMAND AND OUTPUT:



Statement 4

```
desc studentrecord
```

TABLE STUDENTRECORD

Column	Null?	Type
ROLLNO	NOT NULL	NUMBER(20,0)
SNAME	NOT NULL	VARCHAR2(20)
AGE	-	NUMBER(2,0)
FATHERNAME	-	VARCHAR2(20)
MOTHERNAME	-	VARCHAR2(20)
PNUMBER	-	NUMBER(10,0)

6 rows selected.

Statement 5

```
rename studentrecord to srecord
```

Statement processed.

Statement 6

```
desc srecord
```

TABLE SRECORD

Column	Null?	Type
ROLLNO	NOT NULL	NUMBER(20,0)
SNAME	NOT NULL	VARCHAR2(20)
AGE	-	NUMBER(2,0)
FATHERNAME	-	VARCHAR2(20)
MOTHERNAME	-	VARCHAR2(20)
PNUMBER	-	NUMBER(10,0)

6 rows selected.

Statement 7

```
alter table srecord drop column pnumber
```

Table altered.

Statement 8

```
desc srecord
```

TABLE SRECORD

Column	Null?	Type
ROLLNO	NOT NULL	NUMBER(20,0)

Statement 9

```
alter table srecord add spercent number(4)
```

Table altered.

Statement 10

```
alter table srecord modify spercent number(4,2)
```

Table altered.

Statement 11

```
desc srecord
```

TABLE SRECORD

Column	Null?	Type
ROLLNO	NOT NULL	NUMBER(20,0)
SNAME	NOT NULL	VARCHAR2(20)
AGE	-	NUMBER(2,0)
FATHERNAME	-	VARCHAR2(20)
MOTHERNAME	-	VARCHAR2(20)
SPERCENT	-	NUMBER(4,2)

6 rows selected.

Statement 12

```
desc srecord
```

TABLE SRECORD

Column	Null?	Type
ROLLNO	NOT NULL	NUMBER(20,0)
SNAME	NOT NULL	VARCHAR2(20)
AGE	-	NUMBER(2,0)
FATHERNAME	-	VARCHAR2(20)
MOTHERNAME	-	VARCHAR2(20)
SPERCENT	-	NUMBER(4,2)

6 rows selected.

Statement 23

```
insert into srecord values(015, 'Rashi Agarwal', 19, 'Dheeraj Agarwal', 'Swati Agarwal', 95.40)
```

1 row(s) inserted.

Statement 24

```
insert into srecord values(019, 'Ayush Agarwal', 19, 'Neeraj Agarwal', 'Pooja Agarwal', 91.30)
```

1 row(s) inserted.

Statement 25

```
select * from srecord
```

ROLLNO	SNAME	AGE	FATHERNAME	MOTHERNAME	PERCENT
15	Rashi Agarwal	19	Dheeraj Agarwal	Swati Agarwal	95.4
19	Ayush Agarwal	19	Neeraj Agarwal	Pooja Agarwal	91.3

2 rows selected.

Statement 26

```
alter table srecord rename column sname to name
```

Table altered.

Statement 27

```
desc srecord
```

TABLE SRECORD

Column	Null?	Type
ROLLNO	NOT NULL	NUMBER(20,0)
NAME	NOT NULL	VARCHAR2(20)
AGE	-	NUMBER(2,0)
FATHERNAME	-	VARCHAR2(20)
MOTHERNAME	-	VARCHAR2(20)
PERCENT	-	NUMBER(4,2)

6 rows selected.

Statement 28

```
select * from srecord
```

Statement 31

```
select * from srecord
```

ROLLNO	NAME	AGE	FATHERNAME	PERCENT
15	Rashi Agarwal	19	Dheeraj Agarwal	95.4
19	Ayush Agarwal	19	Neeraj Agarwal	91.3

2 rows selected.

Statement 32

```
truncate table srecord
```

Table truncated.

Statement 33

```
select * from srecord
```

no data found

Statement 34

```
desc srecord
```

TABLE SRECORD

Column	Null?	Type
ROLLNO	NOT NULL	NUMBER(20,0)
NAME	NOT NULL	VARCHAR2(20)
AGE	-	NUMBER(2,0)
FATHERNAME	-	VARCHAR2(20)
PERCENT	-	NUMBER(4,2)

5 rows selected.

Statement 35

```
drop table srecord
```

Table dropped.

Statement 36

```
desc srecord
```

ORA-20001: object SRECORD does not exist

Result:
All the DDL Comm

TITLE OF EXPERIMENT : DML Commands

DATE OF EXPERIMENT : 13 January 2022
EXPERIMENT NO
: 02

AIM:
To perform basic Data Manipulation Language commands.

INTRODUCTION:
The SQL commands that deal with the manipulation of data present in the database belong to DML or Data Manipulation Language and this includes most of the SQL statements. It is the component of the SQL statement that controls access to data and to the database. Basically, DCL statements are grouped with DML statements. List of DML commands:
INSERT : It is used to insert data into a table.
UPDATE: It is used to update existing data within a table.
DELETE : It is used to delete records from a database table.

COMMAND AND OUTPUT:

Live SQL

Sign In

Session 02

View All Public Scripts

Login to Run Script

Script Name	Session 02
Visibility	Unlisted - anyone with the share link can access
Description	DML COMMANDS
Contributor	ABHINAV RANJAN
Created	Thursday January 13, 2022

Statement 1

```
CREATE TABLE EMP
       (EMPNO NUMBER(4) NOT NULL,
        ENAME VARCHAR2(10),
        JOB VARCHAR2(9),
        MGR NUMBER(4),
        HIREDATE DATE,
        SAL NUMBER(7, 2),
        COMM NUMBER(7, 2),
        DEPTNO NUMBER(2))
```

```
Table created.
```

Statement 2

```
desc emp
```

```
TABLE EMP
```

Statement 3

```
INSERT INTO EMP VALUES
        (7369, 'SMITH',  'CLERK',     7902,
        TO_DATE('17-DEC-1980', 'DD-MON-YYYY'),  800, NULL, 20)
```

```
1 row(s) inserted.
```

Statement 4

```
INSERT INTO EMP VALUES
        (7499, 'ALLEN',  'SALESMAN',  7698,
        TO_DATE('20-FEB-1981', 'DD-MON-YYYY'), 1600,  300, 30)
```

```
1 row(s) inserted.
```

Statement 5

```
INSERT INTO EMP VALUES
        (7521, 'WARD',   'SALESMAN',  7698,
        TO_DATE('22-FEB-1981', 'DD-MON-YYYY'), 1250,  500, 30)
```

```
1 row(s) inserted.
```

Statement 6

```
INSERT INTO EMP VALUES
        (7566, 'JONES',  'MANAGER',   7839,
        TO_DATE('2-APR-1981', 'DD-MON-YYYY'),  2975, NULL, 20)
```

```
1 row(s) inserted.
```

Statement 7

```
INSERT INTO EMP VALUES
        (7654, 'MARTIN', 'SALESMAN',  7698,
        TO_DATE('28-SEP-1981', 'DD-MON-YYYY'), 1250, 1400, 30)
```

```
1 row(s) inserted.
```

Statement 8

```
INSERT INTO EMP VALUES
        (7698, 'BLAKE',  'MANAGER',   7839,
        TO_DATE('1-MAY-1981', 'DD-MON-YYYY'),  2850, NULL, 30)
```

```
1 row(s) inserted.
```

Statement 17

```
select * from emp
```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300	30
7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400	30
7788	SCOTT	ANALYST	7566	09-DEC-82	3000	-	20
7844	TURNER	SALESMAN	7698	08-SEP-81	1500	0	30
7521	WARD	SALESMAN	7698	22-FEB-81	1250	500	30
7782	CLARK	MANAGER	7839	09-JUN-81	2450	-	10
7566	JONES	MANAGER	7839	02-APR-81	2975	-	20
7788	SCOTT	ANALYST	7566	09-DEC-82	3000	-	20
7876	ADAMS	CLERK	7788	12-JAN-83	1100	-	20
7934	MILLER	CLERK	7782	23-JAN-82	1300	-	10
7369	SMITH	CLERK	7902	17-DEC-80	800	-	20
7698	BLAKE	MANAGER	7839	01-MAY-81	2850	-	30
7839	KING	PRESIDENT	-	17-NOV-81	5000	-	10
7902	FORD	ANALYST	7566	03-DEC-81	3000	-	20

14 rows selected.

Statement 18

```
select empno, ename from emp
```

EMPNO	ENAME
7499	ALLEN
7654	MARTIN
7788	SCOTT
7844	TURNER
7521	WARD
7782	CLARK
7566	JONES
7788	SCOTT
7876	ADAMS
7934	MILLER
7369	SMITH
7698	BLAKE
7839	KING
7902	FORD

Statement 19

```
select * from emp where sal>1500
```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300	30
7788	SCOTT	ANALYST	7566	09-DEC-82	3000	-	20
7782	CLARK	MANAGER	7839	09-JUN-81	2450	-	10
7566	JONES	MANAGER	7839	02-APR-81	2975	-	20
7788	SCOTT	ANALYST	7566	09-DEC-82	3000	-	20
7698	BLAKE	MANAGER	7839	01-MAY-81	2850	-	30
7839	KING	PRESIDENT	-	17-NOV-81	5000	-	10
7902	FORD	ANALYST	7566	03-DEC-81	3000	-	20

8 rows selected.

Statement 20

```
select * from emp where comm = NULL
```

no data found

Statement 21

```
select * from emp where comm = null
```

no data found

Statement 22

```
select * from emp where comm is NULL
```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7788	SCOTT	ANALYST	7566	09-DEC-82	3000	-	20
7782	CLARK	MANAGER	7839	09-JUN-81	2450	-	10
7566	JONES	MANAGER	7839	02-APR-81	2975	-	20
7788	SCOTT	ANALYST	7566	09-DEC-82	3000	-	20
7876	ADAMS	CLERK	7788	12-JAN-83	1100	-	20
7934	MILLER	CLERK	7782	23-JAN-82	1300	-	10
7369	SMITH	CLERK	7902	17-DEC-80	800	-	20
7698	BLAKE	MANAGER	7839	01-MAY-81	2850	-	30
7839	KING	PRESIDENT	-	17-NOV-81	5000	-	10
7902	FORD	ANALYST	7566	03-DEC-81	3000	-	20

10 rows selected.

Statement 23

```
select empno, ename from emp where job= 'CLERK'
```

EMPNO	ENAME
7876	ADAMS
7934	MILLER
7369	SMITH

3 rows selected.

Statement 24

```
select * from emp
```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300	30
7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400	30
7788	SCOTT	ANALYST	7566	09-DEC-82	3000	-	20
7844	TURNER	SALESMAN	7698	08-SEP-81	1500	0	30
7521	WARD	SALESMAN	7698	22-FEB-81	1250	500	30
7782	CLARK	MANAGER	7839	09-JUN-81	2450	-	10
7566	JONES	MANAGER	7839	02-APR-81	2975	-	20
7788	SCOTT	ANALYST	7566	09-DEC-82	3000	-	20
7876	ADAMS	CLERK	7788	12-JAN-83	1100	-	20
7934	MILLER	CLERK	7782	23-JAN-82	1300	-	10
7369	SMITH	CLERK	7902	17-DEC-80	800	-	20
7698	BLAKE	MANAGER	7839	01-MAY-81	2850	-	30
7839	KING	PRESIDENT	-	17-NOV-81	5000	-	10
7902	FORD	ANALYST	7566	03-DEC-81	3000	-	20

14 rows selected.

Statement 25

```
select * from emp order by ename
```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7876	ADAMS	CLERK	7788	12-JAN-83	1100	-	20
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300	30
7698	BLAKE	MANAGER	7839	01-MAY-81	2850	-	30
7782	CLARK	MANAGER	7839	09-JUN-81	2450	-	10
7902	FORD	ANALYST	7566	03-DEC-81	3000	-	20
7566	JONES	MANAGER	7839	02-APR-81	2975	-	20
7839	KING	PRESIDENT	-	17-NOV-81	5000	-	10
7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400	30
7934	MILLER	CLERK	7782	23-JAN-82	1300	-	10
7788	SCOTT	ANALYST	7566	09-DEC-82	3000	-	20
7788	SCOTT	ANALYST	7566	09-DEC-82	3000	-	20

Statement 26

```
select * from emp order by ename asc
```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7876	ADAMS	CLERK	7788	12-JAN-83	1100	-	20
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300	30
7698	BLAKE	MANAGER	7839	01-MAY-81	2850	-	30
7782	CLARK	MANAGER	7839	09-JUN-81	2450	-	10
7902	FORD	ANALYST	7566	03-DEC-81	3000	-	20
7566	JONES	MANAGER	7839	02-APR-81	2975	-	20
7839	KING	PRESIDENT	-	17-NOV-81	5000	-	10
7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400	30
7934	MILLER	CLERK	7782	23-JAN-82	1300	-	10
7788	SCOTT	ANALYST	7566	09-DEC-82	3000	-	20
7788	SCOTT	ANALYST	7566	09-DEC-82	3000	-	20
7369	SMITH	CLERK	7902	17-DEC-80	800	-	20
7844	TURNER	SALESMAN	7698	08-SEP-81	1500	0	30
7521	WARD	SALESMAN	7698	22-FEB-81	1250	500	30

14 rows selected.

Statement 27

```
select * from emp order by ename, job asc
```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7876	ADAMS	CLERK	7788	12-JAN-83	1100	-	20
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300	30
7698	BLAKE	MANAGER	7839	01-MAY-81	2850	-	30
7782	CLARK	MANAGER	7839	09-JUN-81	2450	-	10
7902	FORD	ANALYST	7566	03-DEC-81	3000	-	20
7566	JONES	MANAGER	7839	02-APR-81	2975	-	20
7839	KING	PRESIDENT	-	17-NOV-81	5000	-	10
7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400	30
7934	MILLER	CLERK	7782	23-JAN-82	1300	-	10
7788	SCOTT	ANALYST	7566	09-DEC-82	3000	-	20
7788	SCOTT	ANALYST	7566	09-DEC-82	3000	-	20
7369	SMITH	CLERK	7902	17-DEC-80	800	-	20
7844	TURNER	SALESMAN	7698	08-SEP-81	1500	0	30
7521	WARD	SALESMAN	7698	22-FEB-81	1250	500	30

14 rows selected.

Statement 28

```
select * from emp order by job, ename asc
```

Statement 31

```
select job from emp group by job
```

JOB
SALESMAN
ANALYST
CLERK
MANAGER
PRESIDENT

5 rows selected.

Statement 32

```
SELECT job from emp group by job
```

JOB
SALESMAN
ANALYST
CLERK
MANAGER
PRESIDENT

5 rows selected.

Statement 33

```
SELECT job ,avg(sal), max(sal),min(sal),count(job) from emp group by job
```

JOB	AVG(SAL)	MAX(SAL)	MIN(SAL)	COUNT(JOB)
SALESMAN	1400	1600	1250	4
ANALYST	3000	3000	3000	3
CLERK	1066.6666666666666666666666666666666667	1300	800	3
MANAGER	2758.3333333333333333333333333333333333	2975	2450	3
PRESIDENT	5000	5000	5000	1

5 rows selected.

Statement 34

```
SELECT job , count(job) from emp group by job
```

JOB	COUNT(JOB)
SALESMAN	4
ANALYST	3
CLERK	3
MANAGER	3
PRESIDENT	1

5 rows selected.

Statement 36

```
SELECT job , count(job) from emp group by job
```

JOB	COUNT(JOB)
SALESMAN	4
ANALYST	3
CLERK	3
MANAGER	3
PRESIDENT	1

5 rows selected.

Statement 37

```
SELECT job , count(job) from emp group by job
```

JOB	COUNT(JOB)
SALESMAN	4
ANALYST	3
CLERK	3
MANAGER	3
PRESIDENT	1

5 rows selected.

Statement 38

```
SELECT job , count(job) from emp group by job
```

JOB	COUNT(JOB)
SALESMAN	4
ANALYST	3
CLERK	3
MANAGER	3
PRESIDENT	1

5 rows selected.

Statement 39

```
SELECT job , count(job) from emp group by job having count(job) >2
```

JOB	COUNT(JOB)
SALESMAN	4
ANALYST	3
CLERK	3
MANAGER	3

4 rows selected.

Result : All The commands were successfully executed

TITLE OF EXPERIMENT : DCL Commands

DATE OF EXPERIMENT : 02 February 2022
EXPERIMENT NO
: 03

AIM:
To perform basic Data Control Language commands.

DATA CONTROL LANGUAGE :

Data control language (DCL) is used to access the stored data. It is mainly used for revoke and to grant the user the required access to a database. In the database, this language does not have the feature of rollback.

- It is a part of the structured query language (SQL).
- It helps in controlling access to information stored in a database. It complements the data manipulation language (DML) and the data definition language (DDL).
- It is the simplest among three commands.
- It provides the administrators, to remove and set database permissions to desired users as needed.
- These commands are employed to grant, remove and deny permissions to users for retrieving and manipulating a database.

<u>GRANT Command</u>

It is employed to grant a privilege to a user. GRANT command allows specified users to perform specified tasks

Syntax

```
GRANT privilege_name on objectname to user;
```

Here,

- privilege names are SELECT,UPDATE,DELETE,INSERT,ALTER,ALL
- objectname is table name
- user is the name of the user to whom we grant privileges

REVOKE Command

It is employed to remove a privilege from a user. REVOKE helps the owner to cancel previously granted permissions.

Syntax

```
REVOKE privilege_name on objectname from user;
```

Here,

- privilege names are SELECT,UPDATE,DELETE,INSERT,ALTER,ALL
- objectname is table name
- user is the name of the user whose privileges are removing

Example

```
GRANT SELECT, UPDATE ON employees TO Bhanu
```

Explanation − Firstly, to give the permissions to the user, we have to use the GRANT command. The privileges are SELECT because to view the records and UPDATE to modify the records. The objectname is table name which is Employee. The user name is bhanu.

```
REVOKE SELECT, UPDATE ON employees TO Bhanu
```

Explanation − Firstly, to revoke the permissions of the user, we have to use the REVOKE command. The privileges needed to revoke are SELECT because to view the records and UPDATE to modify the records. The objectname is table name which is Employee. The user name is Bhanu.

ORACLE LIVE SQL LINK -

https://livesql.oracle.com/apex/livesql/s/m1i4if89mlpakxovva0he9fx6

COMMAND AND OUTPUT:

Live SQL

Sign In

Session 03

View All Public Scripts

Login to Run Script

Script Name	Session 03
Visibility	Unlisted - anyone with the share link can access
Description	DCL commands using grant and revoke since live.sql doesn't work just wrote the three commands as u said. 1.Grant :- SQL Grant command is specifically used to provide privileges to database objects for a user. This command also allows users to grant permissions to other users too. 2.Revoke :- Revoke command withdraw user privileges on database objects if any granted. It does operations opposite to the Grant command. When a privilege is revoked from a particular user U, then the privileges granted to all other users by user U will be revoked.
Contributor	ABHINAV RANJAN
Created	Tuesday February 08, 2022

Statement 8

```
desc emp
```

TABLE EMP

Column	Null?	Type
EMNO	-	NUMBER(10,0)
EMNA	-	VARCHAR2(20)
JOB	-	VARCHAR2(10)

3 rows selected.

Statement 9

```
select * from emp
```

EMNO	EMNA	JOB
123	Bruce Wayne	CEO
321	Clerk Kent	HR
555	Steve Rogers	Security

3 rows selected.

Additional Information

Database on OTN

SQL and PL/SQL Discussion forums
Oracle Database
Download Oracle Database
Latest Database Tutorials

Result:
All the commands were successfully executed.

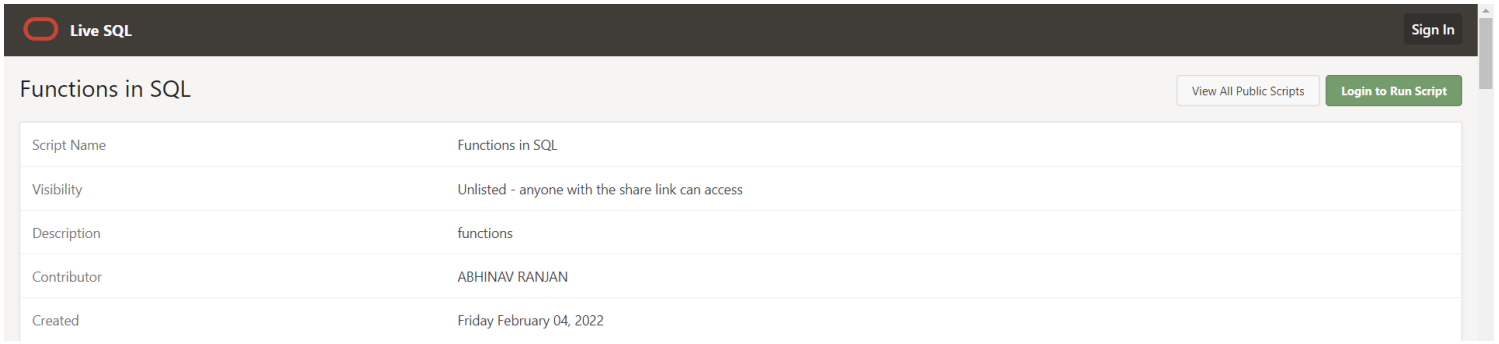
TITLE OF EXPERIMENT : Built in Functions

DATE OF EXPERIMENT : 02 February 2022
EXPERIMENT NO
: 04

AIM:
To perform Built in Functions.

INTRODUCTION:
SQL Inbuilt functions are divided into the following categories
Date Functions
Character Functions
Conversion functions
Numeric functions
Miscellaneous functions

COMMAND AND OUTPUT:



Statement 7

```
insert into abc VALUES(07,'jonas',101,TO_DATE('5-MAR-2005','DD-MM-YYYY'))
```

```
1 row(s) inserted.
```

Statement 8

```
UPDATE abc set hiredate = TO_DATE('4-MAR-2200','DD-MM-YYYY') where id = 06
```

```
1 row(s) updated.
```

Statement 9

```
select * from abc
```

ID	ENAME	SAL	HIREDATE
7	jonas	101	05-MAR-05
6	allen	125	04-MAR-00
20	abhinav	1600	03-DEC-00

3 rows selected.

Statement 10

```
select abs(sal) from abc
```

ABS(SAL)
101
125
1600

3 rows selected.

Statement 11

```
select round(sal,2) from abc
```

ROUND(SAL,2)
101
125
1600

3 rows selected.

Statement 12

```
select substr(ename,1,4) from abc
```

SUBSTR(ENAME,1,4)
jona
alle
abhi

Statement 14

```
select upper(ename) from abc
```

UPPER(ENAME)
JONAS
ALLEN
ABHINAV

3 rows selected.

Statement 15

```
select lower(ename) from abc
```

LOWER(ENAME)
jonas
allen
abhinav

3 rows selected.

Statement 16

```
select replace(ename,'j','l') from abc
```

REPLACE(ENAME,'J','L')
lonas
allen
abhinav

3 rows selected.

Statement 17

```
insert into abc values(3,'rahul',-520,TO_DATE('03-NOV-2000','DD-MM-YYYY'))
```

1 row(s) inserted.

Statement 18

```
select Ceil(sal) from abc
```

CEIL(SAL)
101
-520
125
1600

4 rows selected.

Statement 19

```
select cos(sal) from abc
```

COS(SAL)
.8920048697881601314463350992238204220242
.0663670142592570142798001894028388447 6
.7877145121442344746292146772656608541273
-.5983634637950124785293808466064343438 4

4 rows selected.

Statement 20

```
select mod(sal,2) from abc
```

MOD(SAL,2)
1
0
1
0

4 rows selected.

Statement 21

```
select power(sal,2) from abc
```

POWER(SAL,2)
10201
270400
15625
2560000

4 rows selected.

Statement 22

```
select sin(sal) from abc
```

SIN(SAL)
.4520257871783505768702669583561246036314
.9977952793125008446960779939891655412746
-.6160404591886564380273509956495825160005
-.8012247906768953631275318410432744233 8

4 rows selected.

Statement 23

```
select rtrim(ename,'a') from abc
```

4 rows selected.

Statement 27

```
select trunc(hiredate,'year') from abc
```

TRUNC(HIREDATE,'YEAR')
01-JAN-05
01-JAN-00
01-JAN-00
01-JAN-00

4 rows selected.

Statement 28

```
select add_months(hiredate,2) from abc
```

ADD_MONTHS(HIREDATE,2)
05-MAY-05
03-JAN-01
04-MAY-00
03-FEB-01

4 rows selected.

Statement 29

```
select last_day(hiredate) from abc
```

LAST_DAY(HIREDATE)
31-MAR-05
30-NOV-00
31-MAR-00
31-DEC-00

4 rows selected.

Statement 30

```
select next_day(hiredate,'monday') from abc
```

NEXT_DAY(HIREDATE,'MONDAY')
07-MAR-05
06-NOV-00
10-MAR-00
04-DEC-00

4 rows selected.

Result:
All the commands were successfully executed

ABHINAV RANJAN
RA1911003010003
CSE A1 SECTION
SRMIST , KTR

DBMS LAB 5 - ER DIAGRAM

WHAT IS AN ER DIAGRAM ?

An Entity–relationship model (ER model) describes the structure of a database with the help of a diagram, which is known as Entity Relationship Diagram (ER Diagram). An ER model is a design or blueprint of a database that can later be implemented as a database. The main components of the E-R model are: entity set and relationship set.

An ER diagram shows the relationship among entity sets. An entity set is a group of similar entities and these entities can have attributes. In terms of DBMS, an entity is a table or attribute of a table in a database, so by showing the relationship among tables and their attributes, the ER diagram shows the complete logical structure of a database.

Here are the geometric shapes and their meaning in an E-R Diagram:

Rectangle: Represents Entity sets.

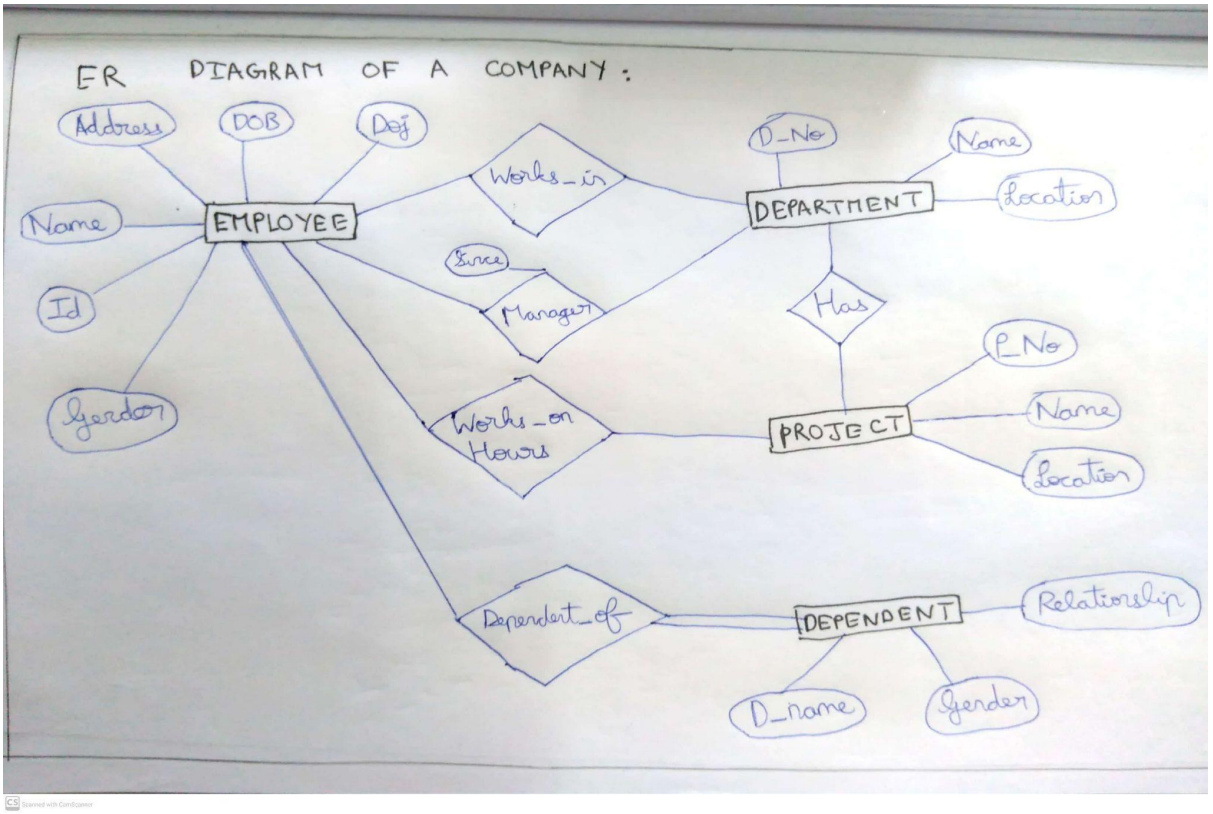
Ellipses: Attributes

Diamonds: Relationship Set
Lines: They link attributes to Entity Sets and Entity sets to Relationship Set
Double Ellipses: Multivalued Attributes
Dashed Ellipses: Derived Attributes
Double Rectangles: Weak Entity Sets
Double Lines: Total participation of an entity in a relationship set

ER DIAGRAM OF A COMPANY :

The ER diagram of a Company has the following description :

- Company has several departments.
- Each department may have several locations.
- Departments are identified by a name, D_no, Location.
- A Manager controls a particular department.
- Each department is associated with a number of projects.
- Employees are identified by name, id, address, dob, date_of_joining.
- An employee works in only one department but can work on several projects.
- We also keep track of the number of hours worked by an employee on a single project.
- Each employee has dependent
- Dependent has D_name, Gender and relationship.



This Company ER diagram illustrates key information about the Company, including entities such as employee, department, project and dependent. It allows us to understand the relationships between entities.
Entities and their Attributes are

- Employee Entity : Attributes of Employee Entity are Name, Id, Address, Gender, Dob and Doj.
  Id is the Primary Key for an Employee Entity.
- Department Entity : Attributes of the Department Entity are D_no, Name and Location.
  D_no is the Primary Key for the Department Entity.
- Project Entity : Attributes of Project Entity are P_No, Name and Location.
  P_No is the Primary Key for the Project Entity.

- Dependent Entity : Attributes of Dependent Entity are D_no, Gender and relationship.

Relationships are :

- Employees work in Departments –
  Many employees work in one Department but one employee can not work in many departments.
- Manager controls a Department –
  the employee works under the manager of the Department and the manager records the date of the employee's joining in the department.
- Department has many Projects –
  One department has many projects but one project can not come under many departments.
- Employee works on project –
  One employee works on several projects and the number of hours worked by the employee on a single project is recorded.
- Employee has dependents –
  Each Employee has dependents. Each dependent is dependent on only one employee.

ABHINAV RANJAN
RA1911003010003
CSE A1 SECTION
SRMIST , KTR

DBMS LAB 6 - NESTED QUERIES

AIM :

To execute nested queries in SQL

THEORY :

In nested queries, a query is written inside a query. The result of the inner query is used in the execution of the outer query.

There are mainly two types of nested queries:

Independent Nested Queries: In independent nested queries, query execution starts from innermost query to outermost queries. The execution of the inner query is independent of the outer query, but the result of the inner query is used in the execution of the outer query. Various operators like IN, NOT IN, ANY, ALL etc are used in writing independent nested queries.

Correlated Nested Queries: In correlated nested queries, the output of the inner query depends on the row which is being currently executed in the outer query.

GUIDELINES FOR NESTED QUERIES :

- Enclose subqueries in parentheses.
- Place subqueries on the right side of the comparison operator.
- Do not add an ORDER BY clause to a subquery.
- Use single-row operators with single-row subqueries.
- Use multiple-row operators with multiple-row subqueries

ORACLE LIVE SQL LINK :

https://livesql.oracle.com/apex/livesql/s/m3s69jf248xae9mkzku7mp0f9

SCREENSHOTS :

Statement 22

```
select * from xyz
```

ENAME	SAL	JOB	EMPNO	DEPTNO
tsmjod	68000	Developer	7	-
person2	7500	manager	7876	-
person4	59570	devops	8176	2
person4	59570	devops	8176	2
person1	6900	manager	7369	-
jack	80000	Backend	7	-
verma	38000	manager	120	-
chacha	70000	Frontend	6	-
don	59950	hr	7566	-
person3	59000	devops	8976	1

10 rows selected.

Statement 23

```
select deptno,sal from xyz where sal in (select min(sal) as sal from xyz where deptno=2)
```

DEPTNO	SAL
2	59570
2	59570

2 rows selected.

Statement 24

```
insert into xyz values('zod',5550,'CLERK',311,66)
```

1 row(s) inserted.

Statement 26

```
select empno,ename,job from xyz where job!='CLERK' and sal > ANY (select sal from xyz where job='CLERK')
```

EMPNO	ENAME	JOB
7	jack	Backend
6	chacha	Frontend
7	tsmjod	Developer
7566	don	hr
8176	person4	devops
8176	person4	devops
8976	person3	devops
120	verma	manager
7876	person2	manager
7369	person1	manager

10 rows selected.

Statement 27

```
select empno,ename,job from xyz where sal > (select avg(sal) from xyz)
```

EMPNO	ENAME	JOB
7	tsmjod	Developer
8176	person4	devops
8176	person4	devops
7	jack	Backend
6	chacha	Frontend
7566	don	hr
8976	person3	devops

7 rows selected.

Statement 37

```
select ename,deptno,commission from xyz
where (sal,commission) IN (select sal,commission from xyz where deptno=30)
```

ENAME	DEPTNO	COMMISSION
people678	5	240
people678	30	240
people99	30	249

3 rows selected.

Statement 38

```
insert into xyz values('SCOTT',6569,'manager',69,020,269)
```

1 row(s) inserted.

Statement 39

```
insert into xyz values('WARD',6549,'devops',67,030,249)
```

1 row(s) inserted.

Statement 40

```
select ename,job,sal from xyz where sal IN (select sal from xyz where ename = 'SCOTT' or ename = 'WARD')
```

ENAME	JOB	SAL
WARD	devops	6549
SCOTT	manager	6569

2 rows selected.

Statement 41

```
insert into xyz values('FORD',6009,'devops',65,025,200)
```

Statement 43

```
select ename,job,sal from xyz where (sal,job) IN (select sal,job from xyz where ename = 'FORD')
```

ENAME	JOB	SAL
CHARLIE	devops	6009
FORD	devops	6009

2 rows selected.

Statement 44

```
insert into xyz values('JONES',6079,'devops',58,020,149)
```

1 row(s) inserted.

Statement 45

```
select ename,job,deptno,sal from xyz where job IN (select job from xyz where ename = 'JONES' ) and sal > (select sal from xyz where ename='FORD')
```

ENAME	JOB	DEPTNO	SAL
person4	devops	2	59570
person4	devops	2	59570
WARD	devops	30	6549
JONES	devops	20	6079
person3	devops	1	59000

5 rows selected.

Statement 46

```
alter table xyz add department varchar2(10)
```

Table altered.

RESULT :

Thus we have successfully implemented nested queries in SQL

ABHINAV RANJAN
RA1911003010003
CSE A1 SECTION
SRMIST , KTR

DBMS LAB 7 - JOINS IN SQL

AIM :

To implement different types of joins in oracle live SQL .

THEORY :

SQL JOIN

A join clause is used to combine rows from two or more tables, based on a related column between them.

SQL INNER JOIN

The INNER JOIN keyword selects records that have matching values in both tables.

INNER JOIN Syntax

```
SELECT column_name(s)
FROM table1
INNER JOIN table2
ON table1.column_name = table2.column_name;
```

SQL LEFT JOIN

The LEFT JOIN keyword returns all records from the left table (table1), and the matching records from the right table (table2). The result is 0 records from the right side, if there is no match.

LEFT JOIN Syntax

```
SELECT column_name(s)
FROM table1
LEFT JOIN table2
ON table1.column_name = table2.column_name;
```

SQL RIGHT JOIN

The RIGHT JOIN keyword returns all records from the right table (table2), and the matching records from the left table (table1). The result is 0 records from the left side, if there is no match.

RIGHT JOIN Syntax

```
SELECT column_name(s)
FROM table1
RIGHT JOIN table2
ON table1.column_name = table2.column_name;
```

SQL FULL OUTER JOIN

The FULL OUTER JOIN keyword returns all records when there is a match in left (table1) or right (table2) table records.

FULL OUTER JOIN Syntax

```
SELECT column_name(s)
FROM table1
FULL OUTER JOIN table2
ON table1.column_name = table2.column_name
WHERE condition;
```

SQL SELF JOIN

A self join is a regular join, but the table is joined with itself.

Self Join Syntax

```
SELECT column_name(s)
FROM table1 T1, table1 T2
WHERE condition;
```

ORACLE LIVE SQL LINK :

https://livesql.oracle.com/apex/livesql/s/m3vgzkr0ertnq65xl64p2oem3

SCREENSHOTS :

Statement 27

```
select * from emp,dept where emp.deptno=dept.deptno
```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO	DEPTNO	DNAME	LOC
7698	BLAKE	MANAGER	7839	01-MAY-81	2850	-	30	30	SALES	CHICAGO
7902	FORD	ANALYST	7566	03-DEC-81	3000	-	20	20	RESEARCH	DALLAS
7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400	30	30	SALES	CHICAGO
7782	CLARK	MANAGER	7839	09-JUN-81	2450	-	10	10	ACCOUNTING	NEW YORK
7844	TURNER	SALESMAN	7698	08-SEP-81	1500	0	30	30	SALES	CHICAGO
7876	ADAMS	CLERK	7788	12-JAN-83	1100	-	20	20	RESEARCH	DALLAS
7521	WARD	SALESMAN	7698	22-FEB-81	1250	500	30	30	SALES	CHICAGO
7566	JONES	MANAGER	7839	02-APR-81	2975	-	20	20	RESEARCH	DALLAS
7788	SCOTT	ANALYST	7566	09-DEC-82	3000	-	20	20	RESEARCH	DALLAS
7934	MILLER	CLERK	7782	23-JAN-82	1300	-	10	10	ACCOUNTING	NEW YORK
7839	KING	PRESIDENT	-	17-NOV-81	5000	-	10	10	ACCOUNTING	NEW YORK
7900	JAMES	CLERK	7698	03-DEC-81	950	-	30	30	SALES	CHICAGO
7369	SMITH	CLERK	7902	17-DEC-80	800	-	20	20	RESEARCH	DALLAS
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300	30	30	SALES	CHICAGO

14 rows selected.

Statement 28

```
select * from emp,dept where emp.deptno=dept.deptno(+)
```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO	DEPTNO	DNAME	LOC
7782	CLARK	MANAGER	7839	09-JUN-81	2450	-	10	10	ACCOUNTING	NEW YORK
7934	MILLER	CLERK	7782	23-JAN-82	1300	-	10	10	ACCOUNTING	NEW YORK
7839	KING	PRESIDENT	-	17-NOV-81	5000	-	10	10	ACCOUNTING	NEW YORK

Statement 32

```
select emp.ename,dept.dname,dept.deptno from emp,dept where emp.deptno=dept.deptno and dept.deptno>10
```

ENAME	DNAME	DEPTNO
BLAKE	SALES	30
FORD	RESEARCH	20
MARTIN	SALES	30
TURNER	SALES	30
ADAMS	RESEARCH	20
WARD	SALES	30
JONES	RESEARCH	20
SCOTT	RESEARCH	20
JAMES	SALES	30
SMITH	RESEARCH	20
ALLEN	SALES	30

11 rows selected.

Statement 33

```
select a.ename,a.empno,b.mgr from emp a , emp b where a.empno=b.mgr
```

ENAME	EMPNO	MGR
KING	7839	7839
JONES	7566	7566
BLAKE	7698	7698
KING	7839	7839
BLAKE	7698	7698
SCOTT	7788	7788
BLAKE	7698	7698

Statement 40

```
select * from emp,dept where emp.deptno=dept.deptno and comm is not null
```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO	DEPTNO	DNAME	LOC
7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400	30	30	SALES	CHICAGO
7844	TURNER	SALESMAN	7698	08-SEP-81	1500	0	30	30	SALES	CHICAGO
7521	WARD	SALESMAN	7698	22-FEB-81	1250	500	30	30	SALES	CHICAGO
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300	30	30	SALES	CHICAGO

4 rows selected.

Statement 41

```
select * from emp,dept where emp.deptno=dept.deptno and comm is not null order by ename
```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO	DEPTNO	DNAME	LOC
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300	30	30	SALES	CHICAGO
7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400	30	30	SALES	CHICAGO
7844	TURNER	SALESMAN	7698	08-SEP-81	1500	0	30	30	SALES	CHICAGO
7521	WARD	SALESMAN	7698	22-FEB-81	1250	500	30	30	SALES	CHICAGO

4 rows selected.

Statement 42

```
select * from emp,dept where emp.deptno=dept.deptno
```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO	DEPTNO	DNAME	LOC
7698	BLAKE	MANAGER	7839	01-MAY-81	2850	-	30	30	SALES	CHICAGO
7902	FORD	ANALYST	7566	03-DEC-81	3000	-	20	20	RESEARCH	DALLAS
7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400	30	30	SALES	CHICAGO
7782	CLARK	MANAGER	7839	09-JUN-81	2450	-	10	10	ACCOUNTING	NEW YORK

RESULT :

Thus we have successfully executed SQL queries of different types of joins in SQL.

ABHINAV RANJAN
RA1911003010003
CSE A1 SECTION
SRMIST , KTR

DBMS LAB 8 - SET OPERATORS AND VIEWS

AIM :

To implement the set operations (union , union all , intersect and minus) and view concepts in SQL.

THEORY :

SQL set operators are used to combine the results obtained from two or more queries into a single result. The queries which contain two or more subqueries are known as compounded queries.
There are four major types of SQL operators, namely:

- Union
- Union all
- Intersect
- Minus

Points to remember -

- Same number of columns must be selected by all participating SELECT statements.Column names used in the display are taken from the first query.
- Data types of the column list must be compatible/implicitly convertible by oracle. Oracle will not perform implicit type conversion if corresponding columns in the component queries belong to different data type groups.For example, if a column in the first component query is of data type DATE, and the corresponding column in the second component query is of data type CHAR,Oracle will not perform implicit conversion, but raise ORA-01790 error.
- Positional ordering must be used to sort the result set. Individual result set ordering is not allowed with Set operators. ORDER BY can appear once at the end of the query. For example,

TYPES OF SET OPERATORS :

1. Union Set Operator

The UNION set operator is used to combine the results obtained from two or more SELECT statements

2. Union All Set Operator

The UNION set operator is used to combine all the results obtained from two or more SELECT statements. Unlike the Union operator, it considers duplicate values and includes them in the final result.

3. Intersect Set Operator

The intersect set operator is used to combine all the results of two SELECT statements. But returns only those records that are common to both the SELECT statements.

4. Minus Set Operator

The MINUS set operator is used to combine all the results of two or more SELECT statements. But returns only those records that are present exclusively in the first table.

The generic syntax for working with SQL set operators is as follows:

Syntax:

```
SELECT column_name FROM table_name_1
SET OPERATOR
SELECT column_name FROM table_name_2
SET OPERATOR
SELECT column_name FROM table_name_3
```

Parameters:

The different parameters used in the syntax are :

- SET OPERATOR: Mention the type of set operation you want to perform from { Union, Union all, Intersect, Minus}
- column_name: Mention the column name on which you want to perform the set operation and want in the result set
- FROM table_name_1: Mention the first table name from which the column has to be fetched
- FROM table_name_2: Mention the second table name from which the column has to be fetched

ORACLE LIVE SQL LINK :

https://livesql.oracle.com/apex/livesql/s/m6mdqmhdsoa76nkf1r4bss9w4

SCREENSHOTS :

Statement 27

```
INSERT INTO dept VALUES(40,'OPERATIONS',511)
```

```
1 row(s) inserted.
```

Statement 28

```
create view emp_view as select * from emp
```

```
View created.
```

Statement 29

```
select * from emp_view
```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7521	WARD	SALESMAN	7698	02-DEC-89	1250	500	30
7656	MART	SALESMAN	7698	02-DEC-89	1600	300	30
7499	ALLEN	SALESMAN	7698	17-DEC-80	1600	300	30
7656	BLAKE	MANAGER	7839	05-DEC-01	1600	300	10
7489	MANISH	DIRECTOR	7839	23-JAN-01	4000	300	40

```
5 rows selected.
```

Statement 30

```
update emp_view set COMM=196 WHERE ENAME='WARD
```

```
ORA-01756: quoted string not properly terminated
```

Statement 31

```
update emp_view set COMM=196 WHERE ENAME='WARD'
```

```
1 row(s) updated.
```

Statement 31

```
update emp_view set COMM=196 WHERE ENAME='WARD'
```

1 row(s) updated.

Statement 32

```
delete emp_view where deptno=10
```

1 row(s) deleted.

Statement 33

```
select * from emp_view
```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7521	WARD	SALESMAN	7698	02-DEC-89	1250	196	30
7656	MART	SALESMAN	7698	02-DEC-89	1600	300	30
7499	ALLEN	SALESMAN	7698	17-DEC-80	1600	300	30
7489	MANISH	DIRECTOR	7839	23-JAN-01	4000	300	40

4 rows selected.

Statement 34

```
create view a_view as select ename,job,hiredate,dname from emp,dept where emp.deptno=dept.deptno order by ename,dname asc
```

View created.

Statement 35

```
select * from a_view
```

ENAME	JOB	HIREDATE	DNAME
ALLEN	SALESMAN	17-DEC-80	SALES
MANISH	DIRECTOR	23-JAN-01	OPERATIONS

RESULT :

Thus we have successfully implemented set operators and view concepts in SQL.

ABHINAV RANJAN
RA1911003010003
CSE A1 SECTION
SRMIST , KTR

DBMS LAB 9 - PL/SQL CONDITIONAL AND OPERATIVE STATEMENTS

AIM :

To build a PL SQL Program for updating the salary in emp table using some conditions.

THEORY:

Features of PL/SQL :

PL/SQL has the following features −

- PL/SQL is tightly integrated with SQL.
- It offers extensive error checking.
- It offers numerous data types.
- It offers a variety of programming structures.
- It supports structured programming through functions and procedures.
- It supports object-oriented programming.
- It supports the development of web applications and server pages.

Advantages of PL/SQL

PL/SQL has the following advantages −

- SQL is the standard database language and PL/SQL is strongly integrated with SQL. PL/SQL supports both static and dynamic SQL. Static SQL supports DML operations and transaction control from PL/SQL blocks. In Dynamic SQL, SQL allows embedding DDL statements in PL/SQL blocks.
- PL/SQL allows sending an entire block of statements to the database at one time. This reduces network traffic and provides high performance for the applications.
- PL/SQL gives high productivity to programmers as it can query, transform, and update data in a database.
- PL/SQL saves time on design and debugging by strong features, such as exception handling, encapsulation, data hiding, and object-oriented data types.
- Applications written in PL/SQL are fully portable.
- PL/SQL provides a high security level.
- PL/SQL provides access to predefined SQL packages.
- PL/SQL provides support for Object-Oriented Programming.
- PL/SQL provides support for developing Web Applications and Server Pages.

ALGORITHM:

1.Create a table emp with attributes as

- Eid
- Name
- Dept
- salary

2.Insert into emp values() and make a table

3.Insert upto 3 rows

4.ed

5.in the notepad file write the program according to the select and print total number of rows with name='Jayesh'.

6.

1. declare

2. total_rows numbers(2)
3. Begin
4. Update emp set salary = salary+100 where name='Jayesh';
5. IF sql%notfound THEN
6. dbms_output.put_line('No Employee selected');
7. ELSIF sql%found THEN
8. total_rows:=sql%rowcount;
9. dbms_output.put_line(total_rows || 'Employee selected');
10. END IF;
11. END;

7./

8.SELECT * from emp

1.For the next program display employee number employee name and employee salary for all the employees in a loop

2.

1. declare
2. Emp_no emp.eid%type;
3. Emp_name emp.name%type;
4. Emp_sal emp.salary%type;
5. CURSOR emp_cur is SELECT eid,name,salary from employee
6. Begin
7. Open emp_curr;
8. dbms_output.put_line('emp_no' || 'Employee selected');
9. END LOOP;
10. CLOSE emp_cur;
11. END;
12. /

SOURCE CODE:

1)declare

```
total_rows numbers(2)
Begin
Update emp set salary = salary+100 where name='Jayesh';
IF sql%notfound THEN
dbms_output.put_line('No Employee selected');
ELSIF sql%found THEN
total_rows:=sql%rowcount;
dbms_output.put_line(total_rows || 'Employee selected');
END IF;
END;
```

2)
```
declare
Emp_no emp.eid%type;
Emp_name emp.name%type;
Emp_sal emp.salary%type;
CURSOR emp_cur is SELECT eid,name,salary from employee
Begin
Open emp_curr;
dbms_output.put_line('emp_no' || 'Employee selected');
END LOOP;
CLOSE emp_cur;
END;
```

SCREENSHOTS:

C:\Users\cseadmin\Downloads\instantclient_11_2-20220315T030517Z-001\instantclient_11_2\sqlplus.exe

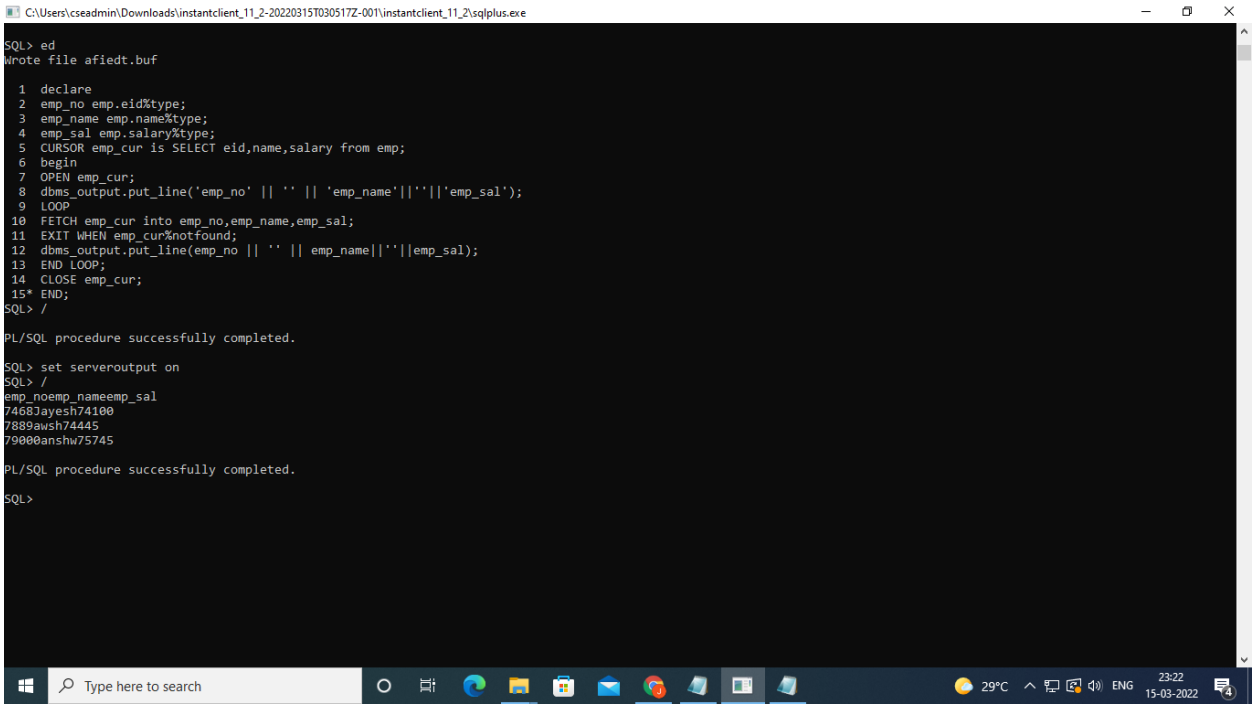
```
SQL>
SQL> CREATE TABLE EMP(EID NUMBER(10),NAME VARCHAR2(20),DEPT VARCHAR2(10),SALARY NUMBER(10));
CREATE TABLE EMP(EID NUMBER(10),NAME VARCHAR2(20),DEPT VARCHAR2(10),SALARY NUMBER(10))
            *
ERROR at line 1:
ORA-00955: name is already used by an existing object


SQL> SELECT * FROM emp
  2  /

no rows selected

SQL> desc emp
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 EID                                                NUMBER(10)
 NAME                                               VARCHAR2(10)
 SALARY                                             NUMBER(10)
 DEPT                                               VARCHAR2(20)

SQL> insert into emp values(7468,'Jayesh',74000,'clerk');

1 row created.

SQL> insert into emp values(7889,'awsh',74445,'clerk');

1 row created.

SQL> insert into emp values(79000,'anshw',75745,'clerk');

1 row created.

SQL> ed
Wrote file afiedt.buf

  1  declare
  2  total_rows number(2)
  3  begin
  4  update emp set salary = salary+100 where name='Jayesh';
  5  IF sql%notfound THEN
  6  dbms_output.put_line('No Employee selected');
  7  ELSIF sql%found THEN
```

Type here to search 29°C ENG 23:22 15-03-2022

C:\Users\cseadmin\Downloads\instantclient_11_2-20220315T030517Z-001\instantclient_11_2\sqlplus.exe

```
SQL> /
begin
*
ERROR at line 3:
ORA-06550: line 3, column 1:
PLS-00103: Encountered the symbol "BEGIN" when expecting one of the following:
:= ; not null default character
The symbol ";" was substituted for "BEGIN" to continue.


SQL> ed
Wrote file afiedt.buf

  1  declare
  2  total_rows number(2);
  3  begin
  4  update emp set salary = salary+100 where name='Jayesh';
  5  IF sql%notfound THEN
  6  dbms_output.put_line('No Employee selected');
  7  ELSIF sql%found THEN
  8  total_rows:=sql%rowcount;
  9  dbms_output.put_line(total_rows ||'Employee selected');
 10  END IF;
 11* END;
SQL> /

PL/SQL procedure successfully completed.

SQL> SELECT * FROM emp
  2  /

       EID NAME           SALARY DEPT
---------- ---------- ---------- --------------------
      7468 Jayesh          74100 clerk
      7889 awsh            74445 clerk
     79000 anshw           75745 clerk

SQL> ed
Wrote file afiedt.buf

  1  declare
  2  emp_no emp.eid%type;
  3  emp_name emp.name%type;
  4  emp_sal emp.salary%type;
```

Type here to search 29°C ENG 23:22 15-03-2022



```
C:\Users\cseadmin\Downloads\instantclient_11_2-20220315T030517Z-001\instantclient_11_2\sqlplus.exe

SQL> ed
Wrote file afiedt.buf

  1  declare
  2  emp_no emp.eid%type;
  3  emp_name emp.name%type;
  4  emp_sal emp.salary%type;
  5  CURSOR emp_cur is SELECT eid,name,salary from emp;
  6  begin
  7  OPEN emp_cur;
  8  dbms_output.put_line('emp_no' || '' || 'emp_name'||''||'emp_sal');
  9  LOOP
 10  FETCH emp_cur into emp_no,emp_name,emp_sal;
 11  EXIT WHEN emp_cur%notfound;
 12  dbms_output.put_line(emp_no || '' || emp_name||''||emp_sal);
 13  END LOOP;
 14  CLOSE emp_cur;
 15* END;
SQL> /

PL/SQL procedure successfully completed.

SQL> set serveroutput on
SQL> /
emp_noemp_nameemp_sal
7468Jayesh74100
7889awsh74445
79000anshw75745

PL/SQL procedure successfully completed.

SQL>
```

RESULT:

Thus we have successfully built a PL SQL procedure which can make changes in the emp table based on some conditions.

ABHINAV RANJAN
RA1911003010003
CSE A1 SECTION
SRMIST , KTR

DBMS LAB 10 - PL/SQL PROCEDURES

AIM :

To create PL SQL procedures and successfully execute them

REQUIREMENTS :

1. Knowledge of the concepts of PL/SQL
2. Knowledge of the concepts of procedures and its syntax
3. AWS and SQL plus insta client

THEORY:

PL SQL PROCEDURE :

The PL/SQL stored procedure or simply a procedure is a PL/SQL block which performs one or more specific tasks. It is just like procedures in other programming languages.
The procedure contains a header and a body.

- Header: The header contains the name of the procedure and the parameters or variables passed to the procedure.

- Body: The body contains a declaration section, execution section and exception section similar to a general PL/SQL block.

A procedure is created with the CREATE OR REPLACE PROCEDURE statement. The simplified syntax for the CREATE OR REPLACE PROCEDURE statement is as follows −

Syntax:

```
CREATE [OR REPLACE] PROCEDURE procedure_name
[(parameter_name [IN | OUT | IN OUT] type [, ...])]
{IS | AS}
BEGIN
  < procedure_body >
END procedure_name;
```

Where,

- procedure-name specifies the name of the procedure.
- [OR REPLACE] option allows the modification of an existing procedure.
- The optional parameter list contains name, mode and types of the parameters. IN represents the value that will be passed from outside and OUT represents the parameter that will be used to return a value outside of the procedure.
- procedure-body contains the executable part.
- The AS keyword is used instead of the IS keyword for creating a standalone procedure.

ALGORITHM :

1. Find the minimum of 2 values

- Declare 3 variables
- Create procedure with 2 input parameters x and y and one output parameter z
- Based on if else condition , minimum of x and y is found and it is stored in z
- Then call the procedure and also take dynamic inputs from the user
- After setting serveroutput on , the result is displayed and the procedure is successfully completed

2. **Find the cube of a number**
    - Declare a variable x and give “IN OUT number” (same variable is input , then some manipulation is done to it and it is shown in the output)
    - In the procedure , give x:=x*x*x . This will calculate the cube of x
    - Then take dynamic input from the user and call the function
    - The cube of the given input is calculated and the cube of it is displayed successfully

SOURCE CODE:

1. **Find minimum of 2 values**

```
DECLARE
a number;
b number;
c number;
PROCEDURE findMin(x IN number, y IN number , z
OUT number) IS
BEGIN
```

```
IF x>y THEN
  z:=y;
ELSIF x=y THEN
  z:=x;
  dbms_output.put_line('Both values are actually
same');
ELSE
  z:=x;
END IF;
END;
BEGIN
a:=&a;
b:=&b;
findMin(a,b,c);
dbms_output.put_line('Minimum of ' || a || ' and ' || b || '
is ' || c);
END;
```

2. **Find the cube of a number**

```
DECLARE
 a number;
PROCEDURE cubeofNum(x IN OUT number) IS
BEGIN
x:=x*x*x;
END;
BEGIN
a:=&a;
cubeofNum(a);
dbms_output.put_line('The cube of the given number is ' || '
is ' || a);
END;
```

SCREENSHOTS :

1. Find minimum of 2 values :

```
  5  PROCEDURE findMin(x IN number, y IN number , z OUT number) IS
  6  BEGIN
  7  IF x>y THEN
  8    z:=y;
  9  ELSIF x=y THEN
 10    z:=x;
 11    dbms_output.put_line('Both values are actually same');
 12  ELSE
 13    z:=x;
 14  END IF;
 15  END;
 16  BEGIN
 17  a:=&a;
 18  b:=&b;
 19  findMin(a,b,c);
 20  dbms_output.put_line('Minimum of ' || a || ' and ' || b || ' is ' || c);
 21* END;
SQL> /
Enter value for a: 35
old  17: a:=&a;
new  17: a:=35;
Enter value for b: 35
old  18: b:=&b;
new  18: b:=35;
Both values are actually same
Minimum of 35 and 35 is 35

PL/SQL procedure successfully completed.
```

```
 16  BEGIN
 17  a:=&a;
 18  b:=&b;
 19  findMin(a,b,c);
 20  dbms_output.put_line('Minimum of ' || a || ' and ' || b || ' is ' || c);
 21* END;
SQL> /
Enter value for a: 35
old  17: a:=&a;
new  17: a:=35;
Enter value for b: 35
old  18: b:=&b;
new  18: b:=35;
Both values are actually same
Minimum of 35 and 35 is 35

PL/SQL procedure successfully completed.

SQL> /
Enter value for a: 35
old  17: a:=&a;
new  17: a:=35;
Enter value for b: 65
old  18: b:=&b;
new  18: b:=65;
Minimum of 35 and 65 is 35

PL/SQL procedure successfully completed.
```

2. Find the cube of a number

```
Wrote file afiedt.buf

  1  DECLARE
  2  a number;
  3  PROCEDURE cubeofNum(x IN OUT number) IS
  4  BEGIN
  5  x:=x*x*x;
  6  END;
  7  BEGIN
  8  a:=&a;
  9  cubeofNum(a);
 10  dbms_output.put_line('The cube of the given number is ' || ' is ' || a);
 11* END;
SQL> /
Enter value for a: 3
old   8: a:=&a;
new   8: a:=3;

PL/SQL procedure successfully completed.

SQL> set serveroutput on
SQL> /
Enter value for a: 3
old   8: a:=&a;
new   8: a:=3;
The cube of the given number is  is 27

PL/SQL procedure successfully completed.
```

RESULT:

Thus we have successfully implemented and verified the output of PL/SQL procedures.

ABHINAV RANJAN
RA1911003010003
CSE A1 SECTION
SRMIST , KTR

DBMS LAB 11 - CURSORS

AIM :

To show the implementation of cursors in PL/SQL

THEORY:

Cursors

1. Cursor is a private SQL workgroup area allocated temporarily
2. The required amount of memory space will be allocated in cursor name
3. A cursor holds the records written by select statement
4. There are two types of cursors
    - Implicit Cursors
    - Explicit Cursors

S.No	Attribute & Description
1	%FOUND Returns TRUE if an INSERT, UPDATE, or DELETE statement affected one or more rows or a SELECT INTO statement returned one or more rows. Otherwise, it returns FALSE.
2	%NOTFOUND The logical opposite of %FOUND. It returns TRUE if an INSERT, UPDATE, or DELETE statement affected no rows, or a SELECT INTO statement returned no rows. Otherwise, it returns FALSE.
3	%ISOPEN Always returns FALSE for implicit cursors, because Oracle closes the SQL cursor automatically after executing its associated SQL statement.
4	%ROWCOUNT Returns the number of rows affected by an INSERT, UPDATE, or DELETE statement, or returned by a SELECT INTO statement.

SOURCE CODE :

1.DISPLAYING COLUMNS FROM EMP TABLE(explicit cursor)

```
DECLARE
CURSOR emp_currec is SELECT empno, ename FROM emp;
emp_rec emp_currec%rowtype;
BEGIN
OPEN emp_currec;
DBMS_OUTPUT.put_line('EmpNo' || ' ' || 'Name');
LOOP
FETCH emp_currec into emp_rec;
 EXIT WHEN emp_currec%notfound;
DBMS_OUTPUT.put_line(emp_rec.empno || ' ' || emp_rec.ename);
END LOOP;
END;
```

2.UPDATING SALARY IN EMP TABLE (implicit cursor)

```
DECLARE
  total_rows number(2);
BEGIN
  UPDATE emp
  SET sal = sal + 500;
  IF sql%notfound THEN
    dbms_output.put_line('no customers selected');
  ELSIF sql%found THEN
    total_rows := sql%rowcount;
    dbms_output.put_line( total_rows || ' customers selected ');
  END IF;
END;
```

SCREENSHOTS :

1.DISPLAYING COLUMNS FROM EMP TABLE

```
  3  emp_rec emp_currec%rowtype;
  4  BEGIN
  5  OPEN emp_currec;
  6  DBMS_OUTPUT.put_line('EmpNo' || ' ' || 'Name');
  7  LOOP
  8  FETCH emp_currec into emp_rec;
  9   EXIT WHEN emp_currec%notfound;
 10  DBMS_OUTPUT.put_line(emp_rec.empno || ' ' || emp_rec.ename);
 11  END LOOP;
 12* END;
SQL> /
EmpNo Name
7839 KING
7698 BLAKE
7782 CLARK
7566 JONES
7788 SCOTT
7902 FORD
7369 SMITH
7499 ALLEN
7521 WARD
7654 MARTIN
7844 TURNER
7876 ADAMS
7900 JAMES
7934 MILLER

PL/SQL procedure successfully completed.
```

2.UPDATING SALARY IN EMP TABLE

Table with old salary :

```
SQL> select * from emp;

     EMPNO ENAME      JOB              MGR HIREDATE         SAL       COMM
---------- ---------- --------- ---------- --------- ---------- ----------
    DEPTNO
----------
      7839 KING       PRESIDENT            17-NOV-81       5000
        10

      7698 BLAKE      MANAGER         7839 01-MAY-81       2850
        30

      7782 CLARK      MANAGER         7839 09-JUN-81       2450
        10


     EMPNO ENAME      JOB              MGR HIREDATE         SAL       COMM
---------- ---------- --------- ---------- --------- ---------- ----------
    DEPTNO
----------
      7566 JONES      MANAGER         7839 02-APR-81       2975
        20

      7788 SCOTT      ANALYST         7566 19-APR-87       3000
        20
```

Implicit cursor program :

```
SQL> ed
Wrote file afiedt.buf

  1  DECLARE
  2     total_rows number(2);
  3  BEGIN
  4     UPDATE emp
  5     SET sal = sal + 500;
  6     IF sql%notfound THEN
  7        dbms_output.put_line('no customers selected');
  8     ELSIF sql%found THEN
  9        total_rows := sql%rowcount;
 10        dbms_output.put_line( total_rows || ' customers selected ');
 11     END IF;
 12* END;
SQL> /
14 customers selected

PL/SQL procedure successfully completed.
```

New table with updated salary :

```
SQL> select * from emp;

     EMPNO ENAME      JOB              MGR HIREDATE         SAL       COMM
---------- ---------- --------- ---------- --------- ---------- ----------
    DEPTNO
----------
      7839 KING       PRESIDENT            17-NOV-81       5500
        10

      7698 BLAKE      MANAGER         7839 01-MAY-81       3350
        30

      7782 CLARK      MANAGER         7839 09-JUN-81       2950
        10


     EMPNO ENAME      JOB              MGR HIREDATE         SAL       COMM
---------- ---------- --------- ---------- --------- ---------- ----------
    DEPTNO
----------
      7566 JONES      MANAGER         7839 02-APR-81       3475
        20

      7788 SCOTT      ANALYST         7566 19-APR-87       3500
        20

      7902 FORD       ANALYST         7566 03-DEC-81       3500
```

RESULTS :

Thus we have successfully shown the implementation of cursors in PL/SQL.

ABHINAV RANJAN
RA1911003010003
CSE A1 SECTION
SRMIST , KTR

DBMS LAB 12 - EXCEPTION HANDLING IN PL/SQL

AIM :

To show the implementation of exception handling in PL/SQL and elaborate its types - system defined and user defined

THEORY :

Definition of Exception :

An exception is an error which disrupts the normal flow of program instructions. PL/SQL provides us the exception block which raises the exception thus helping the programmer to find out the fault and resolve it.

Syntax for Exception Handling

The general syntax for exception handling is as follows. Here you can list down as many exceptions as you can handle. The default exception will be handled using WHEN others THEN −

```
DECLARE
  <declarations section>
BEGIN
  <executable command(s)>
```

```
EXCEPTION
  <exception handling goes here >
  WHEN exception1 THEN
    exception1-handling-statements
  WHEN exception2  THEN
    exception2-handling-statements
  WHEN exception3 THEN
    exception3-handling-statements
  ........
  WHEN others THEN
    exception3-handling-statements
END;
```

Types of Exception :

System defined exceptions:
These exceptions are predefined in PL/SQL which get raised WHEN certain database rules are violated.
System-defined exceptions are further divided into two categories:

1. **Named system exceptions.**

They have a predefined name by the system like ACCESS_INTO_NULL, DUP_VAL_ON_INDEX, LOGIN_DENIED .

- NO_DATA_FOUND: It is raised WHEN a SELECT INTO statement returns no rows.
- TOO_MANY_ROWS:It is raised WHEN a SELECT INTO statement returns more than one row
- VALUE_ERROR:This error is raised WHEN a statement is executed that resulted in an arithmetic, numeric, string, conversion, or constraint error. This error mainly results from programmer error or invalid data input.

- ZERO_DIVIDE = raises exception WHEN dividing with zero.

2. **Unnamed system exceptions.**

Unnamed system exceptions:Oracle doesn't provide name for some system exceptions called unnamed system exceptions.These exceptions don't occur frequently.These exceptions have two parts code and an associated message. The way to handle to these exceptions is to assign name to them using Pragma EXCEPTION_INIT
Syntax:
PRAGMA EXCEPTION_INIT(exception_name, -error_number);

error_number are predefined and have a negative integer range from -20000 to -20999.

User defined exceptions:
This type of users can create their own exceptions according to the need and to raise these exceptions explicitly raise command is used.
Example:

- Divide non-negative integer x by y such that the result is greater than or equal to 1.
  From the given question we can conclude that there exist two exceptions
  - Division be zero.
  - If result is greater than or equal to 1 means y is less than or equal to x.

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.

ALGORITHM :

1. SYSTEM DEFINED EXCEPTION

- In this program , we take empno from the user as input.
- Based on empno , we search records from the emp table relating to that particular empno and display empname
- If no record is found , then a system defined exception - no_data_found is raised and appropriate message is displayed.

2. USER DEFINED EXCEPTION

- In this program , based on given empno as input , we are trying to display empname
- We define an exception which says that empno is invalid if it is lesser than 1000 or greater than 9999
- If empno is in appropriate range , output is displayed or else exception is raised

3. BOTH SYSTEM AND USER DEFINED EXCEPTION

- In this program , based on input empno we are trying to find the empname in emp table
- User defined exception is same as previous program , that range of empno should be between 1000 and 9999 else exception is raised
- System defined exception is the same as the first program . If no data is found then exception is raised

SOURCE CODE :

1. SYSTEM DEFINED EXCEPTION

```
declare
    emp_no number(10) := &empno;
```

```
    emp_name varchar2(10);
   begin
    select name into emp_name from emp where
     eid = emp_no;
     dbms_output.put_line('employee name is'||emp_name);
   exception
     when no_data_found then
        dbms_output.put_line('Not found' || emp_name);
 End;
```

2. USER DEFINED EXCEPTION

```
declare
 emp_name varchar2(10);
 emp_number number(10);
 empno_out_of_range EXCEPTION;
begin
emp_number:=&empno;
IF emp_number > 9999 OR emp_number < 1000 then
                RAISE empno_out_of_range;
                ELSE
                select name INTO emp_name from emp where
eid=emp_number;
                dbms_output.put_line('Employee name is'||emp_name);
END IF;
EXCEPTION
                WHEN empno_out_of_range THEN
                dbms_output.put_line('Employee number'||emp_number||'is
out of range');
END;
```

3. BOTH SYSTEM AND USER DEFINED EXCEPTION

```
declare
 emp_name varchar2(10);
 emp_number number(10);
 empno_out_of_range EXCEPTION;
begin
```

```
 emp_number := &empno;
IF emp_number > 9999 OR emp_number<1000 THEN
        RAISE empno_out_of_range;
ELSE
        SELECT name into emp_name from emp where eid =
emp_number;
        dbms_output.put_line('Employee name is' || emp_name);
END IF;
EXCEPTION
        WHEN empno_out_of_range THEN
         dbms_output.put_line('Exception out of range');
        WHEN NO_DATA_FOUND THEN
         dbms_output.put_line('Employee not found');
END;
```

SCREENSHOTS :

1. SYSTEM DEFINED EXCEPTION

```
SQL> ed
Wrote file afiedt.buf

  1  declare
  2    emp_no number(10) := &empno;
  3    emp_name varchar2(10);
  4  begin
  5    select name into emp_name from emp where
  6    eid = emp_no;
  7    dbms_output.put_line('employee name is'||emp_name);
  8  exception
  9    when no_data_found then
 10      dbms_output.put_line('Not found' || emp_name);
 11* end;
SQL> /
Enter value for empno: 7468
old   2:   emp_no number(10) := &empno;
new   2:   emp_no number(10) := 7468;

PL/SQL procedure successfully completed.

SQL> set serveroutput on
SQL> /
Enter value for empno: 7468
old   2:   emp_no number(10) := &empno;
new   2:   emp_no number(10) := 7468;
employee name isJayesh

PL/SQL procedure successfully completed.

SQL> /
Enter value for empno: 9999
old   2:   emp_no number(10) := &empno;
new   2:   emp_no number(10) := 9999;
Not found

PL/SQL procedure successfully completed.

SQL> _
```

2. USER DEFINED EXCEPTION

```
C:\Users\admin\Desktop\DBMS LAB\aws\instantclient_11_2\instantclient_11_2\sqlplus.exe
Connected.
SQL> ED
Wrote file afiedt.buf

  1  declare
  2   emp_name varchar2(10);
  3   emp_number number(10);
  4   empno_out_of_range EXCEPTION;
  5  begin
  6  emp_number:=&empno;
  7  IF emp_number > 9999 OR emp_number < 1000 then
  8     RAISE empno_out_of_range;
  9     ELSE
 10     select name INTO emp_name from emp where eid=emp_number;
 11     dbms_output.put_line('Employee name is'||emp_name);
 12  END IF;
 13  EXCEPTION
 14     WHEN empno_out_of_range THEN
 15     dbms_output.put_line('Employee number'||emp_number||'is out of range');
 16* END;
SQL> SET SERVEROUTPUT ON
SQL> /
Enter value for empno: 79000
old   6: emp_number:=&empno;
new   6: emp_number:=79000;
Employee number79000is out of range

PL/SQL procedure successfully completed.

SQL> /
Enter value for empno: 7468
old   6: emp_number:=&empno;
new   6: emp_number:=7468;
Employee name isJayesh

PL/SQL procedure successfully completed.

SQL> _
```

3. BOTH SYSTEM AND USER DEFINED EXCEPTION

```
SQL> ed
Wrote file afiedt.buf

  1  declare
  2   emp_name varchar2(10);
  3   emp_number number(10);
  4   empno_out_of_range EXCEPTION;
  5  begin
  6   emp_number := &empno;
  7  IF emp_number > 9999 OR emp_number<1000 THEN
  8     RAISE empno_out_of_range;
  9  ELSE
 10     SELECT name into emp_name from emp where eid = emp_number;
 11     dbms_output.put_line('Employee name is' || emp_name);
 12  END IF;
 13  EXCEPTION
 14     WHEN empno_out_of_range THEN
 15      dbms_output.put_line('Exception out of range');
 16     WHEN NO_DATA_FOUND THEN
 17      dbms_output.put_line('Employee not found');
 18* END;
SQL> /
Enter value for empno: 7468
old   6:  emp_number := &empno;
new   6:  emp_number := 7468;
Employee name isJayesh

PL/SQL procedure successfully completed.

SQL> /
Enter value for empno: 9999
old   6:  emp_number := &empno;
new   6:  emp_number := 9999;
Employee not found

PL/SQL procedure successfully completed.

SQL> _
```

RESULT :

Thus we have successfully implemented exception handling in PL/SQL

ABHINAV RANJAN
RA1911003010003
CSE A1 SECTION
SRMIST , KTR

DBMS LAB 13 - TRIGGERS IN PL/SQL

AIM :

To show the implementation of triggers in PL/SQL

THEORY :

TRIGGERS :

Triggers are stored programs, which are automatically executed or fired when some events occur.
Triggers are, in fact, written to be executed in response to any of the following events −

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

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

BENEFITS OF TRIGGERS

Triggers can be written for the following purposes −

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

SYNTAX FOR CREATING TRIGGERS :

```
CREATE [OR REPLACE ] TRIGGER trigger_name
{BEFORE | AFTER | INSTEAD OF }
{INSERT [OR] | UPDATE [OR] | DELETE}
[OF col_name]
ON table_name
[REFERENCING OLD AS o NEW AS n]
[FOR EACH ROW]
WHEN (condition)
DECLARE
  Declaration-statements
BEGIN
  Executable-statements
EXCEPTION
  Exception-handling-statements
END;
```

Where,

- CREATE [OR REPLACE] TRIGGER trigger_name − Creates or replaces an existing trigger with the trigger_name.
- {BEFORE | AFTER | INSTEAD OF} − This specifies when the trigger will be executed. The INSTEAD OF clause is used for creating trigger on a view.

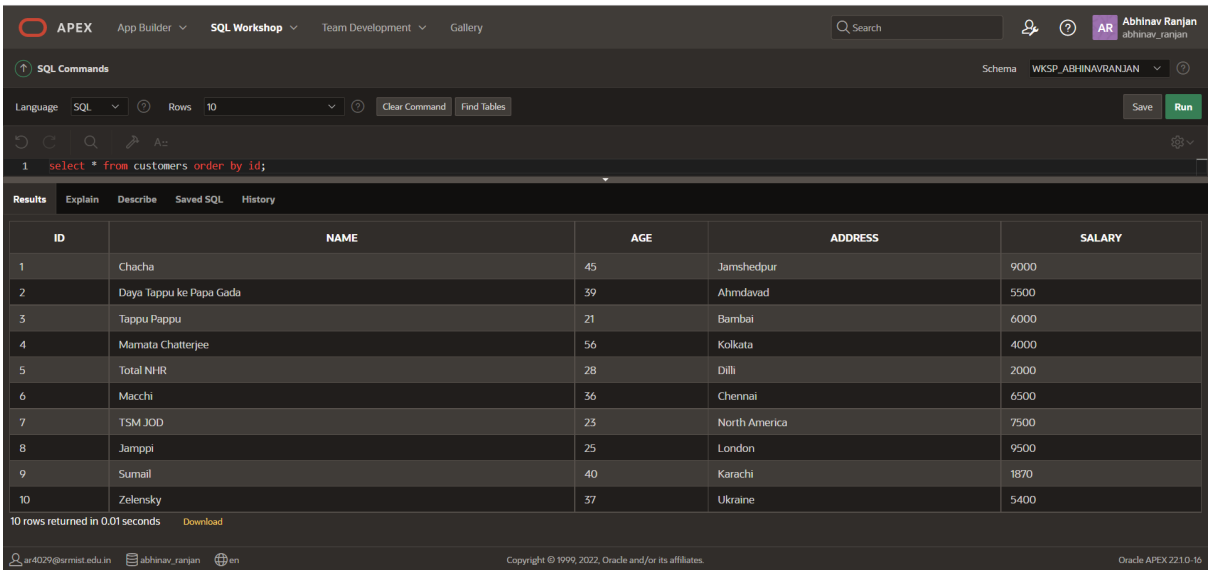
- {INSERT [OR] | UPDATE [OR] | DELETE} − This specifies the DML operation.
- [OF col_name] − This specifies the column name that will be updated.
- [ON table_name] − This specifies the name of the table associated with the trigger.
- [REFERENCING OLD AS o NEW AS n] − This allows you to refer new and old values for various DML statements, such as INSERT, UPDATE, and DELETE.
- [FOR EACH ROW] − This specifies a row-level trigger, i.e., the trigger will be executed for each row being affected. Otherwise the trigger will execute just once when the SQL statement is executed, which is called a table level trigger.
- WHEN (condition) − This provides a condition for rows for which the trigger would fire. This clause is valid only for row-level triggers.

ALGORITHM :

1. Assign a name and create trigger
2. Define the operations - before or after ,insert or update indicating when the trigger will activate
3. Declare a variable sal_diff (salary difference)
4. The variable sal_diff will hold the difference between old and new salary of a person
5. A message “trigger created” will pop up after we have finished defining trigger according to proper syntax as given above within the begin and end block
6. When we perform an insert or update operation next time , old and new salary is displayed as defined in the trigger

7. During insert , the old salary doesnt show anything as the record is inserted for the first time . The new salary is what we just inserted.
8. During the update operation it shows the old as well as the new salary

SAMPLE TABLE USED :



ID	NAME	AGE	ADDRESS	SALARY
1	Chacha	45	Jamshedpur	9000
2	Daya Tappu ke Papa Gada	39	Ahmdavad	5500
3	Tappu Pappu	21	Bambai	6000
4	Mamata Chatterjee	56	Kolkata	4000
5	Total NHR	28	Dilli	2000
6	Macchi	36	Chennai	6500
7	TSM JOD	23	North America	7500
8	Jamppi	25	London	9500
9	Sumail	40	Karachi	1870
10	Zelensky	37	Ukraine	5400

SOURCE CODE :

1. **CREATING THE TRIGGER**

```
CREATE OR REPLACE TRIGGER
display_salary_changes
BEFORE DELETE OR INSERT OR UPDATE ON
customers
FOR EACH ROW
WHEN (NEW.ID > 0)
DECLARE
  sal_diff number;
BEGIN
  sal_diff := :NEW.salary  - :OLD.salary;
  dbms_output.put_line('Old salary: ' || :OLD.salary);
  dbms_output.put_line('New salary: ' || :NEW.salary);
```

```
    dbms_output.put_line('Salary difference: ' || sal_diff);
END;
```

2. **USING UPDATE STATEMENT TO SHOW THE WORKING OF TRIGGER**

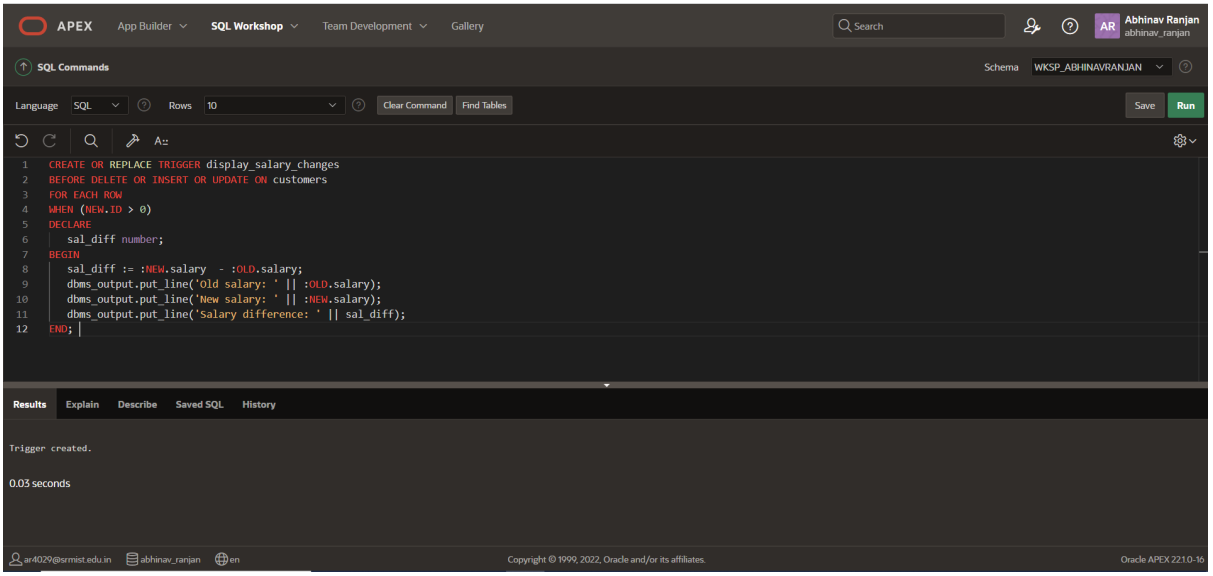
```
UPDATE customers SET salary = salary + 500
WHERE id = 2;
```

3. **USING INSERT STATEMENT TO SHOW THE WORKING OF TRIGGER**

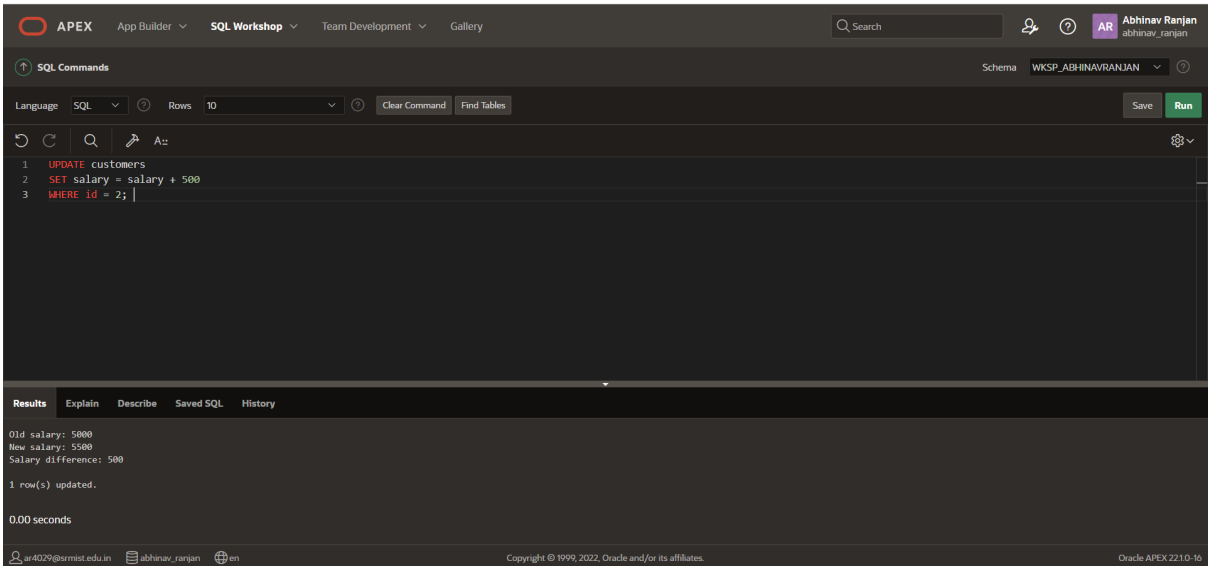
```
insert into customers
(ID,NAME,AGE,ADDRESS,SALARY)
values (11,'LaBoiDre',22,Arizona,8970);
```

SCREENSHOTS :

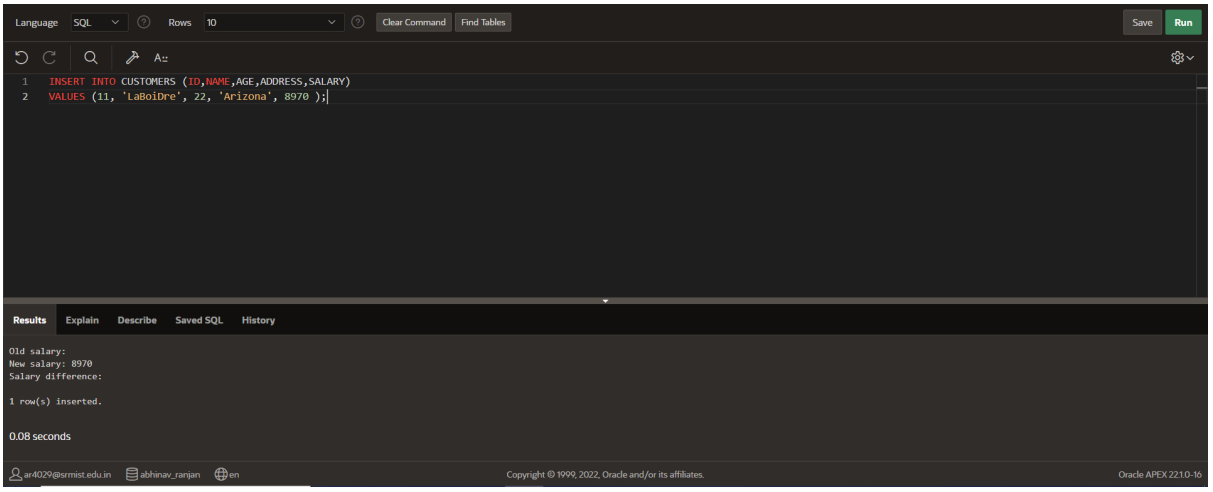
1. **CREATING THE TRIGGER**



2. **TRIGGER WORKING AFTER UPDATE STATEMENT**



3. TRIGGER WORKING AFTER INSERT STATEMENT



RESULTS :

Thus we have successfully implemented triggers in SQL and shown its working after insert and update statements have been executed.