SRM INSTITUTE OF SCIENCE AND TECHNOLOGY, Ramapuram

FACULTY OF ENGINEERING AND TECHNOLOGY

SCHOOL OF COMPUTER SCIENCE ENGINEERING

SUBJECT NAME: DATABASE MANAGEMENT SYSTEM LAB MANUAL
SUBJECT CODE: 21CSC205P
YEAR/SEMESTER: II/IV

EX. NO :1 a
DATE:

DATA DEFINITION LANGUAGE COMMANDS

AIM:

To demonstrate the use of SQL Data Definition Language (DDL) queries for creating, modifying, and managing database objects.

PROCEDURE:

Data Definition Commands:

DDL consists of SQL commands used to define and manage the structure of a database. It deals with the descriptions of the database schema and is used to create, modify, and delete database objects such as tables, indexes, and views.

Examples of DDL commands:

1. **CREATE**: Used to create a new database or its objects (such as tables, indexes, functions, views, stored procedures, and triggers).
2. **DROP**: Used to delete database objects (such as tables, indexes, or views) from the database permanently.
3. **ALTER**: Used to modify the structure of an existing database object (such as adding, modifying, or dropping columns in a table).
4. **TRUNCATE**: Used to remove all records from a table, freeing the space allocated for the data. The table structure remains intact.
5. **COMMENT**: Used to add comments or descriptions to the data dictionary or schema objects.
6. **RENAME**: Used to rename an existing object in the database, such as a table or column.

i) CREATE TABLE

The CREATE TABLE command is used to create a new table in the database.

Rules:
1. **Reserved Words**: Oracle reserved keywords cannot be used as table names or column names.
2. **Naming Restrictions**: Table names can contain underscores, numerals, and letters, but cannot contain spaces.
3. **Maximum Length**: The maximum length for a table name is 30 characters.
4. **Unique Table Names**: Each table in a database must have a unique name.
5. **Column Names**: Each column in the table must have a unique name.

6. **Data Types**: Proper data types must be specified for each column, along with the appropriate size or precision (where applicable).

Syntax for CREATE TABLE:

```
SQL> CREATE TABLE table_name (
column_name1 data_type [constraints],
column_name2 data_type [constraints],
...);
```

Explanation: The CREATE TABLE command is used to create a new table in the database with specified column names, data types, and optional constraints.

Syntax for DESC:

```
SQL> DESC table_name;
```

Explanation: The DESC (short for "DESCRIBE") command displays the structure of a table, including column names, data types, and whether the column can accept NULL values.

Creating a New Table from an Existing Table:

You can create a new table based on the structure and data of an existing table using the following syntax:

```
SQL> CREATE TABLE new_table_name AS
SELECT column1, column2, ...
FROM existing_table_name WHERE
condition;
```

Explanation: The CREATE TABLE ... AS SELECT command creates a new table by selecting specific columns from an existing table based on a condition (optional). This can be used to copy data and structure from the existing table.

Syntax:

```
SQL>Create table tablename (column_name1 data_ type constraints, column_name2 data_ type
constraints …);
```

2. DROP TABLE

The DROP TABLE command is used to delete a table and all of its data from the database permanently. Once a table is dropped, it cannot be recovered unless there is a backup.

Syntax:

```
SQL> DROP TABLE table_name;
```

3. ALTER COMMAND

The ALTER TABLE command is used to modify the structure of an existing table. It can perform the following actions:

1. Add a new column.
2. Modify the definition of an existing column.
3. Add or drop integrity constraints (such as PRIMARY KEY, FOREIGN KEY, etc.).

i) ADD COMMAND

To add a new column to an existing table, use the ADD command.

Syntax:
```
SQL> ALTER TABLE table_name ADD column_name data_type(size);
```

Explanation: The ADD command adds a new column to the table with the specified data type and size.

ii) MODIFY COMMAND

To modify an existing column’s definition (such as its data type or size), use the MODIFY command.

Syntax:
```
SQL> ALTER TABLE table_name MODIFY column_name data_type(size);
```

Explanation: The MODIFY command changes the definition of an existing column, allowing you to update its data type, size, or other attributes.

4. TRUNCATE TABLE

The TRUNCATE TABLE command removes all rows from a table but retains the table structure for future use. It is faster than DELETE, and unlike DELETE, it cannot be rolled back.

Syntax:
```
SQL> TRUNCATE TABLE table_name;
```

Explanation: The TRUNCATE command deletes all rows from the table while keeping the table structure intact. It frees up the space used by the data.

5. COMMENT

Comments can be written in SQL in three formats:

1. **Single-Line Comments**: Comments that start and end on the same line.
2. **Multi-Line Comments**: Comments that span multiple lines.
3. **Inline Comments**: Comments placed within a SQL statement.

Single-Line Comment

A comment that starts with -- and extends to the end of the line.

Syntax:
```
-- This is a single-line comment
-- Another comment
```

Multi-Line Comment

A comment that spans multiple lines, starting with /*and ending with */.

Syntax:
```
/* This is a multi-line comment
spanning multiple lines */
```

Inline Comment

An extension of the multi-line comment that can be placed within a SQL statement.

Syntax:
```
SQL> SELECT * FROM /* comment here */ table_name;
```

6. RENAME

The RENAME command is used to change the name of an existing database object, such as a table. It allows database users to give more relevant names to tables or other objects.

Syntax:
```
SQL> RENAME old_table_name TO new_table_name;
```

Explanation: The RENAME command changes the name of a table or other database object.
Program:
```
SQL> connect
Enter user-name: system
Enter password: admin
Connected.
```

```
SQL> CREATE TABLE emp (id NUMBER(10), name VARCHAR(10));
Table created.

SQL> desc emp;
Name                                    Null?    Type
 ----------------------------------------- -------- ----------------------------
ID                                               NUMBER(10)
NAME                                             VARCHAR2(10)

SQL> alter table emp add(dept varchar(10));
Table altered.

SQL> desc emp;
Name                                    Null?    Type
 ----------------------------------------- -------- ----------------------------
ID                                               NUMBER(10)
NAME                                             VARCHAR2(10)
DEPT                                             VARCHAR2(10)

SQL> alter table emp modify dept varchar(20);
Table altered.

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

SQL> alter table emp drop column dept;
Table altered.

SQL> desc emp;
Name                                    Null?    Type
 ----------------------------------------- -------- ----------------------------
ID                                               NUMBER(10)
NAME                                             VARCHAR2(10)

SQL> alter table emp rename to emp1;
Table altered.
```

```
SQL> desc emp1;
Name                                      Null?    Type
 ----------------------------------------- --------- ----------------------------
ID                                              NUMBER(10)
NAME                                            VARCHAR2(10)

SQL> desc emp2;
Name                                      Null?    Type
 ----------------------------------------- --------- ----------------------------
ID                                                   NUMBER(10)
NAME                                                 VARCHAR2(10)
DEPT                                                 VARCHAR2(10)
SQL> drop table emp2;

Table dropped.

SQL> select * from emp2;
select * from emp2
* ERROR at line
1:
ORA-00942: table or view does not exist
SQL> select * from emp1;

ID NAME              DEPT
---------- ---------- ----------
        1 aaa        cse
        2 aaa        cse
        3 aaa        ece
        4 aaa        cse
        5 aaa        cse

SQL> truncate table emp1;
Table truncated.
SQL> select * from emp1;

no rows selected

SQL> desc emp1;
Name                                       Null?    Type
 ----------------------------------------- --------- ----------------------------
ID                                                   NUMBER(10)
NAME                                                 VARCHAR2(10)
DEPT                                                 VARCHAR2(10)
```

```
SQL> drop table emp1;
Table dropped.

SQL> select * from emp1;
select * from emp1
* ERROR at line
1:
ORA-00942: table or view does not exist


SQL> desc emp1;
ERROR:
ORA-04043: object emp1 does not exist
```

Result:

Thus, the SQL Data Definition Language (DDL) queries were successfully executed to create, modify, and manage database objects.

EX.NO: 1 b
DATE:

DATA MANIPULATION COMMANDS FOR INSERTING, DELETING, UPDATING AND RETRIEVING TABLES

AIM:

To create a database and perform operations using Data Manipulation Commands (DML) for inserting, deleting, updating, and retrieving data, along with using Transaction Control statements.

DESCRIPTION:

DML statements access and manipulate data in existing tables. DML commands are the most frequently used SQL commands and is used to query and manipulate the existing database objects. Some of the commands are Insert, Select, Update, Delete.

Examples of DML:

1) **Insert Command:** This is used to add one or more rows to a table. The values are separated by commas and the data types char and date are enclosed in apostrophes. The values must be entered in the same order as they are defined.

```
SQL>INSERT INTO table_name (column1, column2, column3) VALUES (value1, value2,
value3);
```

2) **Select Commands:** It is used to retrieve information from the table. It is generally referred to as querying the table. We can either display all columns in a table or only specify column from the table.

```
SQL>SELECT * FROM table_name;  -- to retrieve all columns
SQL>SELECT column1, column2 FROM table_name;  -- to retrieve specific columns
```

3) **Update Command:** It is used to alter the column values in a table. A single column may be updated or more than one column could be updated.

```
SQL> UPDATE table_name
SET column1 = value1, column2 = value2 WHERE
condition;
```

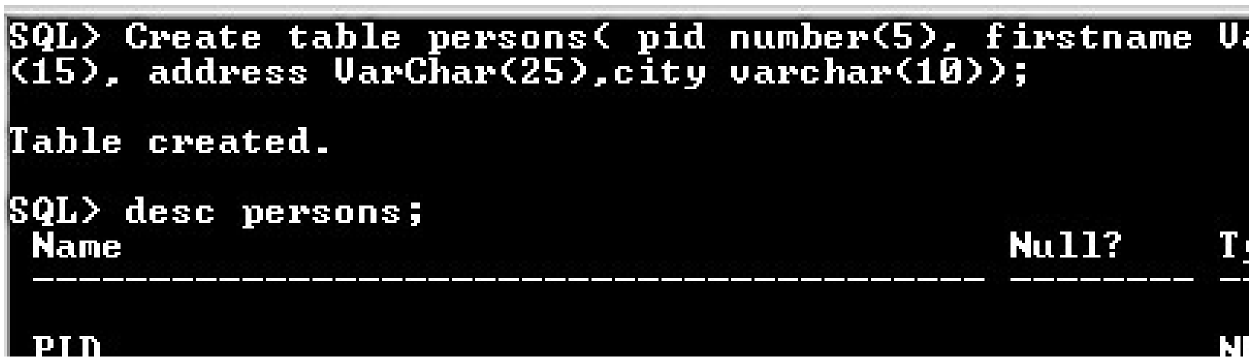
4) **Delete command:** After inserting row in a table we can also delete them if required. The delete command consists of a from clause followed by an optional where clause.

```
SQL>DELETE FROM table_name WHERE condition;
```

INSERTING VALUES INTO TABLE

Create table:

```
SQL> CREATE TABLE persons (
pid NUMBER(5),
firstname VARCHAR2(15),
lastname VARCHAR2(15),
address VARCHAR2(25), city
VARCHAR2(10)
);
```



```
SQL> Create table persons( pid number(5), firstname V
(15), address VarChar(25),city varchar(10));

Table created.

SQL> desc persons;
 Name                                      Null?    T
 ----------------------------------------- -------- -

 PID                                                N
```

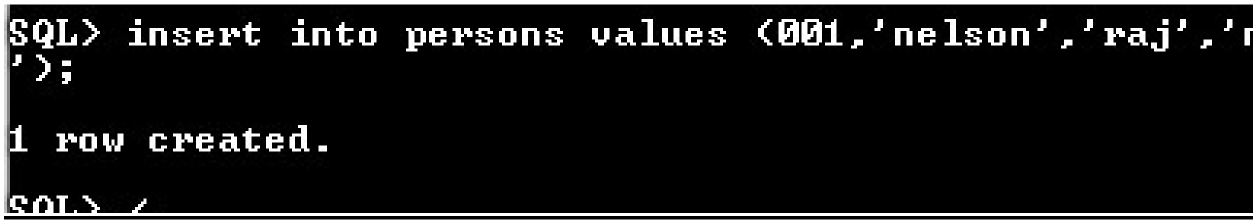
INSERT COMMAND

Insert command is used to insert values into table.

Insert a single record into table.

Syntax: SQL> INSERT INTO <table name> VALUES (value list);

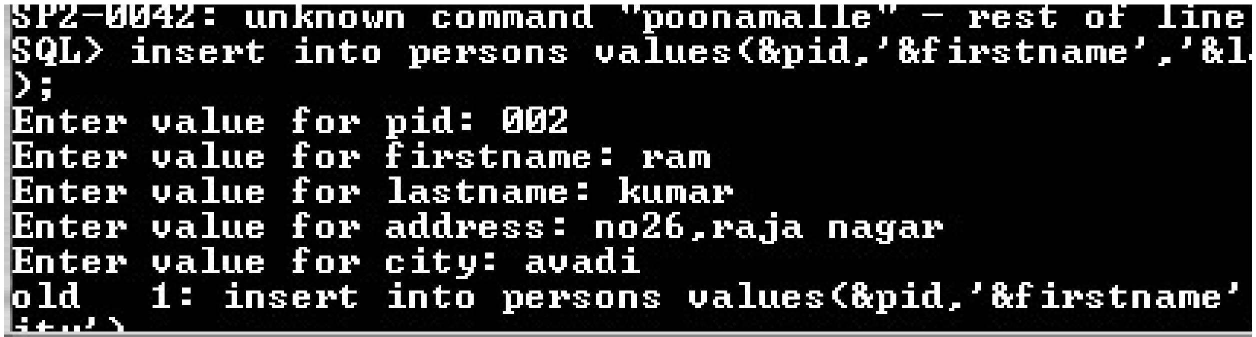
Example: SQL> INSERT INTO persons (pid, firstname, lastname, address, city)
VALUES (1, 'Nelson', 'Raj', 'No25, Annai Street', 'Chennai');
1 row created.



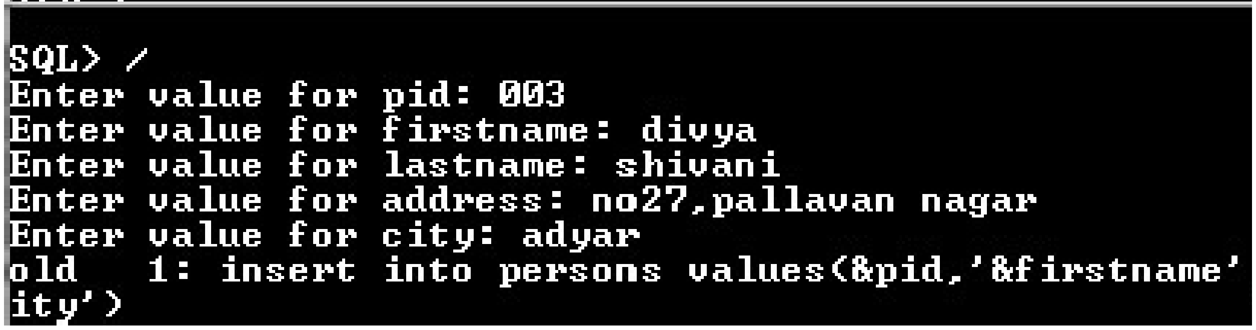
```
SQL> insert into persons values (001,'nelson','raj','
');

1 row created.

SQL> /
```

Insert more than a record into persons table using a single insert command.

Example: SQL> INSERT INTO persons VALUES(&pid, '&firstname', '&lastname', '&address', '&city');



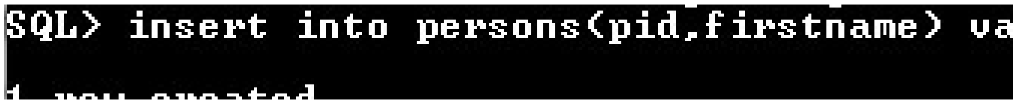
```
SP2-0042: unknown command "poonamalle" - rest of line
SQL> insert into persons values(&pid,'&firstname','&l
);
Enter value for pid: 002
Enter value for firstname: ram
Enter value for lastname: kumar
Enter value for address: no26,raja nagar
Enter value for city: avadi
old   1: insert into persons values(&pid,'&firstname'
```



```
SQL> /
Enter value for pid: 003
Enter value for firstname: divya
Enter value for lastname: shivani
Enter value for address: no27,pallavan nagar
Enter value for city: adyar
old   1: insert into persons values(&pid,'&firstname'
ity')
```

Skipping the fields while inserting:

Example: SQL> INSERT INTO persons(pid, firstname) VALUES(500, 'prabhu');

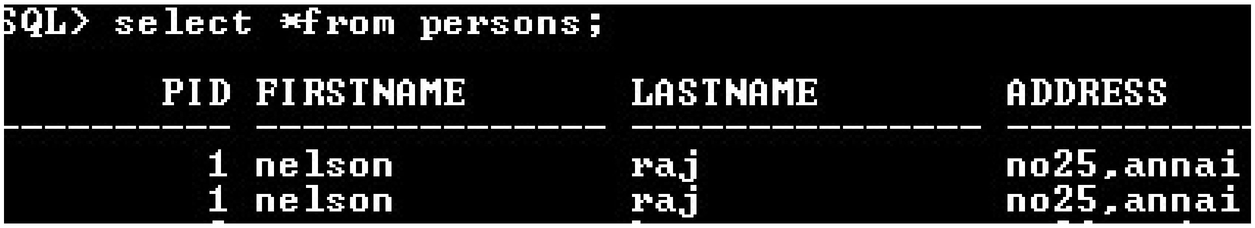


```
SQL> insert into persons(pid,firstname) va

1 row created
```

SELECT COMMAND

It is used to retrieve information from the table. It is generally referred to as querying the table. We can either display all columns in a table or only specify column from the table.
Syntax: SQL> Select * from tablename; // This query selects all rows from the table.

Example: SQL>Select * from persons;



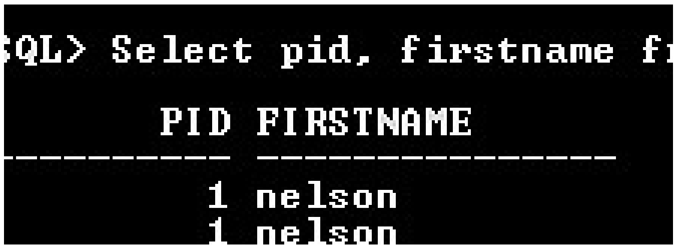
```
SQL> select *from persons;

       PID FIRSTNAME       LASTNAME        ADDRESS
---------- --------------- --------------- ----------
         1 nelson          raj             no25,annai
         1 nelson          raj             no25,annai
```

THE RETRIEVAL OF SPECIFIC COLUMNS FROM A TABLE:

It retrieves the specified columns from the table

Syntax: SQL> SELECT column_name1, ..., column_nameN FROM table_name;

Example: SQL> SELECT pid, firstname FROM persons;



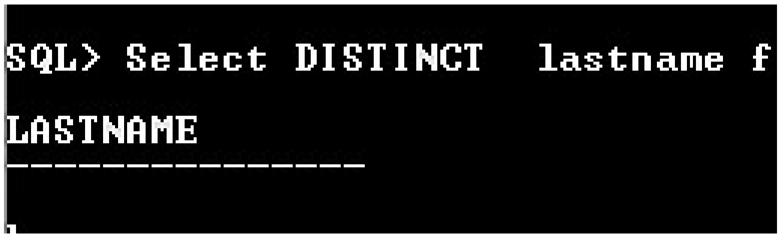
```
SQL> Select pid, firstname fr

       PID FIRSTNAME
---------- ---------------
         1 nelson
         1 nelson
```

Elimination of duplicates from the select clause:

It prevents retrieving the duplicated values. Distinct keyword is to be used.

Syntax: SQL> SELECT DISTINCT col1, col2 FROM table_name;

Example: SQL> SELECT DISTINCT lastname FROM persons;



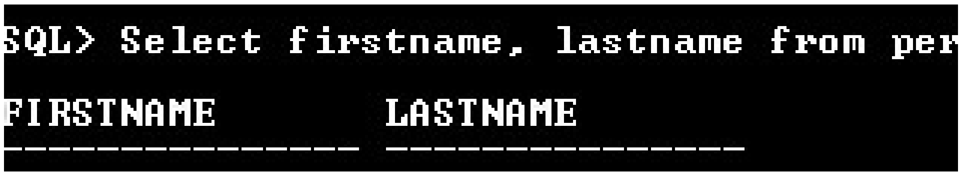
```
SQL> Select DISTINCT  lastname f

LASTNAME
---------------
```

SELECT COMMAND WITH WHERE CLAUSE:

To select specific rows from a table we include 'where' clause in the select command. It can appear only after the 'from' clause.

Syntax: SQL> SELECT column_name1, column_name2, ..., column_nameN FROM table_name WHERE condition;

Example: SQL> SELECT firstname, lastname FROM persons WHERE pid > 2;

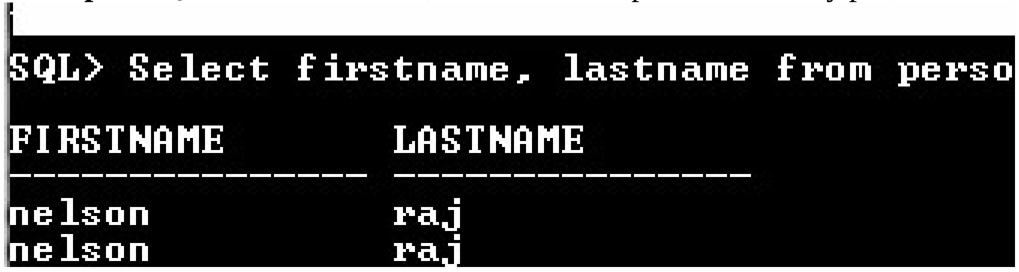


```
SQL> Select firstname, lastname from per

FIRSTNAME       LASTNAME
--------------- ---------------
```

Select command with order by clause:

Syntax: SQL> SELECT column_name1, column_name2, ..., column_namen FROM table_name WHERE condition ORDER BY column_name;

Example: SQL>Select firstname, lastname from persons order by pid;



```
SQL> Select firstname, lastname from perso

FIRSTNAME       LASTNAME
--------------- ---------------
nelson          raj
nelson          raj
```

<u>Select command to create a table:</u>

Syntax: SQL> CREATE TABLE tablename AS SELECT * FROM existing_tablename;

Example: SQL> CREATE TABLE persons1 AS SELECT * FROM persons;
Table created

<u>SELECT COMMAND TO INSERT RECORDS:</u>

Syntax: SQL> INSERT INTO persons1 (SELECT * FROM persons);

Example: SQL> INSERT INTO persons1 (SELECT pid, firstname, lastname FROM persons WHERE city = 'Chennai');

```
PID    FIRSTNAME          LASTNAME     ADDRESS              CITY          PHONENO
-----------------------------------------                   ------------  ----------------
001      nelson         raj           no25,annai street     Chennai
100      niranjan       kumar         10/25 krishna  street Mumbai        999999999
102      arjun          kumar         30 sundaram street    coimbatore
300      gugan          chand         5/10 mettu street     Coimbatore
500      prabhu
```

<u>SELECT COMMAND USING IN KEYWORD:</u>

Syntax: SQL> SELECT column_name1, column_name2, ..., column_n FROM table_name WHERE column_name IN (value1, value2, ...);

Example: SQL> SELECT * FROM persons WHERE pid IN (100, 500);
(OR)
SQL> SELECT * FROM persons WHERE (pid = 100 OR pid = 500);

```
PID    FIRSTNAME          LASTNAME      ADDRESS             CITY          PHONENO
-----------------------------------------------------------               ----------------
100      niranjan       kumar         10/25 krishna  street Mumbai        999999999
500      prabhu
```

SELECT COMMAND USING BETWEEN KEYWORD:

Syntax: SQL> SELECT column_name1, column_name2, ..., column_n FROM table_name WHERE column_name BETWEEN value1 AND value2;

Example: SQL>Select * from persons where pid between 100 and 500;

```
PID    FIRSTNAME          LASTNAME   ADDRESS              CITY          PHONENO
-----------------------------------------------------                   ----------------
100      niranjan           kumar          10/25 krishna  street        Mumbai        999999999
500      prabhu
```

SELECT COMMAND USING PATTERN:

Syntax: SQL> SELECT column_name1, column_name2, ..., column_n FROM table_name WHERE column_name LIKE 'pattern';

Example: SQL>Select * from persons where firstname like ‘nir_n%’;

```
PID    FIRSTNAME          LASTNAME   ADDRESS              CITY          PHONENO
------------ ---------------------- ------------------------- ----------------------      ------------      --------------------
100             niranjan          kumar          10/25 krishna  street    Mumbai        999999999
```

RENAMING THE FIELDNAME AT THE TIME OF DISPLAY USING SELECT STATEMENT:

Syntax: SQL> SELECT old_column_name AS new_column_name FROM table_name WHERE condition;

Example: SQL> SELECT pid AS personid FROM persons;

SELECT COMMAND TO RETRIEVE NULL VALUES:

Syntax: SQL> SELECT column_name FROM table_name WHERE column_name IS NULL;

Example: SQL> SELECT * FROM persons WHERE lastname IS NULL;

```
PID    FIRSTNAME       LASTNAME  ADDRESS                 CITY          PHONENO
------------------------------------------------------------------      ----------        ----------------
500             prabhu
```

UPDATE COMMAND:

Syntax: SQL> UPDATE table_name SET column_name = value [, column_name = value]... [WHERE condition];

Example: SQL> UPDATE persons SET pid = 5 WHERE firstname = 'prabhu';

Table updated.

DELETE COMMAND

Syntax: SQL> DELETE FROM table_name WHERE condition;

Example: SQL> DELETE FROM persons WHERE pid = 500;

1 row deleted.

RESULT:

Thus, the database was successfully created, and data was inserted, deleted, modified, updated, and altered. Records were also retrieved based on specific conditions.

EX.NO: 2
DATE:

DATA CONTROL LANGUAGE COMMANDS AND TRANSACTION CONTROL COMMANDS

AIM:

To create a database using Data Control Commands and Transaction Control Commands to manage transactions in the database.

DESCRIPTION:

Transaction Control statements

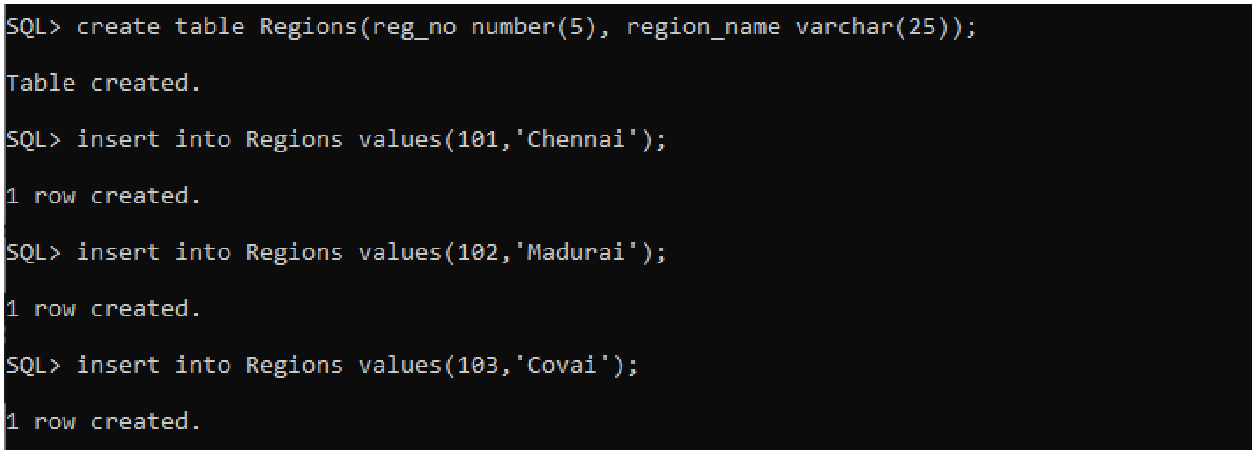
Transaction Control Language (TCL) commands are used to manage transactions in the database. These are used to manage the changes made by DML-statements. It also allows statements to be grouped together into logical transactions.

Examples of TCL:

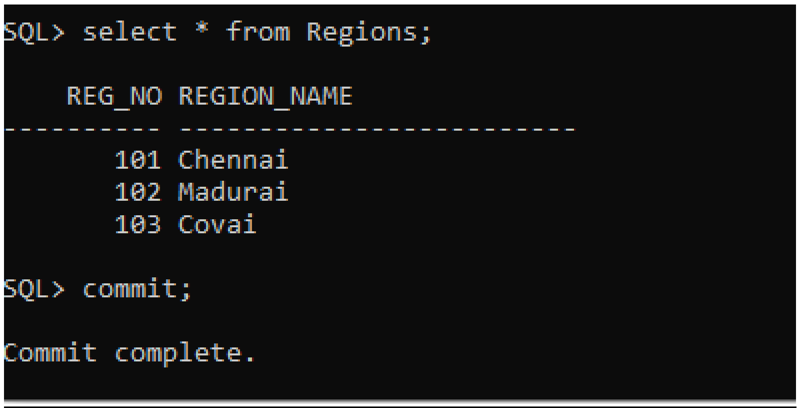
(i) Commit
(ii) Rollback
(iii) Savepoint

(i) Commit: Commit command saves all the work done.

Syntax: commit;



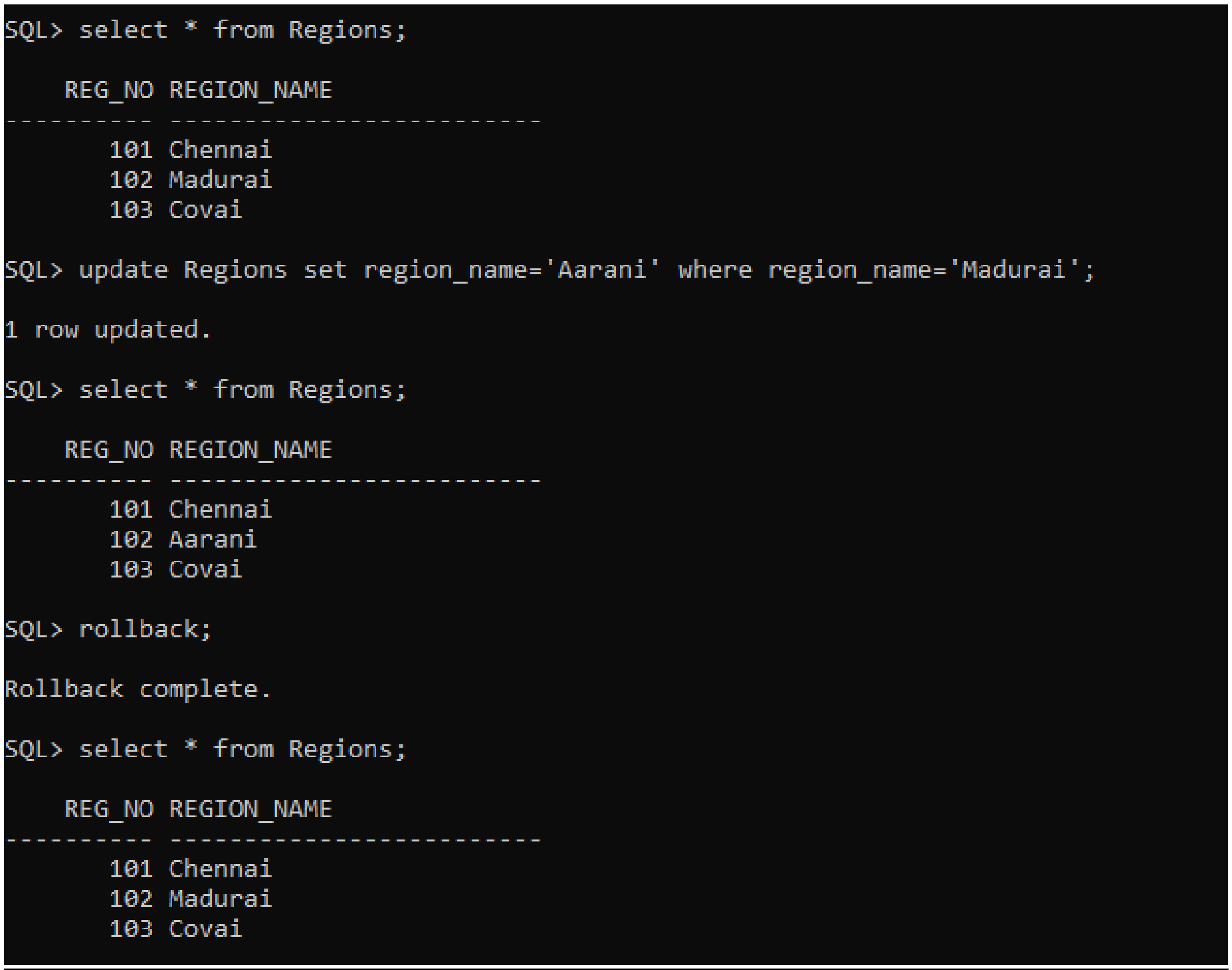
```
SQL> create table Regions(reg_no number(5), region_name varchar(25));

Table created.

SQL> insert into Regions values(101,'Chennai');

1 row created.

SQL> insert into Regions values(102,'Madurai');

1 row created.

SQL> insert into Regions values(103,'Covai');

1 row created.
```



```
SQL> select * from Regions;

    REG_NO REGION_NAME
---------- -------------------------
       101 Chennai
       102 Madurai
       103 Covai

SQL> commit;

Commit complete.
```

(ii) Rollback: Rollback Command restores database to original since the last Commit.

Syntax: ROLLBACK TO SAVEPOINT <savepoint_name>;



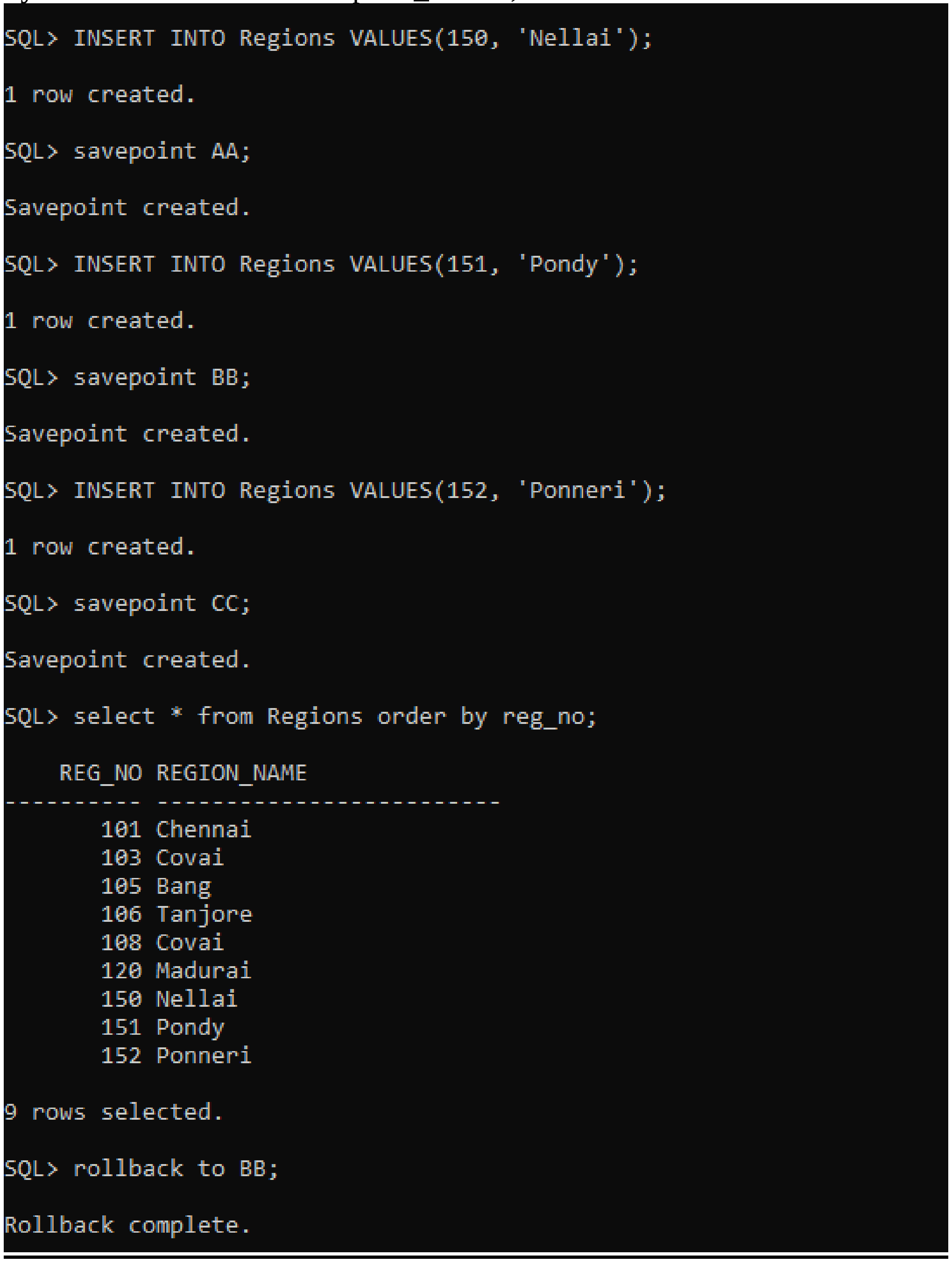
```
SQL> select * from Regions;

    REG_NO REGION_NAME
---------- -------------------------
       101 Chennai
       102 Madurai
       103 Covai

SQL> update Regions set region_name='Aarani' where region_name='Madurai';

1 row updated.

SQL> select * from Regions;

    REG_NO REGION_NAME
---------- -------------------------
       101 Chennai
       102 Aarani
       103 Covai

SQL> rollback;

Rollback complete.

SQL> select * from Regions;

    REG_NO REGION_NAME
---------- -------------------------
       101 Chennai
       102 Madurai
       103 Covai
```

(iii) Savepoint:

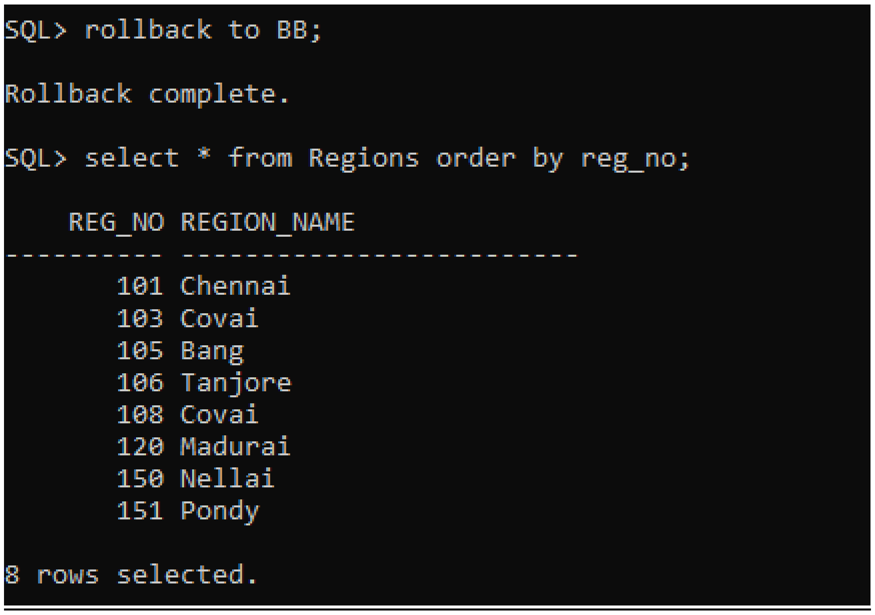
SAVEPOINT command is used to temporarily save a transaction so that you can rollback to that point whenever required.

In short, using this command we can name the different states of our data in any table and then rollback to that state using the ROLLBACK command whenever required.

Syntax: SAVEPOINT <savepoint_name>;



```
SQL> INSERT INTO Regions VALUES(150, 'Nellai');

1 row created.

SQL> savepoint AA;

Savepoint created.

SQL> INSERT INTO Regions VALUES(151, 'Pondy');

1 row created.

SQL> savepoint BB;

Savepoint created.

SQL> INSERT INTO Regions VALUES(152, 'Ponneri');

1 row created.

SQL> savepoint CC;

Savepoint created.

SQL> select * from Regions order by reg_no;

    REG_NO REGION_NAME
---------- -------------------------
       101 Chennai
       103 Covai
       105 Bang
       106 Tanjore
       108 Covai
       120 Madurai
       150 Nellai
       151 Pondy
       152 Ponneri

9 rows selected.

SQL> rollback to BB;

Rollback complete.
```



```
SQL> rollback to BB;

Rollback complete.

SQL> select * from Regions order by reg_no;

    REG_NO REGION_NAME
---------- ------------------------
       101 Chennai
       103 Covai
       105 Bang
       106 Tanjore
       108 Covai
       120 Madurai
       150 Nellai
       151 Pondy

8 rows selected.
```

Data Control Language

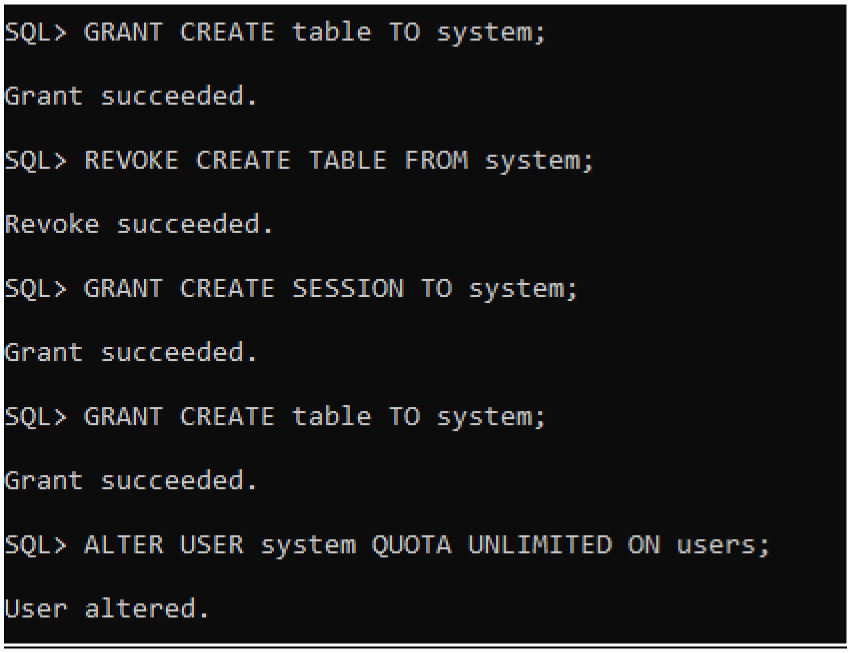
Data Control Language (DCL) is used to control privileges in Database. To perform any operation in the database, such as for creating tables, sequences or views, a user needs privileges. Privileges are of two types,

- **System:** This includes permissions for creating session, table, etc. and all types of other system privileges.
- **Object:** This includes permissions for any command or query to perform any operation on the database tables.

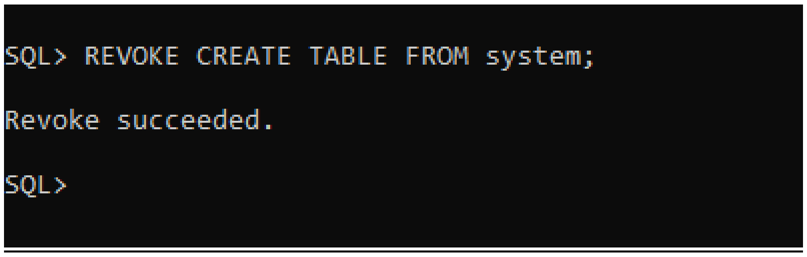
In DCL we have two commands,

- **GRANT:** Used to provide any user access privileges or other privileges for the database.

- **REVOKE:** Used to take back permissions from any user.



```
SQL> GRANT CREATE table TO system;

Grant succeeded.

SQL> REVOKE CREATE TABLE FROM system;

Revoke succeeded.

SQL> GRANT CREATE SESSION TO system;

Grant succeeded.

SQL> GRANT CREATE table TO system;

Grant succeeded.

SQL> ALTER USER system QUOTA UNLIMITED ON users;

User altered.
```



```
SQL> REVOKE CREATE TABLE FROM system;

Revoke succeeded.

SQL>
```

RESULT:

Thus, the Data Control Language (DCL) and Transaction Control Language (TCL) commands were executed successfully.

EX. NO: 3
Date:

SQL FUNCTIONS

AIM:

To study the various SQL functions and their operations on the database.

DESCRIPTION:

What are functions?

Functions are methods used to perform data operations. SQL has many in-built functions used to perform string concatenations, mathematical calculations etc.

SQL functions are categorized into the following two categories:

1. Aggregate Functions
2. Scalar Functions

Let us look into each one of them, one by one.

AGGREGATE SQL FUNCTIONS

The Aggregate Functions in SQL perform calculations on a group of values and then return a single value. Following is a few of the most commonly used Aggregate Functions:

Function	Description
SUM()	Used to return the sum of a group of values.
COUNT()	Returns the number of rows either based on a condition, or without a condition.
AVG()	Used to calculate the average value of a numeric column.
MIN()	This function returns the minimum value of a column.
MAX()	Returns a maximum value of a column.
FIRST()	Used to return the first value of the column.
LAST()	This function returns the last value of the column.

SCALAR SQL FUNCTIONS

The Scalar Functions in SQL are used to return a single value from the given input value. Following is a few of the most commonly used Aggregate Functions:

Function	Description
LCASE()	Used to convert string column values to lowercase
UCASE()	This function is used to convert a string column values to Uppercase.

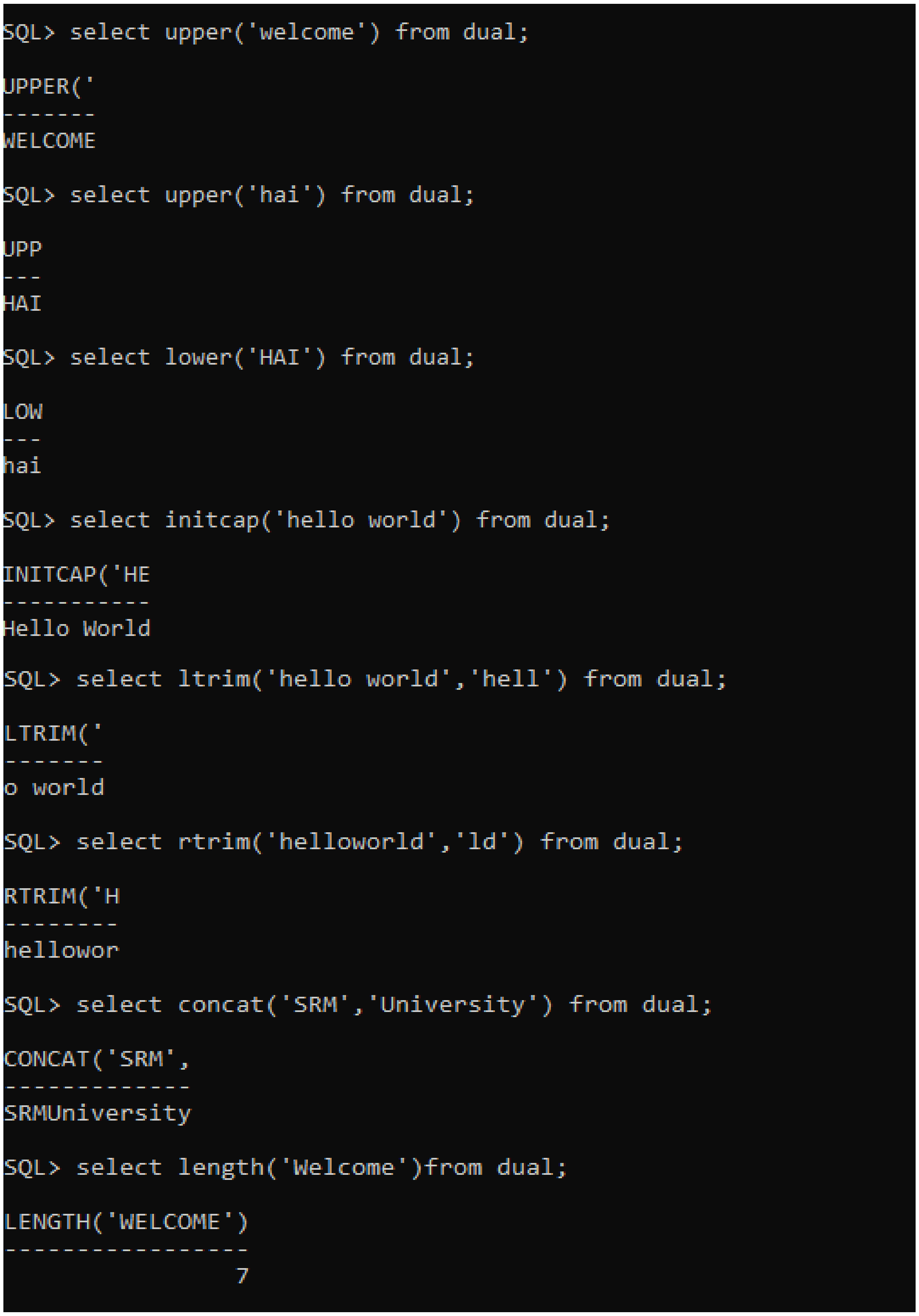
LEN()	Returns the length of the text values in the column.
MID()	Extracts substrings in SQL from column values having String data type.
ROUND()	Rounds off a numeric value to the nearest integer.
NOW()	This function is used to return the current system date and time.
FORMAT()	Used to format how a field must be displayed.

EXAMPLE:

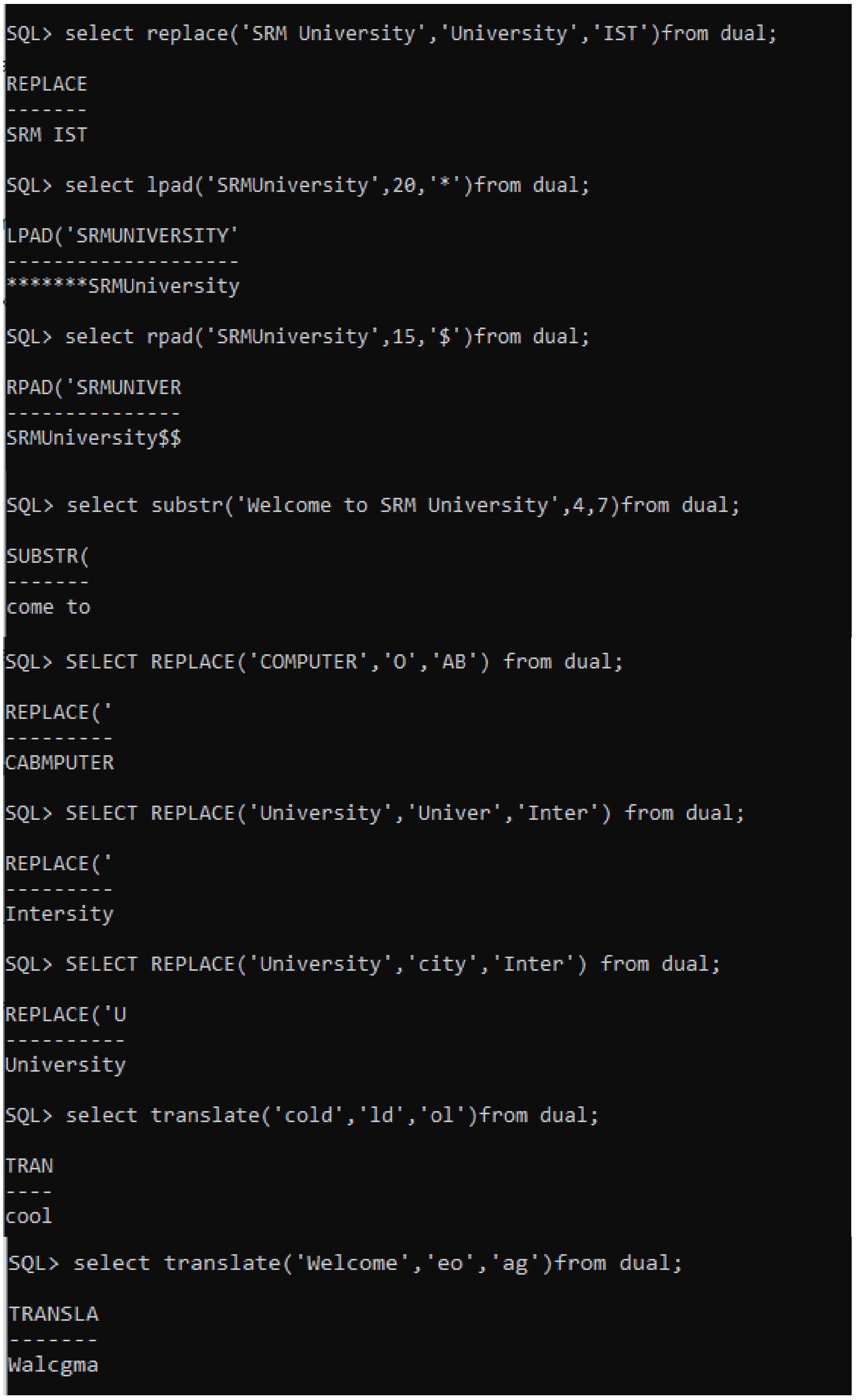
CHARACTER/STRING FUNCTION:

```
SQL> select upper('welcome') from dual;

SQL> select upper('hai') from dual;

SQL> select lower('HAI') from dual;

SQL> select initcap(‘hello world') from dual;

SQL> select ltrim('hello world',’hell’) from dual;

SQL> select rtrim(''hello world',’ld’')from dual;

SQL> select concat('SRM',' University')from dual;

SQL> select length('Welcome’) from dual;

SQL> select replace('SRM University', 'University','IST')from dual;

SQL> select lpad('SRM University',20,'*')from dual;

SQL> select rpad('SRM University',15,'$')from dual;

SQL> select substr('Welcome to SRM University', 4,7)from dual;

SQL> select replace('COMPUTER','O','AB')from dual;

SQL> select replace('University','city’,'Inter')from dual;

SQL> select translate('cold','ld','ol')from dual;
```

OUTPUT:



```
SQL> select upper('welcome') from dual;

UPPER('
-------
WELCOME

SQL> select upper('hai') from dual;

UPP
---
HAI

SQL> select lower('HAI') from dual;

LOW
---
hai

SQL> select initcap('hello world') from dual;

INITCAP('HE
-----------
Hello World

SQL> select ltrim('hello world','hell') from dual;

LTRIM('
-------
o world

SQL> select rtrim('helloworld','ld') from dual;

RTRIM('H
--------
hellowor

SQL> select concat('SRM','University') from dual;

CONCAT('SRM',
-------------
SRMUniversity

SQL> select length('Welcome')from dual;

LENGTH('WELCOME')
-----------------
                7
```



```
SQL> select replace('SRM University','University','IST')from dual;

REPLACE
-------
SRM IST

SQL> select lpad('SRMUniversity',20,'*')from dual;

LPAD('SRMUNIVERSITY'
--------------------
*******SRMUniversity

SQL> select rpad('SRMUniversity',15,'$')from dual;

RPAD('SRMUNIVER
---------------
SRMUniversity$$


SQL> select substr('Welcome to SRM University',4,7)from dual;

SUBSTR(
-------
come to

SQL> SELECT REPLACE('COMPUTER','O','AB') from dual;

REPLACE('
---------
CABMPUTER

SQL> SELECT REPLACE('University','Univer','Inter') from dual;

REPLACE('
---------
Intersity

SQL> SELECT REPLACE('University','city','Inter') from dual;

REPLACE('U
----------
University

SQL> select translate('cold','ld','ol')from dual;

TRAN
----
cool

SQL> select translate('Welcome','eo','ag')from dual;

TRANSLA
-------
Walcgma
```

DATE & TIME FUNCTION

SQL> select sysdate from dual;

SQL> select round(sysdate)from dual;

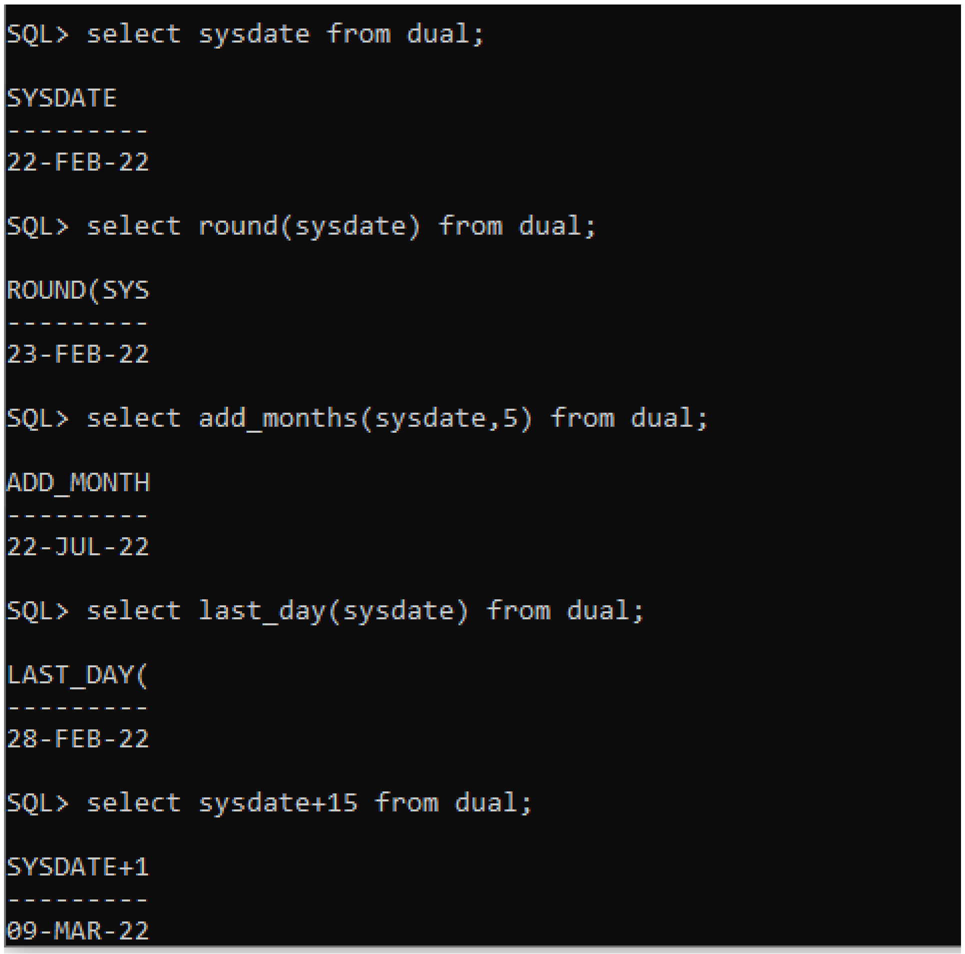
SQL> select add_months(sysdate,3)from dual;

SQL> select last_day(sysdate)from dual;

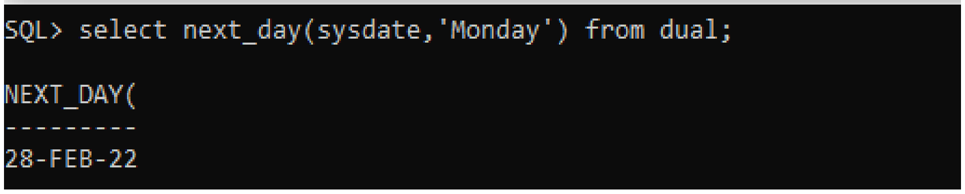
SQL> select sysdate+20 from dual;

SQL> select next_day(sysdate,'tuesday')from dual;

OUTPUT:



```
SQL> select sysdate from dual;

SYSDATE
---------
22-FEB-22

SQL> select round(sysdate) from dual;

ROUND(SYS
---------
23-FEB-22

SQL> select add_months(sysdate,5) from dual;

ADD_MONTH
---------
22-JUL-22

SQL> select last_day(sysdate) from dual;

LAST_DAY(
---------
28-FEB-22

SQL> select sysdate+15 from dual;

SYSDATE+1
---------
09-MAR-22
```



```
SQL> select next_day(sysdate,'Monday') from dual;

NEXT_DAY(
---------
28-FEB-22
```

<u>NUMERIC FUNCTION</u>

SQL> select round(15.6789)from dual;

SQL> select ceil(23.20)from dual;

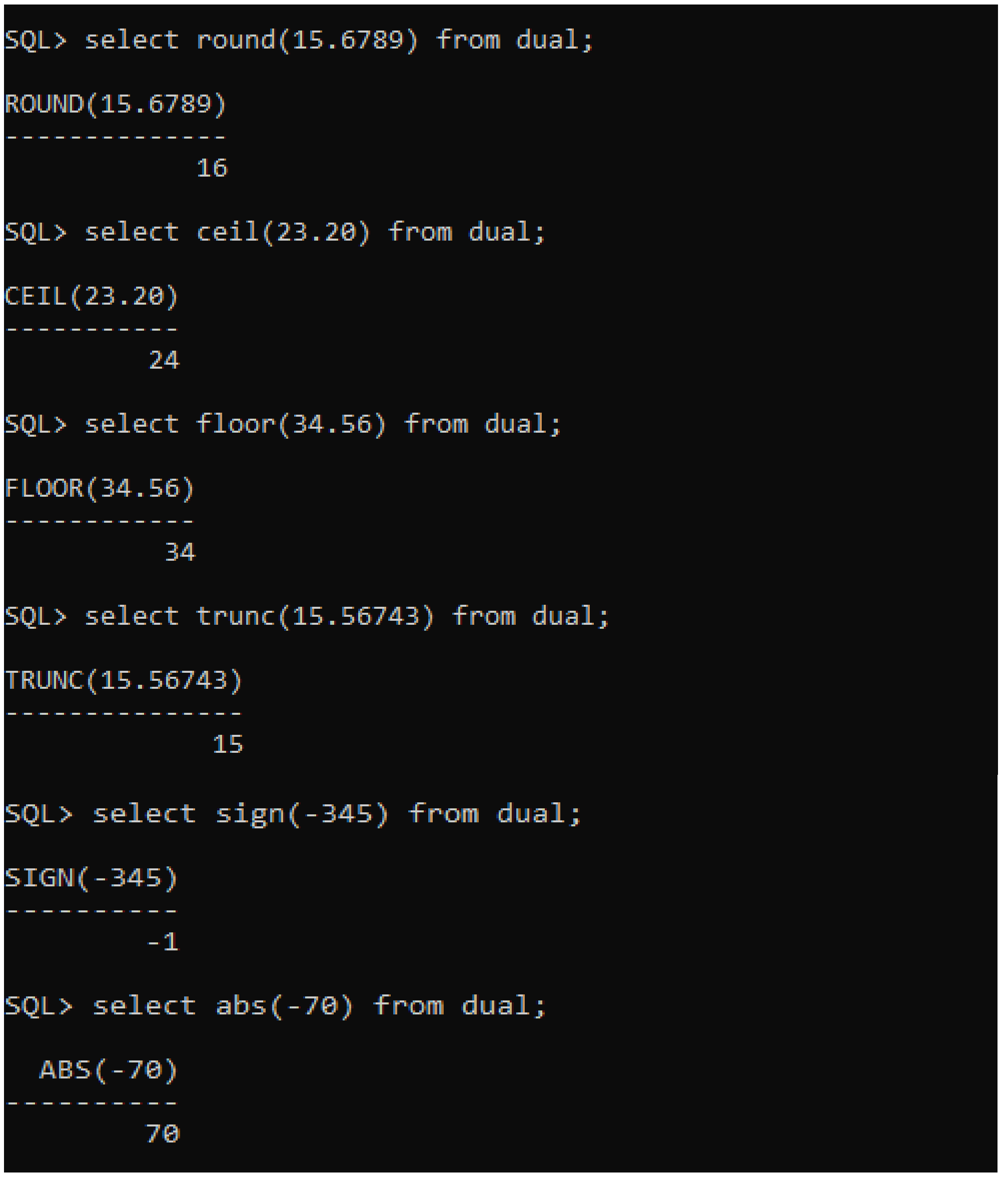
SQL> select floor(34.56)from dual;

SQL> select trunc(15.56743)from dual;

SQL> select sign(-345)from dual;

SQL> select abs(-70)from dual;

OUTPUT:



```
SQL> select round(15.6789) from dual;

ROUND(15.6789)
--------------
            16

SQL> select ceil(23.20) from dual;

CEIL(23.20)
-----------
         24

SQL> select floor(34.56) from dual;

FLOOR(34.56)
------------
          34

SQL> select trunc(15.56743) from dual;

TRUNC(15.56743)
---------------
             15

SQL> select sign(-345) from dual;

SIGN(-345)
----------
        -1

SQL> select abs(-70) from dual;

  ABS(-70)
----------
        70
```

MATH FUNCTION:

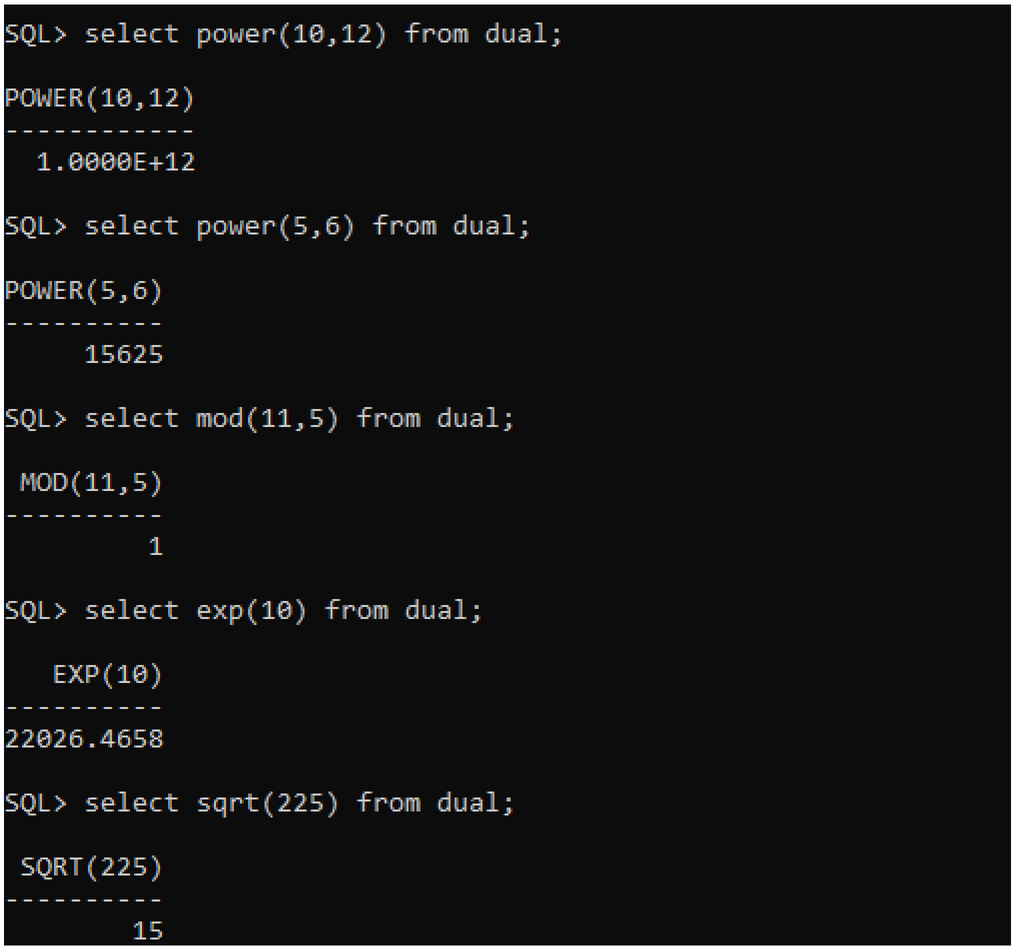
SQL> select power(10,12) from dual;

SQL> select power(5,6) from dual;

SQL> select mod(11,5) from dual;

SQL> select exp(10) from dual;

SQL> select sqrt(225) from dual;



```
SQL> select power(10,12) from dual;

POWER(10,12)
------------
  1.0000E+12

SQL> select power(5,6) from dual;

POWER(5,6)
----------
     15625

SQL> select mod(11,5) from dual;

 MOD(11,5)
----------
         1

SQL> select exp(10) from dual;

   EXP(10)
----------
22026.4658

SQL> select sqrt(225) from dual;

 SQRT(225)
----------
        15
```

RESULT:

Thus, the SQL Functions have been executed successfully.

Ex. No: 4
Date:

CONSTRUCT A E-R MODEL FOR THE APPLICATION TO BE CONSTRUCTED TO A DATABASE

AIM:

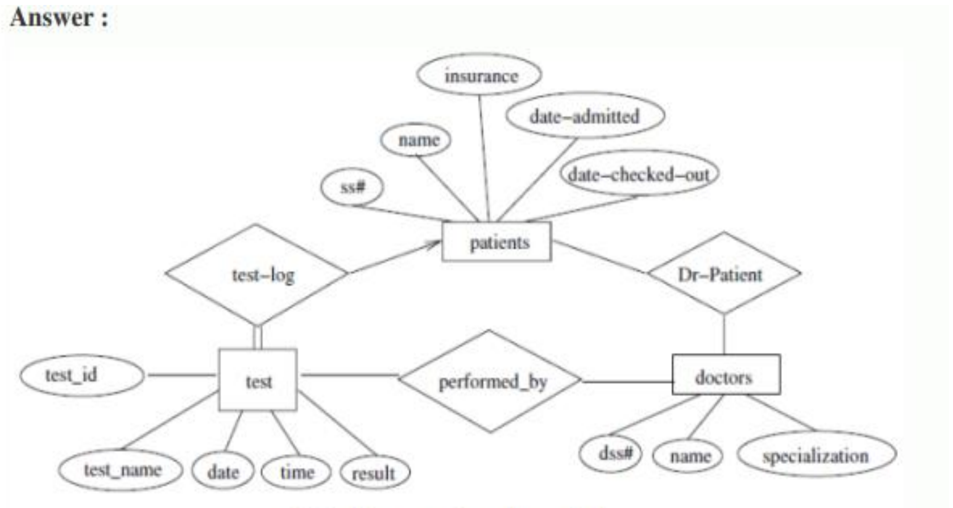
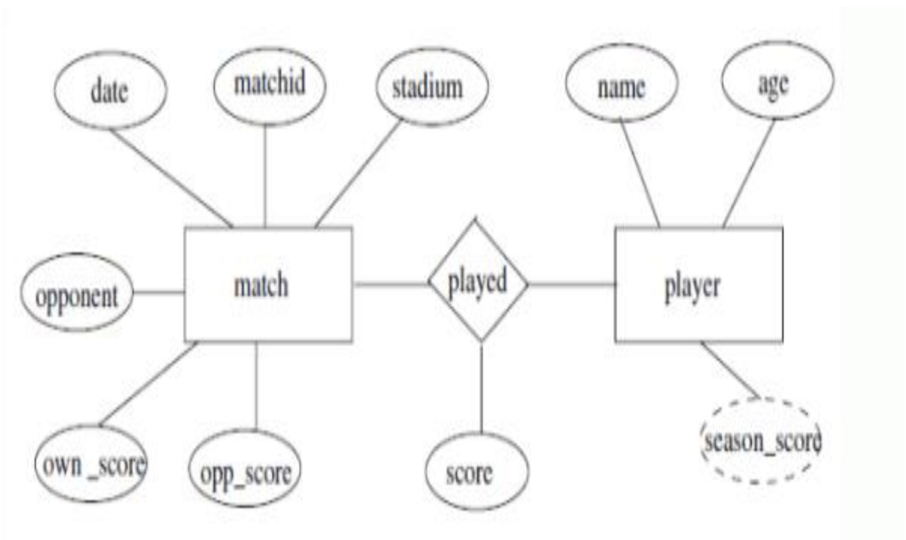
To construct an ER model for the application to be developed into a database.

HOW TO DRAW AN ENTITY RELATIONSHIP DIAGRAM?

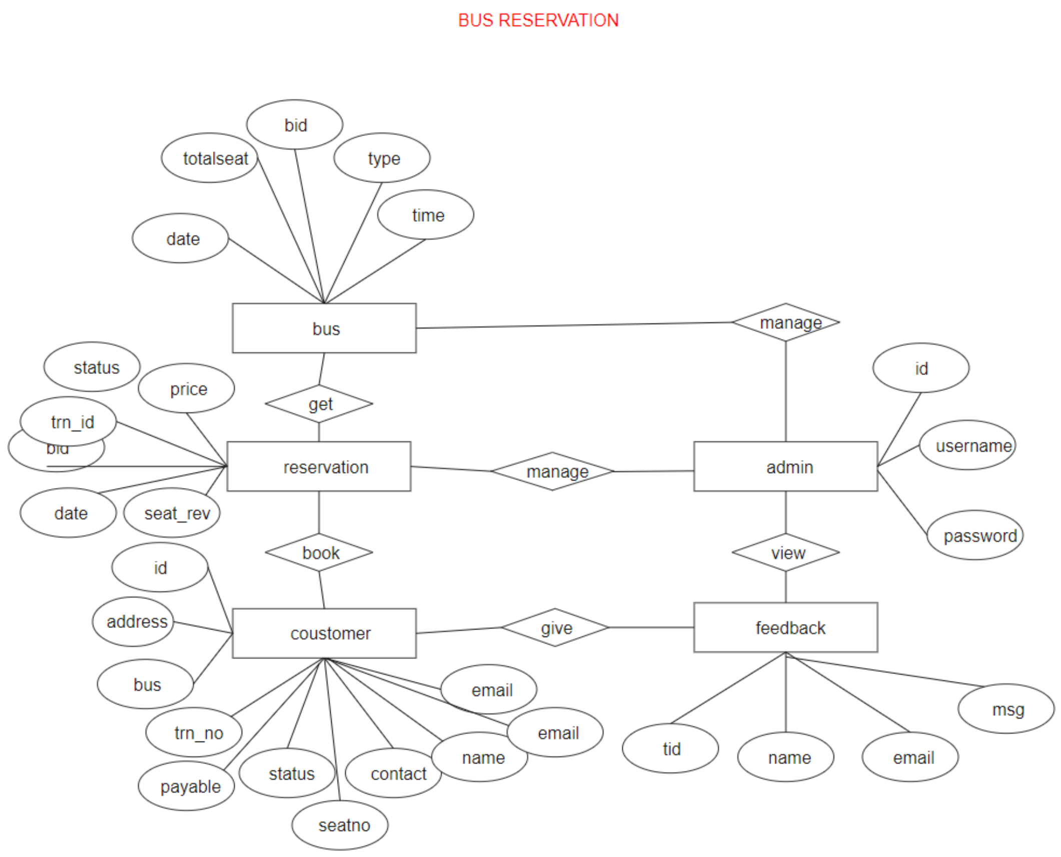
1. Determine the Entities in Your ERD.
2. Add Attributes to Each Entity.
3. Define the Relationships Between Entities.
4. Add Cardinality to Every Relationship in your ER Diagram.
5. Finish and Save Your ERD.

Few Online Entity Relationship Diagram Tools

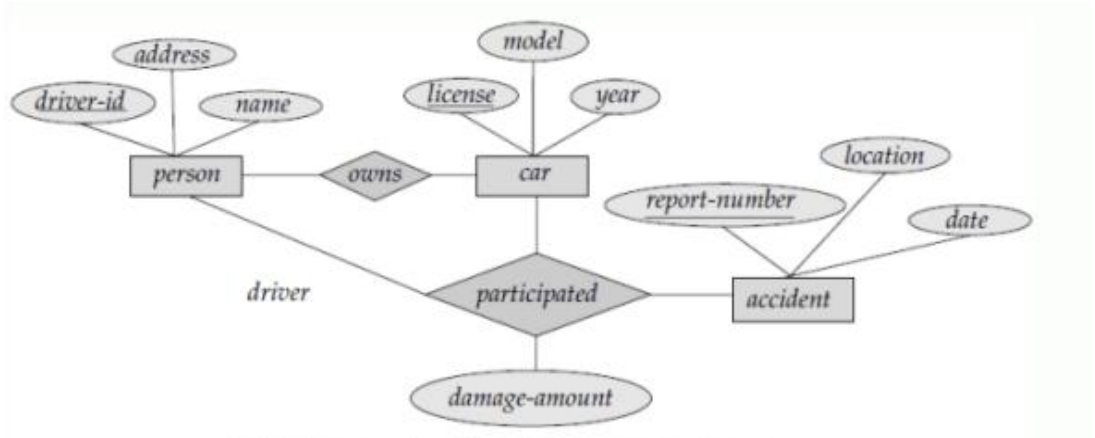
1. Lucidchart
2. Creatly
3. Draw.io
4. Vertabelo
5. ERDPlus
6. Visual Paradigm Online
7. Microsoft Visio
8. Gliffy
9. SqlDBM ER Diagram Online Tool
10. ER Draw Max

S.No.	Problem Statement
1	(a) Construct an E-R diagram for a hospital with a set of patients and a set of medical doctors. Associate with each patient a log of the various tests and examinations conducted. **ER Diagram:**  Answer : insurance date–admitted name date–checked–out ss# patients test–log Dr–Patient test_id test performed_by doctors test_name date time result dss# name specialization E-R diagram for a hospital. (b) Construct appropriate tables for the above ER Diagram: **Hospital tables:** patients (patient-id, name, insurance, date-admitted, date-checked-out) doctors (doctor-id, name, specialization) test (testid, testname, date, time, result) doctor-patient (patient-id, doctor-id) test-log (testid, patient-id) performed-by (testid, doctor-id)
2	Design an E-R diagram for keeping track of the exploits of your favorite sports team. You should store the matches played, the scores in each match, the players in each match and individual player statistics for each match. Summary statistics should be modeled as derived attributes **ER Diagram:**  date matchid stadium name age opponent match played player own _score opp_score score season_score E-R diagram for favourite team statistics.

3 Design an E-R diagram for bus reservation system.



4 (a) Construct an E-R diagram for a car-insurance company whose customers own one or more cars each. Each car has associated with it zero to any number of recorded accidents.



E-R diagram for a Car-insurance company.

(b) Construct appropriate tables for the above ER Diagram?

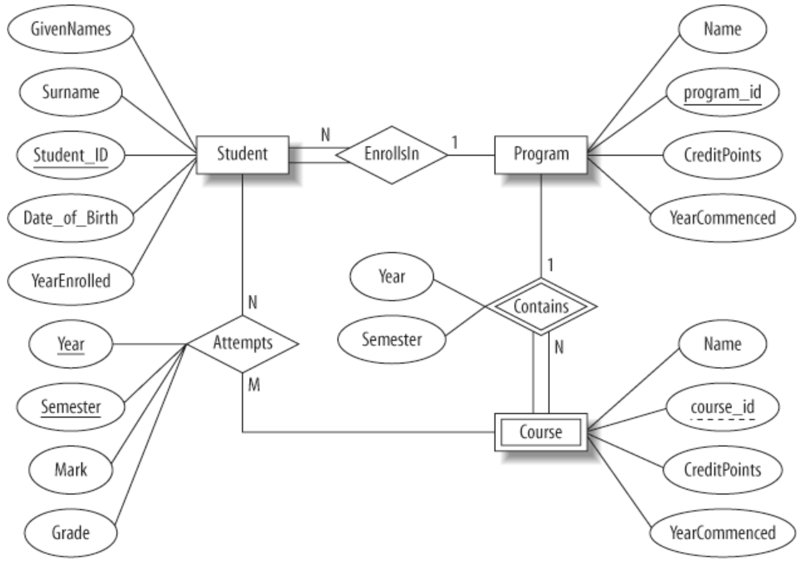
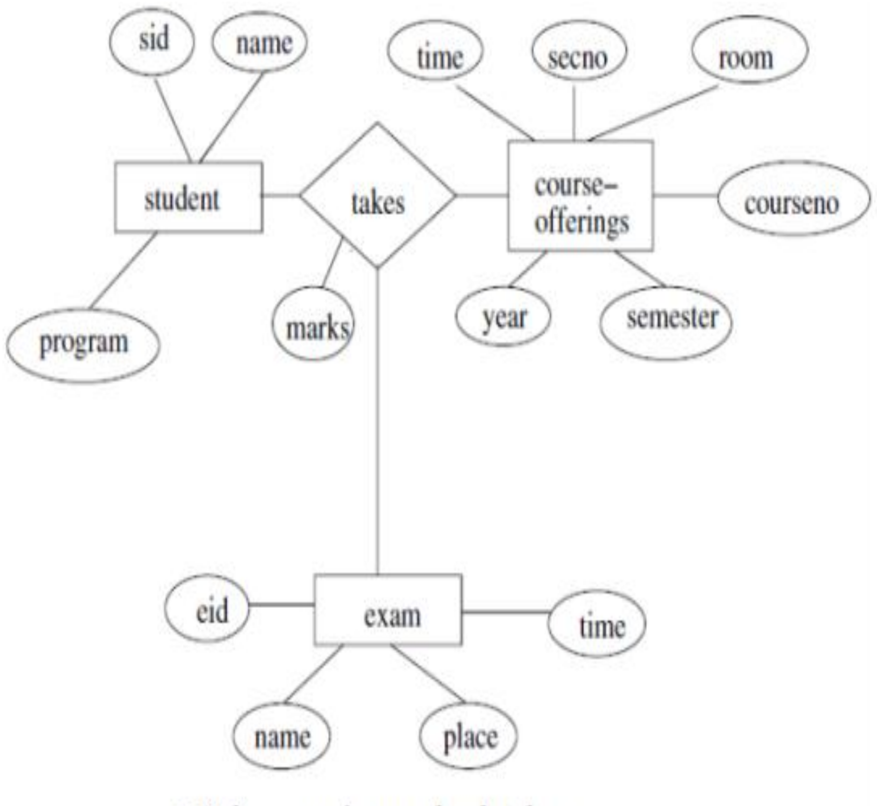
Car insurance tables:

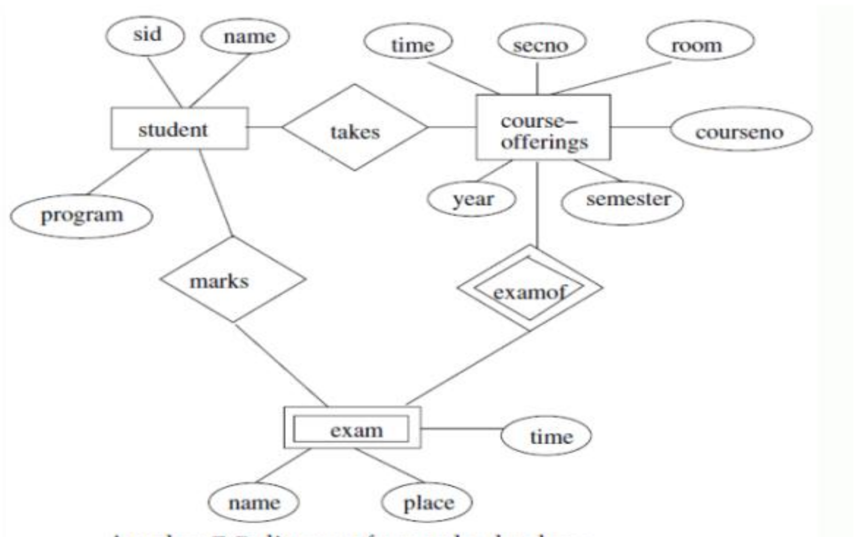
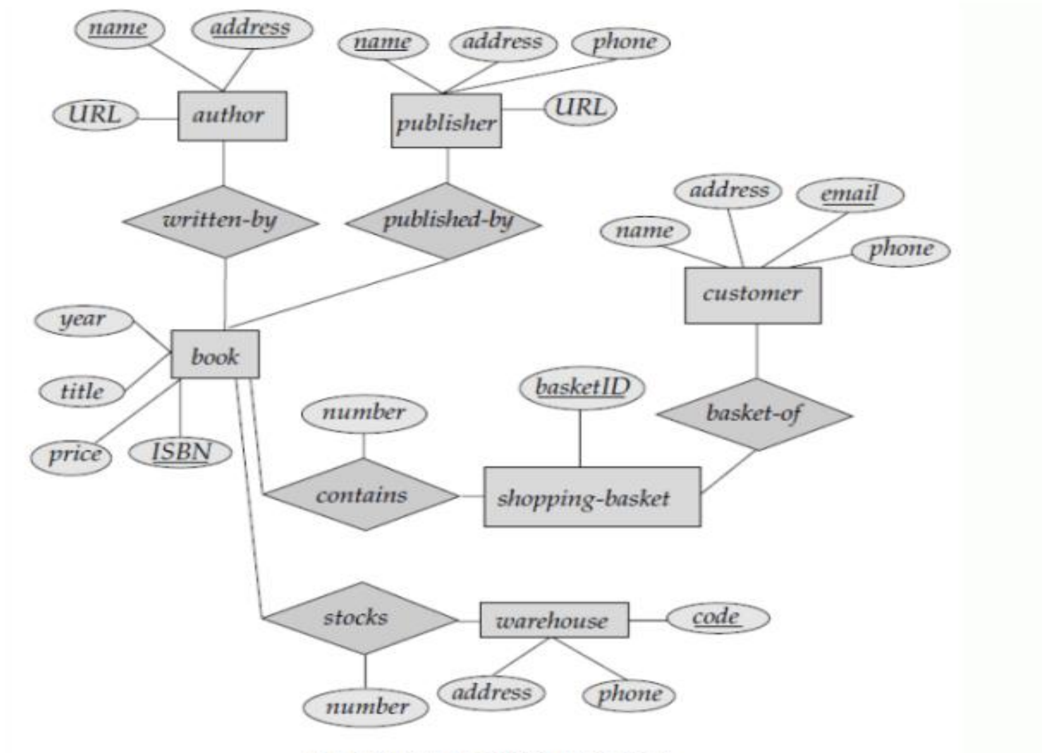
person (<u>driver-id</u>, name, address)

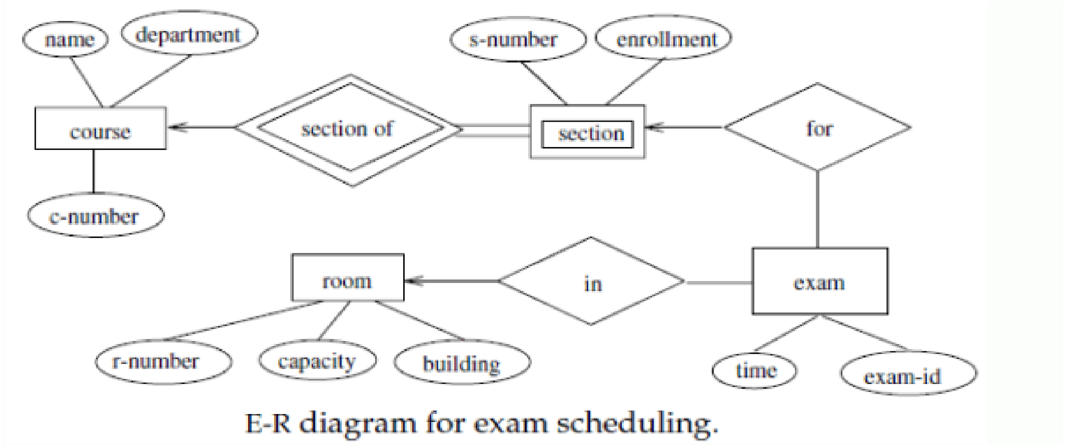
car (<u>license</u>, year, model)

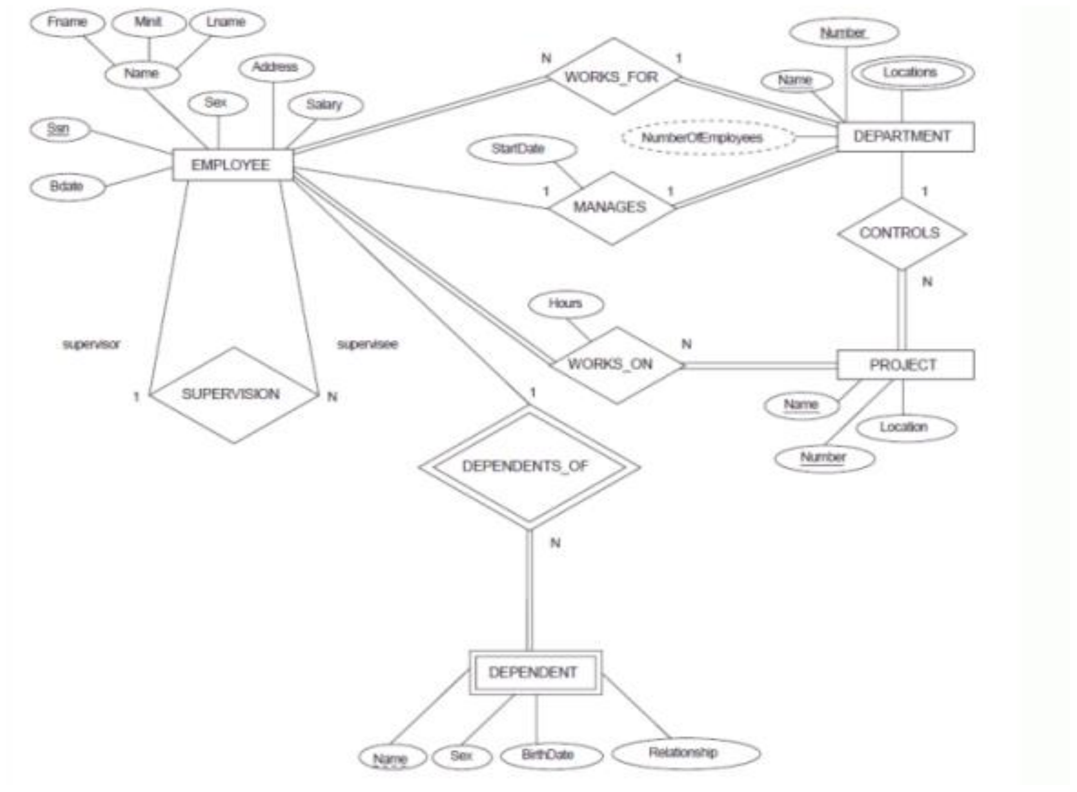
accident (<u>report-number</u>, date, location)

participated(<u>driver-id</u>, <u>license</u>, <u>report-number</u>, damage-amount)

5	Design an E-R diagram for University Database  GivenNames Surname Student_ID Date_of_Birth YearEnrolled Student N EnrollsIn 1 Program Name program_id CreditPoints YearCommenced Year Semester 1 Contains N N Attempts M Year Semester Mark Grade Course Name course_id CreditPoints YearCommenced
6	Consider a database used to record the marks that students get in different exams of different course offerings. a) Construct an E-R diagram that models exams as entities, and uses a ternary relationship, for the above database.  sid name student program takes marks time secno room course-offerings courseno year semester eid exam time name place E-R diagram for marks database. b) Construct an alternative E-R diagram that uses only a binary relationship between students and course-offerings. Make sure that only one relationship exists between a particular student and course-offering pair, yet you can represent the marks that a student gets in different exams of a course offering.

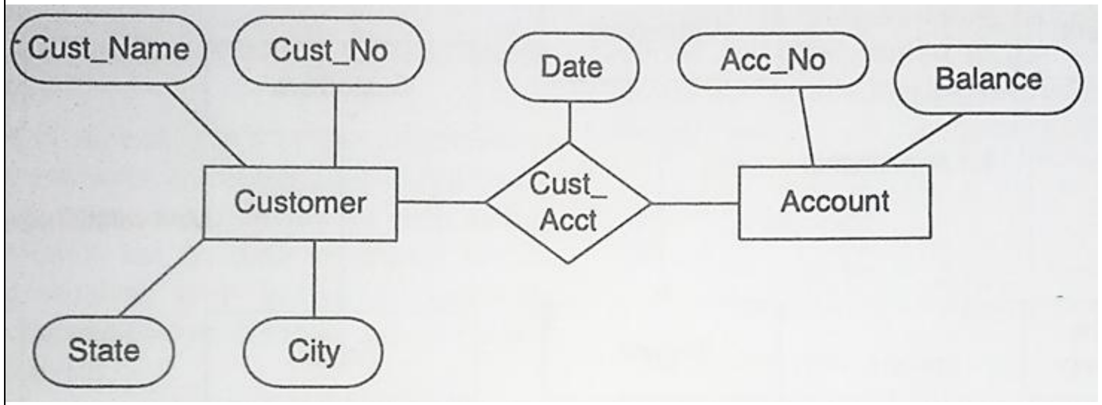
	ER Diagram:  sid name time secno room student takes course–offerings courseno program year semester marks examof exam time name place Another E-R diagram for marks database.
7	Draw the E-R diagram which models an online bookstore.  name address name address phone URL author publisher URL written-by published-by address email name phone customer year book title price ISBN number basketID basket-of contains shopping-basket stocks warehouse code number address phone ER Diagram for Online BookStore
8	Consider a university database for the scheduling of classrooms for -final exams. This database could be modeled as the single entity set exam, with attributes course-name, sectionnumber, room-number, and time. Alternatively, one or more additional entity sets could be defined, along with relationship sets to replace some of the attributes of the exam entity set, as • course with attributes name, department, and c-number • section with attributes s-number and enrollment, and dependent as a weak entity set on

	course • room with attributes r-number, capacity, and building Show an E-R diagram illustrating the use of all three additional entity sets listed.  name department course c-number section of s-number enrollment section for exam in room r-number capacity building time exam-id E-R diagram for exam scheduling.
9	Construct an ER Diagram for Company having following details : • Company organized into DEPARTMENT. Each department has unique name and a particular employee who manages the department. Start date for the manager is recorded. Department may have several locations. • A department controls a number of PROJECT. Projects have a unique name, number and a single location. • Company's EMPLOYEE name, ssno, address, salary, sex and birth date are recorded. An employee is assigned to one department, but may work for several projects (not necessarily controlled by her dept). Number of hours/week an employee works on each project is recorded; The immediate supervisor for the employee. • Employee's DEPENDENT are tracked for health insurance purposes (dependent name, birthdate, relationship to employee).



10

Design an E-R diagram for Customer Account.



Result:

Thus, the ER model for the application was successfully constructed and can be used to design the corresponding database.

EX.NO: 5a
Date:

NESTED QUERIES

AIM:

To perform operations using various SQL nested queries on the database.

DESCRIPTION:

Nested query is one of the most useful functionalities of SQL. Nested queries are useful when we want to write complex queries where one query uses the result from another query. Nested queries will have multiple SELECT statements nested together. A SELECT statement nested within another SELECT statement is called a subquery.

What is a Nested Query in SQL?

A nested query in SQL contains a query inside another query. The result of the inner query will be used by the outer query. For instance, a nested query can have two **SELECT** statements, one on the inner query and the other on the outer query.

What are the Types of Nested Queries in SQL?

Nested queries in SQL can be classified into two different types:

1. Independent Nested Queries
2. Co-related Nested Queries

1. Independent Nested Queries

In independent nested queries, the execution order is from the innermost query to the outer query. An outer query won't be executed until its inner query completes its execution. The result of the inner query is used by the outer query. Operators such as **IN**, **NOT IN**, **ALL**, and **ANY** are used to write independent nested queries.

The **IN** operator checks if a column value in the outer query's result is **present** in the inner query's result. The final result will have rows that satisfy the **IN** condition.

The **NOT IN** operator checks if a column value in the outer query's result is **not present** in the inner query's result. The final result will have rows that satisfy the **NOT IN** condition.

The **ALL** operator compares a value of the outer query's result with **all the values** of the inner query's result and returns the row if it matches all the values.

The **ANY** operator compares a value of the outer query's result with all the inner query's result values and returns the row if there is a match with **any value**.

2. Co-related Nested Queries

In co-related nested queries, the inner query uses the values from the outer query so that the inner query is executed for every row processed by the outer query. The co-related nested queries run slowly because the inner query is executed for every row of the outer query's result.

How to Write Nested Query in SQL?

We can write a nested query in SQL by nesting a **SELECT** statement within another **SELECT** statement. The outer **SELECT** statement uses the result of the inner **SELECT** statement for processing.

The general syntax of nested queries will be:

```
SELECT column_name [, column_name ]
FROM table1 [, table2 ]
WHERE column_name OPERATOR
(   SELECT column_name [, column_name ]
    FROM table1 [, table2 ]
    [WHERE]
)
```

The **SELECT** query inside the brackets (**()**) is the inner query, and the **SELECT** query outside the brackets is the outer query. The result of the inner query is used by the outer query.

EXAMPLE:

TABLE #1 - employeedata

```
SQL> CREATE TABLE employeedata(id NUMBER PRIMARY KEY, name VARCHAR2(25)
NOT NULL, salary NUMBER NOT NULL, role VARCHAR2(15) NOT NULL);

Table created.

SQL> INSERT INTO employeedata VALUES (1, 'Augustine Hammond', 10000, 'Developer');

1 row created.

SQL> INSERT INTO employeedata VALUES (2, 'Perice John', 10000, 'Manager');

1 row created.

SQL> INSERT INTO employeedata VALUES (3, 'Ragu Delafoy', 30000, 'Developer');

1 row created.

SQL> INSERT INTO employeedata VALUES (4, 'Teakwood Saffen', 40000, 'Manager');
```

```
1 row created.

SQL> INSERT INTO employeedata VALUES (5, 'Freddy Malcom', 50000, 'Developer');

1 row created.

SQL> select * from employeedata;
```

OUTPUT:

```
      ID    NAME                SALARY    ROLE
---------- ------------------------ ----------   ---------------
       1 Augustine Hammond     10000         Developer
       2 Perice John           10000         Manager
       3 Ragu Delafoy          30000         Developer
       4 Teakwood Saffen       40000         Manager
       5 Freddy Malcom         50000         Developer
```

TABLE #2 - awards

```
SQL>CREATE TABLE awards( id NUMBER PRIMARY KEY, employee_id NUMBER NOT
NULL, award_date DATE NOT NULL );
Table created.

SQL> INSERT INTO awards VALUES(1, 1, TO_DATE('2022-04-01', 'YYYY-MM-DD'));
1 row created.

SQL> INSERT INTO awards VALUES(2, 3, TO_DATE('2022-05-01', 'YYYY-MM-DD'));
1 row created.

SQL> select * from awards;
```

OUTPUT:

```
      ID   EMPLOYEE_ID   AWARD_DAT
----------   ------------------      ----------------
       1         1                  01-APR-22
       2         3                  01-MAY-22
```

Independent Nested Queries

Example 1: IN

- Select all employees who won an award.

SQL> SELECT id, name FROM employeedata WHERE id IN (SELECT employee_id FROM awards);

OUTPUT:

```
      ID  NAME
----------  ------------------------
       1   Augustine Hammond
       3   Ragu Delafoy
```

Example 2: NOT IN

- Select all employees who never won an award.

SQL> SELECT id, name FROM employeedata WHERE id NOT IN (SELECT employee_id FROM awards);

OUTPUT:

```
      ID          NAME
----------        ------------------------
       2          Perice John
       4          Teakwood Saffen
       5          Freddy Malcom
```

Example 3: ALL

- Select all Developers who earn more than all the Managers

SQL> SELECT * FROM employeedata WHERE role = 'Developer' AND salary > ALL (SELECT salary FROM employeedata WHERE role = 'Manager');

OUTPUT:

```
      ID NAME                      SALARY          ROLE
---------- ------------------------       ----------       ---------------
       5 Freddy Malcom              50000          Developer
```

Example 4: ANY

- Select all Developers who earn more than any Manager

```
SQL> SELECT * FROM employeedata WHERE role = 'Developer' AND salary > ANY
(SELECT salary FROM employeedata WHERE role = 'Manager');
```

OUTPUT:

```
 ID     NAME            SALARY              ROLE
---------- ------------------------ ----------         ---------------
     3 Ragu Delafoy                30000       Developer
     5 Freddy Malcom               50000       Developer
```

Co-related Nested Queries

- Select all employees whose salary is above the average salary of employees in their role.

Example:

```
SQL> SELECT * FROM employeedata emp1 WHERE salary > (SELECT AVG(salary) FROM
employeedata emp2 WHERE emp1.role = emp2.role);
```

OUTPUT:

```
    ID NAME                        SALARY     ROLE
---------- ------------------------ ----------         ---------------
     4 Teakwood Saffen              40000      Manager
     5 Freddy Malcom                50000     Developer
```

Explanation

The manager with id 4 earns more than the average salary of all managers (25000), and the developer with id 5 earns more than the average salary of all developers (30000). The inner query is executed for all rows fetched by the outer query. The role value (emp1.role) of every outer query's row is used by the inner query (emp1.role = emp2.role).

- We can find the average salary of managers and developers using the below query:

```
SQL> SELECT role, AVG(salary) FROM employeedata GROUP BY role;
```

OUTPUT:

```
ROLE            AVG(SALARY)

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

Developer           30000

Manager             25000
```

RESULT:

Thus, the operations using various SQL nested queries on the database were executed successfully.

EX.NO: 5b
Date:

JOIN QUERIES

AIM

To perform operations using various SQL JOIN queries on the database.

PROCEDURE:

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

Different Types of SQL JOINs

Here are the different types of the JOINs in SQL:

1. **(INNER) JOIN: Returns records that have matching values in both tables**
   Syntax:

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

2. **LEFT JOIN: Returns all records from the left table, and the matched records from the right table**
   Syntax:

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

3. **RIGHT (OUTER) JOIN: Returns all records from the right table, and the matched records from the left table**
   Syntax:

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

4. **FULL (OUTER) JOIN: Returns all records when there is a match in either left or right table**

Syntax:

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

EXAMPLE:

CREATE A TABLE PRODUCTORDERS

```
SQL> CREATE table productorders(Order_Id number(5),Orderno number(5),P_Id number(3));

Table created.

SQL> desc productorders;

 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 ORDER_ID                                           NUMBER(5)
 ORDERNO                                            NUMBER(5)
 P_ID                                               NUMBER(3)
```

INSERTING VALUES INTO THE TABLE PRODUCTORDERS

```
SQL> INSERT into productorders values(&Order_Id,&Orderno,&P_Id);

Enter value for order_id: 1

Enter value for orderno: 77895

Enter value for p_id: 3

old   1: INSERT into productorders values(&Order_Id,&Orderno,&P_Id)

new   1: INSERT into productorders values(1,77895,3)

1 row created.


SQL> INSERT into productorders values(&Order_Id,&Orderno,&P_Id);

Enter value for order_id: 2
Enter value for orderno: 44678

Enter value for p_id: 3

old   1: INSERT into productorders values(&Order_Id,&Orderno,&P_Id)

new   1: INSERT into productorders values(2,44678,3)

1 row created.

SQL> INSERT into productorders values(&Order_Id,&Orderno,&P_Id);

Enter value for order_id: 3

Enter value for orderno: 22456

Enter value for p_id: 1

old   1: INSERT into productorders values(&Order_Id,&Orderno,&P_Id)

new   1: INSERT into productorders values(3,22456,1)

1 row created.

SQL> INSERT into productorders values(&Order_Id,&Orderno,&P_Id);

Enter value for order_id: 4

Enter value for orderno: 24562

Enter value for p_id: 1

old   1: INSERT into productorders values(&Order_Id,&Orderno,&P_Id)

new   1: INSERT into productorders values(4,24562,1)

1 row created.

SQL> INSERT into productorders values(&Order_Id,&Orderno,&P_Id);

Enter value for order_id: 5

Enter value for orderno: 34764

Enter value for p_id: 15

old   1: INSERT into productorders values(&Order_Id,&Orderno,&P_Id)

new   1: INSERT into productorders values(5,34764,15)

1 row created.
```

DISPLAYING DATA FROM TABLE PRODUCTORDERS

```
SQL> select * from productorders;

  ORDER_ID    ORDERNO      P_ID

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

         1         77895          3

         2        44678           3

         3        22456           1

         4        24562           1

         5        34764          15
```

CREATE A SECOND TABLE PERSON

```
SQL> CREATE table person(p_Id number(5),LASTNAME varchar2(10),Firstname varchar2(15),
Address varchar2(20),city varchar2(10));

Table created.
```

INSERTING VALUES INTO THE TABLE PERSON

```
SQL> INSERT into person values(&p_Id,'&Lastname','&firstname','&Address','&city');

Enter value for p_id: 1

Enter value for lastname: Smith

Enter value for firstname: Jadon

Enter value for address: Ramapuram

Enter value for city: Chennai

old   1: INSERT into person values(&p_Id,'&Lastname','&firstname','&Address','&city')

new   1: INSERT into person values(1,'Smith','Jadon','Ramapuram','Chennai')

1 row created.


SQL> INSERT into person values(&p_Id,'&Lastname','&firstname','&Address','&city');

Enter value for p_id: 2
Enter value for lastname: Hemal

Enter value for firstname: Elango

Enter value for address: Anna Nagar

Enter value for city: Tamilnadu

old   1: INSERT into person values(&p_Id,'&Lastname','&firstname','&Address','&city')

new   1: INSERT into person values(2,'Hemal','Elango','Anna Nagar','Tamilnadu')

1 row created.


SQL> INSERT into person values(&p_Id,'&Lastname','&firstname','&Address','&city');

Enter value for p_id: 3

Enter value for lastname: Lanser

Enter value for firstname: Kim

Enter value for address: Hyderabad

Enter value for city: AP

old   1: INSERT into person values(&p_Id,'&Lastname','&firstname','&Address','&city')

new   1: INSERT into person values(3,'Lanser','Kim','Hyderabad','AP')

1 row created.
```

DISPLAYING DATA FROM TABLE PERSON

```
SQL> select * from person;

      P_ID LASTNAME   FIRSTNAME       ADDRESS             CITY

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

         1  Smith              Jadon             Ramapuram                  Chennai

         2  Hemal              Elango            Anna Nagar                Tamilnadu

         3  Lanser             Kim               Hyderabad                 AP
```

1 INNER JOIN

OUTPUT

```
SQL> SELECT person.firstname,person.city FROM person INNER JOIN productorders ON person.p_Id
= productorders.p_Id;

FIRSTNAME      CITY

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

Kim                AP

Kim                AP

Jadon              Chennai

Jadon              Chennai
```

2 LEFT JOIN

OUTPUT

```
SQL> SELECT person.lastname,person.firstname,productorders.orderno FROM person LEFT JOIN
productorders ON person.p_Id = productorders.p_Id  ORDER BY person.lastname;

LASTNAME   FIRSTNAME         ORDERNO

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

Hemal              Elango

Lanser             Kim                     77895

Lanser             Kim                     44678

Smith              Jadon                   24562

Smith              Jadon                   22456
```

3 RIGHT (OUTER) JOIN

OUTPUT

```
SQL> SELECT person.lastname,person.firstname,productorders.orderno FROM person RIGHT OUTER
JOIN productorders ON person.p_Id = productorders.p_Id ORDER BY person.firstname;
```

```
LASTNAME   FIRSTNAME        ORDERNO

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

Smith               Jadon              22456

Smith               Jadon              24562

Lanser              Kim                77895

Lanser              Kim                44678

                                        34764
```

4 FULL OUTER JOIN

OUTPUT

```
SQL> SELECT person.lastname,person.address FROM person FULL OUTER JOIN productorders ON
person.p_Id = productorders.p_Id ORDER BY person.firstname;

LASTNAME    ADDRESS

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

Hemal               Anna Nagar

Smith               Ramapuram

Smith               Ramapuram

Lanser              Hyderabad

Lanser              Hyderabad

6 rows selected.
```

<u>RESULT:</u>

Thus, the operations using various SQL JOIN queries on the database were executed successfully.

Ex. No: 6
Date:

SET OPERATORS AND VIEWS

AIM:

To write a SQL queries to implement the set operators and views.

PROCEDURE:

SQL SET OPERATION:

The SQL Set operation is used to combine the two or more SQL SELECT statements.

Types of Set Operation

1. Union
2. UnionAll
3. Intersect
4. Minus

1. Union

- The SQL Union operation is used to combine the result of two or more SQL SELECT queries.
- In the union operation, all the number of datatype and columns must be same in both the tables on which UNION operation is being applied.
- The union operation eliminates the duplicate rows from its resultset.

Syntax:

```
SELECT column_name FROM table1   UNION   SELECT column_name FROM table2;
```

2. Union All

- Union All operation is equal to the Union operation. It returns the set without removing duplication and sorting the data.

Syntax:

```
SELECT column_name FROM table1  UNION ALL  SELECT column_name FROM table2;
```

3. Intersect

- It is used to combine two SELECT statements. The Intersect operation returns the common rows from both the SELECT statements.
- In the Intersect operation, the number of datatype and columns must be the same.
- It has no duplicates and it arranges the data in ascending order by default.

Syntax:

SELECT column_name FROM table1 INTERSECT SELECT column_name FROM table2;

4. Minus

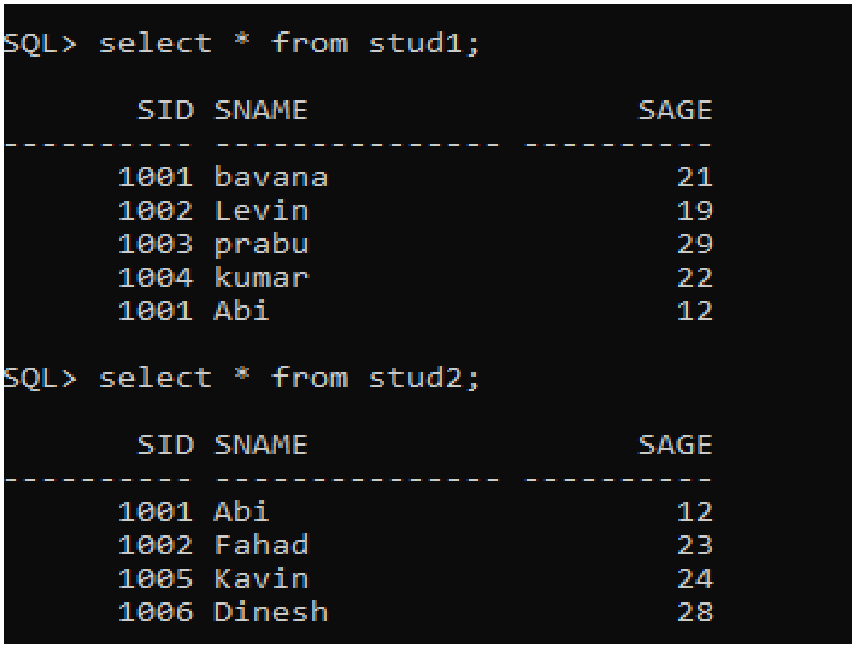
- It combines the result of two SELECT statements. Minus operator is used to display the rows which are present in the first query but absent in the second query.
- It has no duplicates and data arranged in ascending order by default.

Syntax:

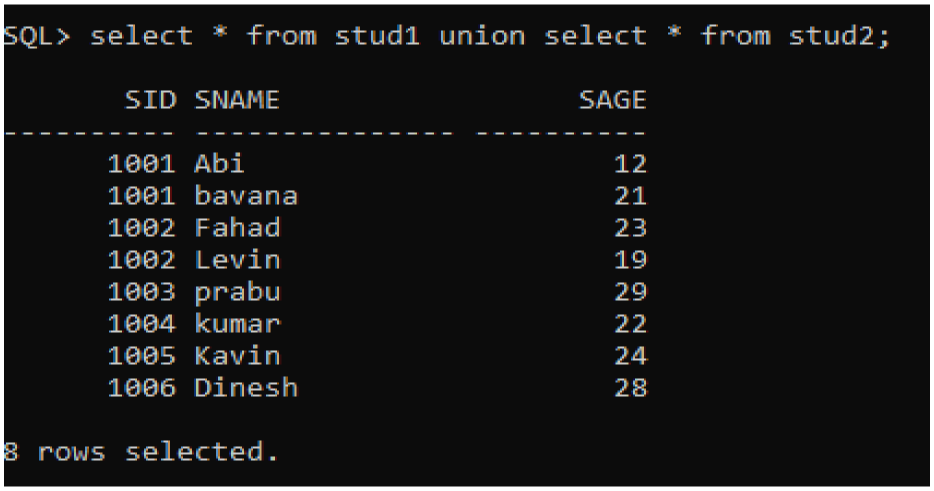
SELECT column_name FROM table1 MINUS SELECT column_name FROM table2;

Output:

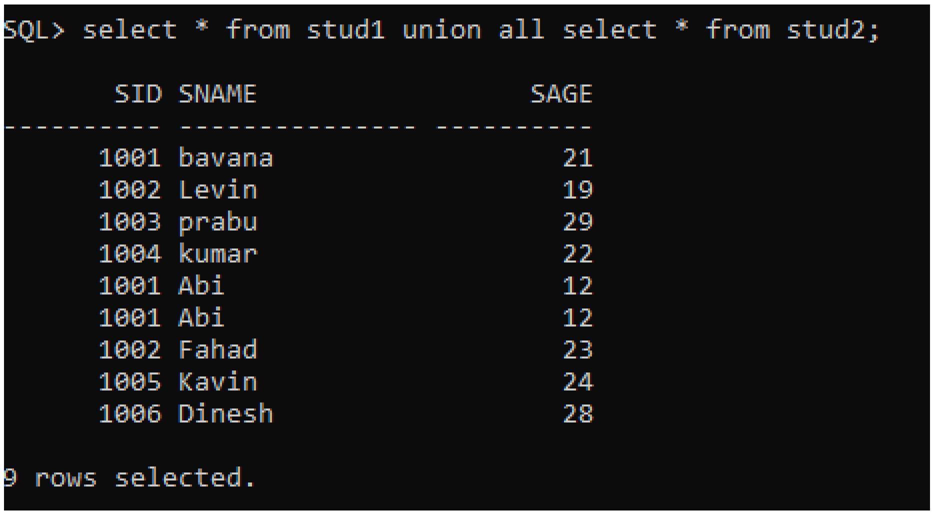
Creating two tables namely stud1 and stud2.



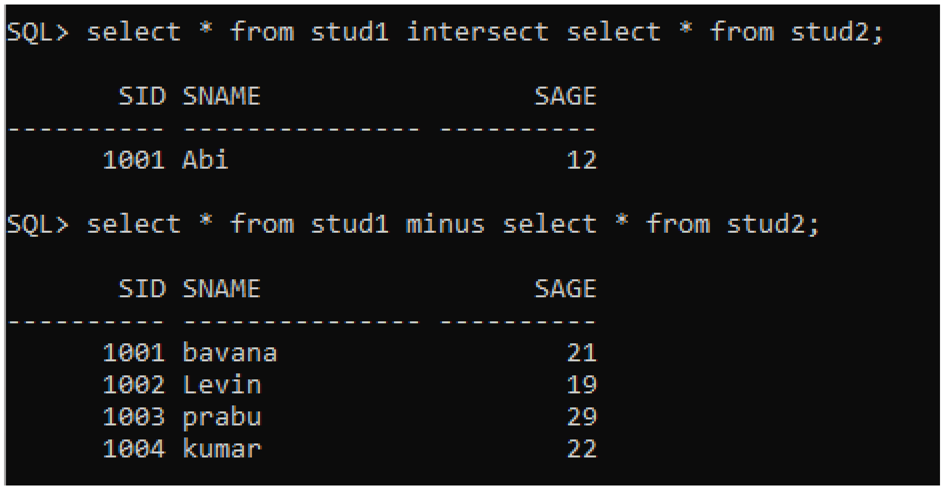
```
SQL> select * from stud1;

       SID SNAME                      SAGE
---------- --------------- ----------
      1001 bavana                       21
      1002 Levin                        19
      1003 prabu                        29
      1004 kumar                        22
      1001 Abi                          12

SQL> select * from stud2;

       SID SNAME                      SAGE
---------- --------------- ----------
      1001 Abi                          12
      1002 Fahad                        23
      1005 Kavin                        24
      1006 Dinesh                       28
```



```
SQL> select * from stud1 union select * from stud2;

       SID SNAME                      SAGE
---------- --------------- ----------
      1001 Abi                          12
      1001 bavana                       21
      1002 Fahad                        23
      1002 Levin                        19
      1003 prabu                        29
      1004 kumar                        22
      1005 Kavin                        24
      1006 Dinesh                       28

8 rows selected.
```



```
SQL> select * from stud1 union all select * from stud2;

       SID SNAME                      SAGE
---------- --------------- ----------
      1001 bavana                       21
      1002 Levin                        19
      1003 prabu                        29
      1004 kumar                        22
      1001 Abi                          12
      1001 Abi                          12
      1002 Fahad                        23
      1005 Kavin                        24
      1006 Dinesh                       28

9 rows selected.
```



```
SQL> select * from stud1 intersect select * from stud2;

       SID SNAME                      SAGE
---------- --------------- ----------
      1001 Abi                          12

SQL> select * from stud1 minus select * from stud2;

       SID SNAME                      SAGE
---------- --------------- ----------
      1001 bavana                       21
      1002 Levin                        19
      1003 prabu                        29
      1004 kumar                        22
```

SQL VIEW

SQL provides the concept of VIEW, which hides the complexity of the data and restricts unnecessary access to the database.

It permits the users to access only a particular column rather than the whole data of the table.

The **View** in the Structured Query Language is considered as the virtual table, which depends on the result-set of the predefined SQL statement.

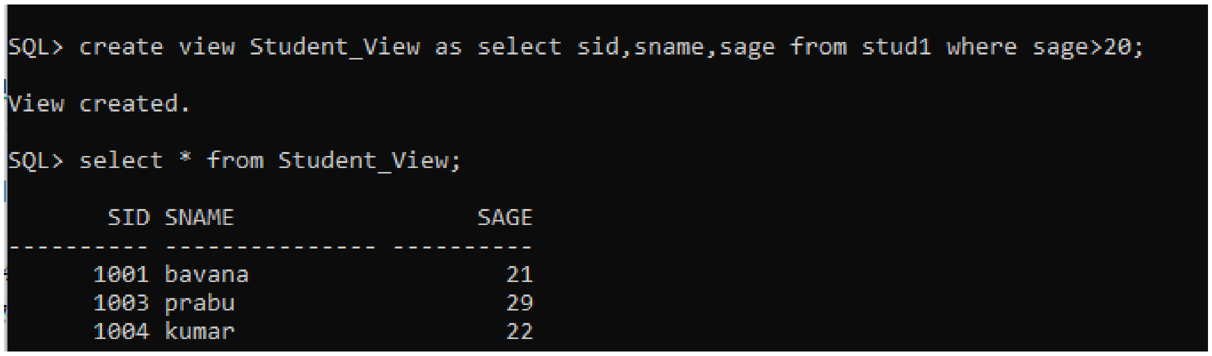
Create a SQL View

To create a View in Structured Query Language by using the CREATE VIEW statement. Creation of View from a single table or multiple tables.

Syntax to Create View from Single Table

> **CREATE VIEW** View_Name **AS**
> **SELECT** Column_Name1, Column_Name2, ..., Column_NameN
> **FROM** Table_Name
> **WHERE** condition;

Example to Create a View from Single table



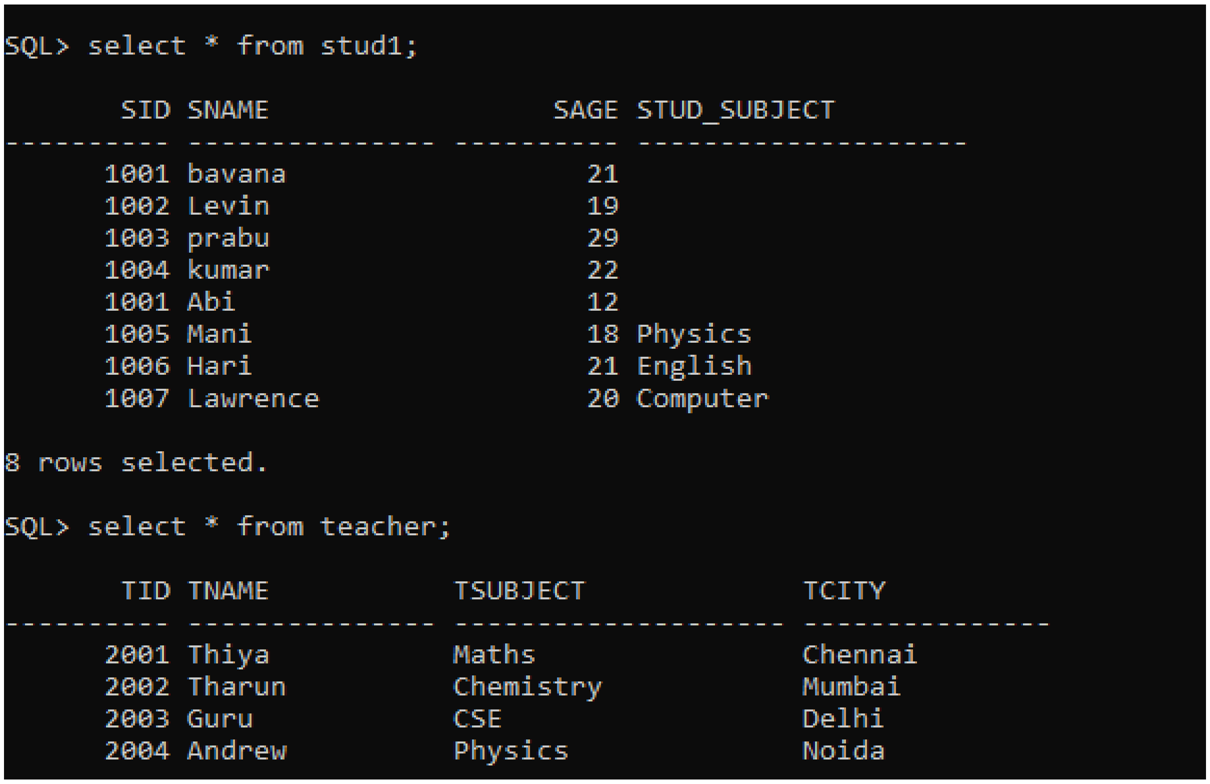
```
SQL> create view Student_View as select sid,sname,sage from stud1 where sage>20;

View created.

SQL> select * from Student_View;

       SID SNAME                 SAGE
---------- --------------- ----------
      1001 bavana                  21
      1003 prabu                   29
      1004 kumar                   22
```

Syntax to Create View from Multiple Tables

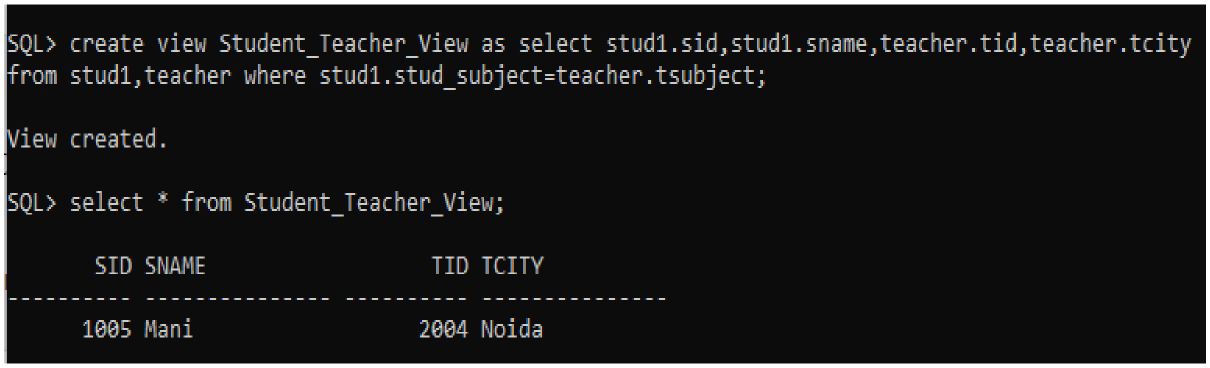
To create a View from multiple tables by including the tables in the SELECT statement.

> CREATE **VIEW** View_Name **AS**
> **SELECT** Table_Name1.Column_Name1, Table_Name1.Column_Name2,
> Table_Name2.Column_Name2, ..., Table_NameN.Column_NameN
> **FROM** Table_Name1, Table_Name2, ..., Table_NameN
> **WHERE** condition;

Example to Create a View from Multiple tables



```
SQL> select * from stud1;

       SID SNAME                 SAGE STUD_SUBJECT
---------- --------------- ---------- --------------------
      1001 bavana                  21
      1002 Levin                   19
      1003 prabu                   29
      1004 kumar                   22
      1001 Abi                     12
      1005 Mani                    18 Physics
      1006 Hari                    21 English
      1007 Lawrence                20 Computer

8 rows selected.

SQL> select * from teacher;

       TID TNAME           TSUBJECT             TCITY
---------- --------------- -------------------- ---------------
      2001 Thiya           Maths                Chennai
      2002 Tharun          Chemistry            Mumbai
      2003 Guru            CSE                  Delhi
      2004 Andrew          Physics              Noida
```



```
SQL> create view Student_Teacher_View as select stud1.sid,stud1.sname,teacher.tid,teacher.tcity
from stud1,teacher where stud1.stud_subject=teacher.tsubject;

View created.

SQL> select * from Student_Teacher_View;

       SID SNAME                  TID TCITY
---------- --------------- ---------- ---------------
      1005 Mani                  2004 Noida
```

Update an SQL View

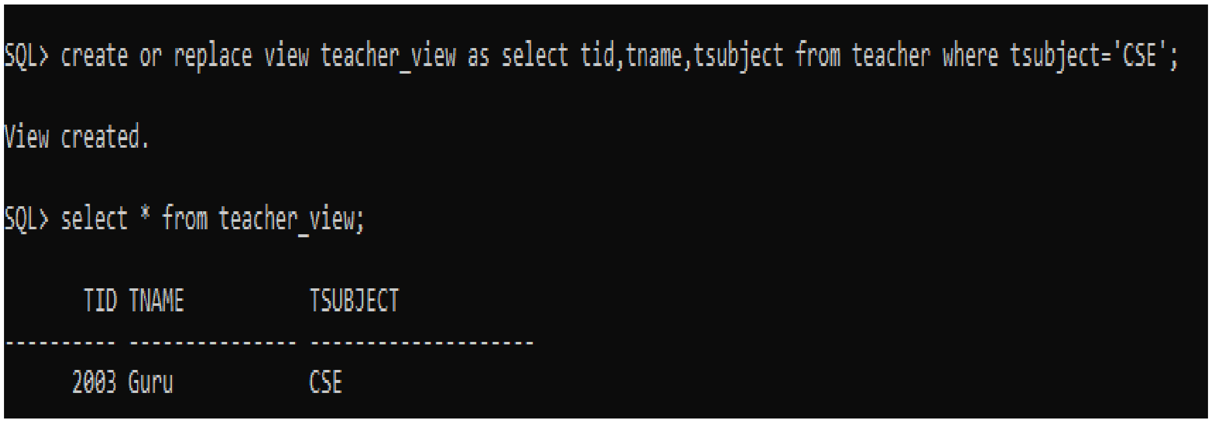
A view in SQL can only be modified if the view follows the following conditions:

1. You can update that view which depends on only one table. SQL will not allow updating the view which is created more than one table.
2. The fields of view should not contain NULL values.
3. The view does not contain any subquery and DISTINCT keyword in its definition.

4. The views cannot be updatable if the SELECT statement used to create a View contains JOIN or HAVING or GROUP BY clause.
5. If any field of view contains any SQL aggregate function, you cannot modify the view.

Syntax to Update a View

CREATE OR REPLACE **VIEW** View_Name **AS**
SELECT Column_Name1, Column_Name2, ..., Column_NameN
FROM Table_Name
WHERE condition;



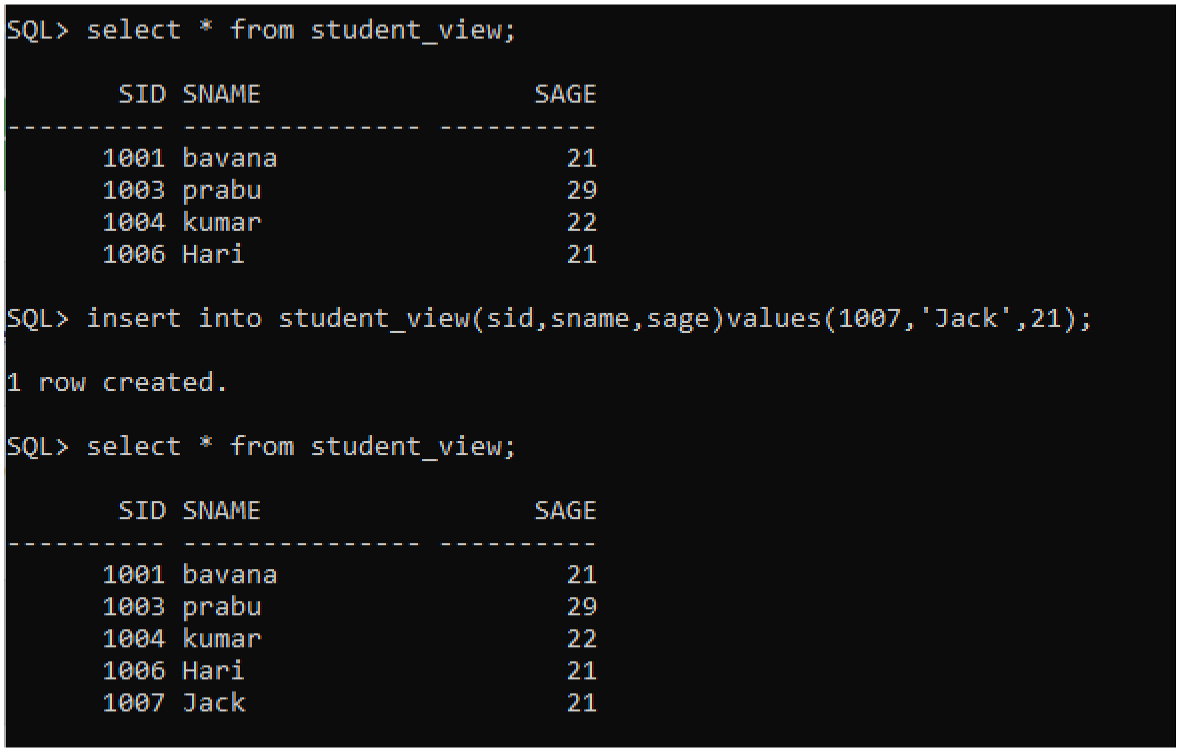
```
SQL> create or replace view teacher_view as select tid,tname,tsubject from teacher where tsubject='CSE';

View created.

SQL> select * from teacher_view;

       TID TNAME          TSUBJECT
---------- -------------- -------------------
      2003 Guru           CSE
```

Insert the new row into the existing view

Just like the insertion process of database tables, we can also insert the record in the views. The following SQL INSERT statement is used to insert the new row or record in the view:

Syntax:

INSERT INTO View_Name(Column_Name1, Column_Name2 , Column_Name3, ..., Column_NameN) **VALUES**(value1, value2, value3, ..., valueN);



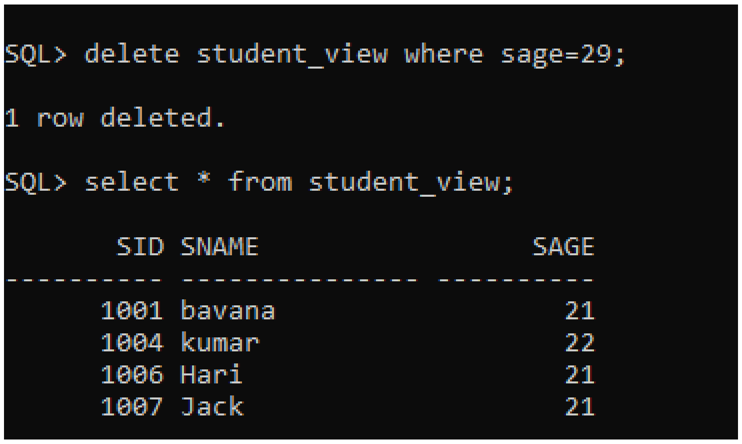
```
SQL> select * from student_view;

       SID SNAME                 SAGE
---------- --------------- ----------
      1001 bavana                  21
      1003 prabu                   29
      1004 kumar                   22
      1006 Hari                    21

SQL> insert into student_view(sid,sname,sage)values(1007,'Jack',21);

1 row created.

SQL> select * from student_view;

       SID SNAME                 SAGE
---------- --------------- ----------
      1001 bavana                  21
      1003 prabu                   29
      1004 kumar                   22
      1006 Hari                    21
      1007 Jack                    21
```

Delete the existing row from the view

Just like the deletion process of database tables, we can also delete the record from the views. The following SQL DELETE statement is used to delete the existing row or record from the view:

Syntax

DELETE FROM View_Name **WHERE** Condition;



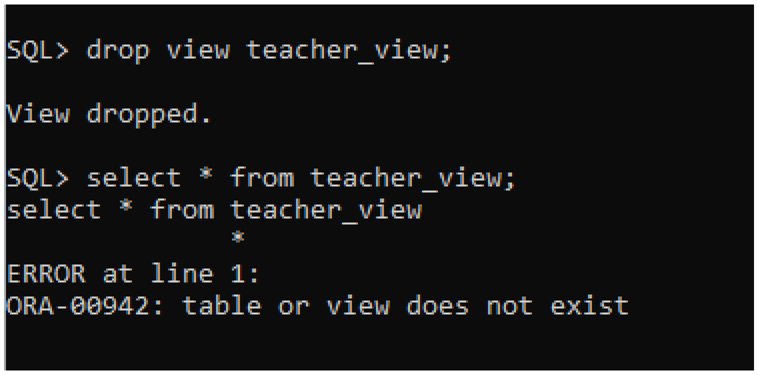
```
SQL> delete student_view where sage=29;

1 row deleted.

SQL> select * from student_view;

       SID SNAME                 SAGE
---------- --------------- ----------
      1001 bavana                  21
      1004 kumar                   22
      1006 Hari                    21
      1007 Jack                    21
```

Drop a View

To delete the existing view from the database if it is no longer needed the following SQL DROP statement is used to delete the view:

Syntax:

DROP VIEW View_Name;



```
SQL> drop view teacher_view;

View dropped.

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

RESULT:

Thus, the SQL queries for implementing the set operators and views are executed successfully.

EX.NO: 7
Date:

PL/SQL CONDITIONAL AND ITERATIVE STATEMENTS

AIM

To study the various basic PL/SQL **Conditional and Iterative Statements** on the database.

DESCRIPTION:

The iterative statements are used to repeat the execution of certain statements multiple times. This is achieved with the help of loops. Loops in PL/SQL provide a mechanism to perform specific tasks multiple times without having to write them multiple times.

This article will discuss three main types of loops:

- **Basic loop**
- **WHILE loop**
- **FOR loop**

Basic loop

The basic loop will execute the statement provided a certain number of times until the exit condition is met. It is necessary to have an EXIT statement so that the loop does not run indefinitely. There is also an increment statement that can be used to increase/decrease the changing variable in the loop.

Syntax:

```
LOOP
Statements;
[increment_statement]
EXIT condition;
END LOOP;
```

WHILE loop

The WHILE loop in PL/SQL is used to check the entry condition, and if the entry condition is true, only then is the loop executed. The basic loop executes at least once, whereas the WHILE loop will first check the condition provided in the boolean expression. If the condition is false, the control does not enter the loop.

Syntax:

```
WHILE (boolean_expression) LOOP
statements ;
[increment_statement]
END LOOP;
```

FOR loop

The FOR loop in PL/SQL provides implicit variable declaration, implicit incrementation of the variable by one, and implicit exit also. In the FOR loop, we do not have to declare the variable as we did in the previous two types of loop. While writing the loop statement, the variable is declared implicitly. The range consists of the starting value, from where the value of the iterating variable begins, and the end value, which determines the last value which the variable can have. In each loop, the variable is incremented by one.

Syntax:

```
FOR variable IN range LOOP
Statements;
END LOOP;
```

EXAMPLES:

1. PL/SQL CODING FOR ADDITION OF TWO NUMBERS

```
SQL> declare
a number;
b number;
c number;
begin
a:=&a;
b:=&b;
c:=a+b;
dbms_output.put_line('sum of'||a||'and'||b||'is'||c);
end;
/
```

INPUT:

```
Enter value for a: 23
old 6: a:=&a;
new 6: a:=23;
Enter value for b: 12
old 7: b:=&b;
new 7: b:=12;
```

OUTPUT:

```
sum of23and12is35
PL/SQL procedure successfully completed.
```

2. PL/ SQL GENERAL SYNTAX FOR IF CONDITION:

```
SQL> DECLARE
<VARIABLE DECLARATION>;
BEGIN
IF(CONDITION)THEN
<EXECUTABLE STATEMENT >;
END;
```

Coding for If Statement:

```
DECLARE
b number;
c number;
BEGIN
B:=10;
C:=20; if(C>B)
THEN
dbms_output.put_line('C is maximum');
end if;
end;
/
```

OUTPUT:

```
C is maximum
PL/SQL procedure successfully completed.
```

3. PL/ SQL GENERAL SYNTAX FOR IF AND ELSECONDITION:

```
SQL> DECLARE
<VARIABLE DECLARATION>;
BEGIN
IF (TEST CONDITION) THEN
<STATEMENTS>;
ELSE
<STATEMENTS>;
ENDIF;
END;
******************Less then or Greater Using IF ELSE **********************
SQL> declare
n number;
begin
dbms_output. put_line('enter a number');
n:=&number;
if n<5 then
dbms_output.put_line('entered number is less than 5');
else
dbms_output.put_line('entered number is greater than 5');
end if;
end;
/
```

INPUT

```
Enter value for number: 2
old 5: n:=&number;
new 5: n:=2;
```

OUTPUT:

```
entered number is less than 5
PL/SQL procedure successfully completed.
```

4.PL/ SQL GENERAL SYNTAX FOR NESTED IF:

```
SQL> DECLARE
<VARIABLE DECLARATION>;
BEGIN
```

```
IF (TEST CONDITION) THEN
<STATEMENTS>;
ELSEIF (TEST CONDITION) THEN
<STATEMENTS>;
ELSE
<STATEMENTS>;
ENDIF;
END;
```

********** GREATEST OF THREE NUMBERS USING IF ELSEIF************

```
SQL> declare
a number;
b number;
c number;
d number;
begin
a:=&a;
b:=&b;
c:=&b;
if(a>b)and(a>c) then
dbms_output.put_line('A is maximum');
elsif(b>a)and(b>c)then
dbms_output.put_line('B is maximum');
else
dbms_output.put_line('C is maximum');
end if;
end;
/
```

INPUT:

```
Enter value for a: 21
old 7: a:=&a;
new 7: a:=21;
Enter value for b: 12
old 8: b:=&b;
new 8: b:=12;
Enter value for b: 45
old 9: c:=&b;
new 9: c:=45;
```

OUTPUT:

```
C is maximum

PL/SQL procedure successfully completed.
```

5.PL/ SQL GENERAL SYNTAX FOR LOOPING STATEMENT:

```
SQL> DECLARE
<VARIABLE DECLARATION>;
BEGIN
LOOP
<STATEMENT>;
END LOOP;
<EXECUTAVLE STATEMENT>;
END;
```

***********SUMMATION OF ODD NUMBERS USING FOR LOOP***********

```
SQL> declare
n number;
sum1 number default 0;
endvalue number;
begin
endvalue:=&endvalue;
n:=1;
for n in 1..endvalue
loop
if mod(n,2)=1
then
sum1:=sum1+n;
end if;
end loop;
dbms_output.put_line('sum ='||sum1);
end;
/
```

INPUT:

```
Enter value for endvalue: 4
old 6: endvalue:=&endvalue;
new 6: endvalue:=4;
```

OUTPUT:

```
sum =4
PL/SQL procedure successfully completed.
```

6.PL/ SQL GENERAL SYNTAX FOR LOOPING STATEMENT:

```
SQL> DECLARE
<VARIABLE DECLARATION>;
BEGIN
WHILE <condition>
LOOP
<STATEMENT>;
END LOOP;
```

```
<EXECUTAVLE STATEMENT>;
END;
```

```
*********SUMMATION OF ODD NUMBERS USING WHILE LOOP**********
SQL> declare
n number;
sum1 number default 0;
endvalue number;
begin
endvalue:=&endvalue;
n:=1;
while(n<endvalue)
loop
sum1:=sum1+n;
n:=n+2;
end loop;
dbms_output.put_line('sum of odd no. bt 1 and' ||endvalue||'is'||sum1);
end;
/
```

INPUT:

```
Enter value for endvalue: 4
old 6: endvalue:=&endvalue;
new 6: endvalue:=4;
```

OUTPUT:

```
sum of odd no. bt 1 and4is4
PL/SQL procedure successfully completed.
```

<u>RESULT:</u>

Thus, the operations using various basic PL/SQL conditional and iterative statements (Basic loop, WHILE loop, FOR loop) on the database were executed successfully.

EX.NO: 8a
Date:

PL/SQL Cursors

AIM

To create a database using implicit & explicit cursors.

Explanation:

Cursor:

Pointer to context area controls (PL/SQL) the context area through a cursor. Hold a row (one or more) returned by a SQL statement. The set of rows the cursor holds is referred to as the active set.

Types of cursors:

1) Implicit Cursors
2) Explicit Cursors

Implicit Cursors:

Automatically created by Oracle whenever SQL statement is created and executed, no explicit cursor is found then.

Programmers cannot control the implicit cursors and information in it.

Attributes in IC are % found, % is OPEN, % IS NOTFOUND and % ROWCOUNT SQL cursors has additional attributes such as % Bulk_rowcount and % Bulk_Exceptions , to use with for all statements.

Explicit Cursors:

Explicit cursors are programmer defined cursors for gaining more control over context area.

It should be defined in declaration section of the PL/SQL block.

It is created on select statement which returns more than one row.

Syntax:

```
CURSOR cursor_name IS select_statement;
```

Program:

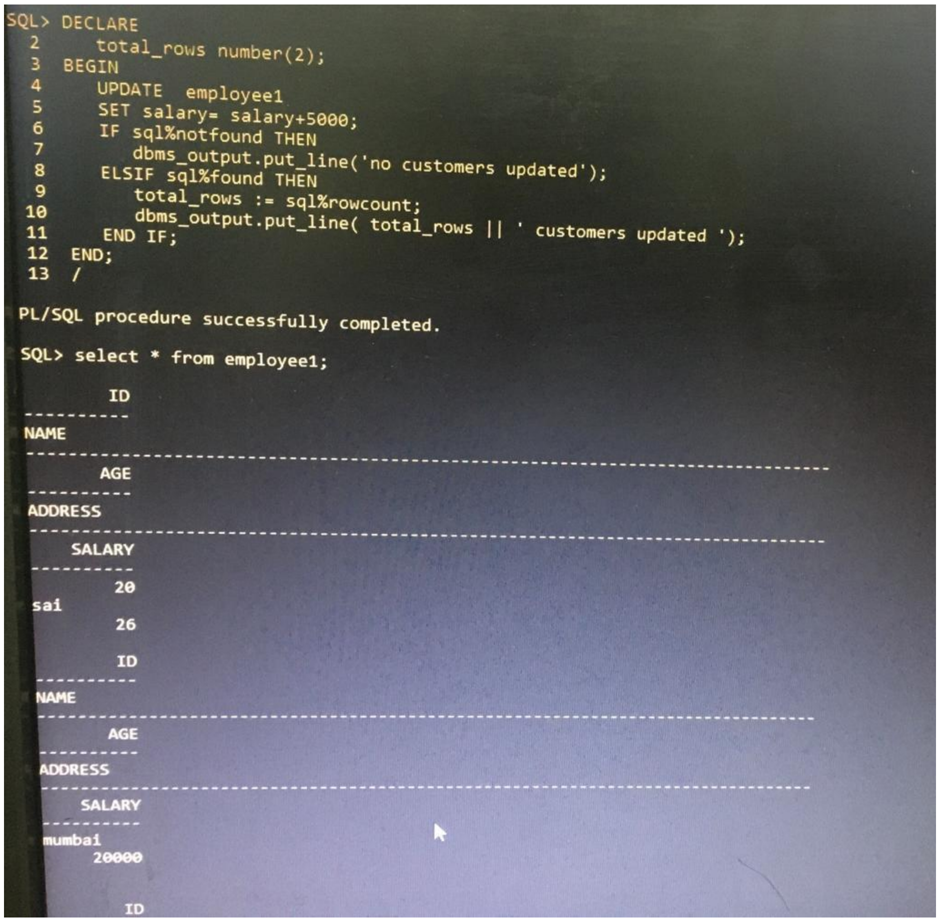
```
SQL>DECLARE
total_rows number(2);
BEGIN
UPDATE employee1
SET salary = salary + 5000;
IF sql%notfound THEN
dbms_output.put_line(‘no customers updated’); ELSIF
sql%found THEN total_rows:= sql%rowcount;
dbms_output.put_line( total_rows || ‘customers updated’);
END IF;
END;
/
3 customers updated

PL/SQL procedure successfully completed.

SQL> select* from employee1;

ID NAME        AGE ADDRESS     SALARY
------------- ----------- ------------- ---------- -----------
30 sai                26 mumbai          155000
40 sam          27 hyderabad  205000
50 tom          29 pune               45630
```

EXPLICIT CURSOR

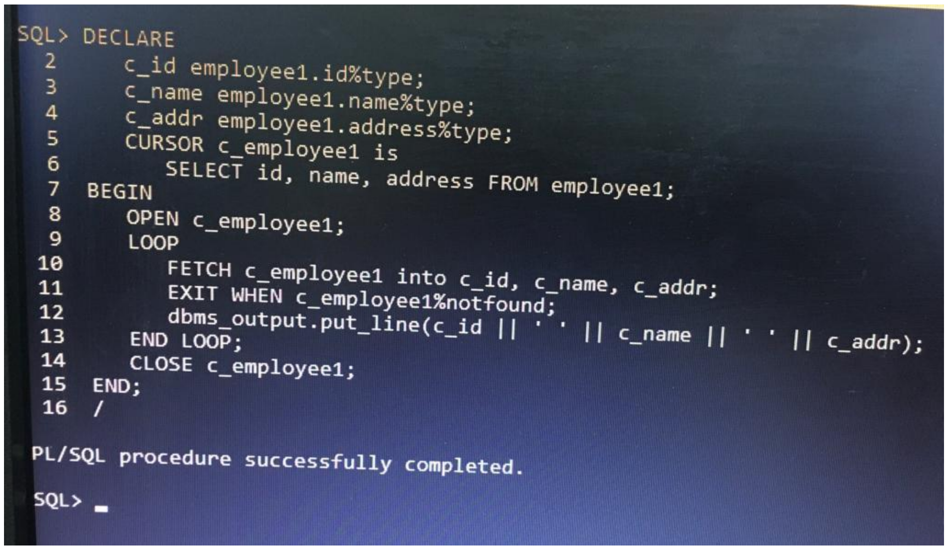
```
SQL> DECLARE
c_id employee1.id%type;
```

```
c_name employee1.name%type;

c_addr employee1.address%type;

CURSOR c_employee1 is

SELECT id,name,address FROM employee1;

begin open c_employee1; loop

FETCH c_employee1 into c_id,c_name,c_addr;

EXIT WHEN

c_employee1%notfound;

 dbms_output.put_line(c_id||' '||

c_name||' '||c_addr);

END LOOP;

CLOSE c_emp;

 END;

/

PL/SQL procedure successfully completed.
```

OUTPUT:



```
SQL> DECLARE
     total_rows number(2);
   BEGIN
     UPDATE  employee1
     SET salary= salary+5000;
     IF sql%notfound THEN
        dbms_output.put_line('no customers updated');
     ELSIF sql%found THEN
        total_rows := sql%rowcount;
        dbms_output.put_line( total_rows || ' customers updated ');
     END IF;
   END;
   /

PL/SQL procedure successfully completed.

SQL> select * from employee1;

        ID
----------
NAME
--------------------------------------------------------------------------------
       AGE
----------
ADDRESS
--------------------------------------------------------------------------------
    SALARY
----------
        20
sai
        26

        ID
----------
NAME
--------------------------------------------------------------------------------
       AGE
----------
ADDRESS
--------------------------------------------------------------------------------
    SALARY
----------
mumbai
     20000

        ID
```



```
SQL> DECLARE
      c_id employee1.id%type;
      c_name employee1.name%type;
      c_addr employee1.address%type;
      CURSOR c_employee1 is
         SELECT id, name, address FROM employee1;
   BEGIN
      OPEN c_employee1;
      LOOP
         FETCH c_employee1 into c_id, c_name, c_addr;
         EXIT WHEN c_employee1%notfound;
         dbms_output.put_line(c_id || ' ' || c_name || ' ' || c_addr);
      END LOOP;
      CLOSE c_employee1;
   END;
   /

PL/SQL procedure successfully completed.

SQL> _
```

<u>RESULT:</u>

Thus, PL/SQL cursors are executed successfully.

Ex. No: 8b
Date:

PL/SQL Trigger

AIM

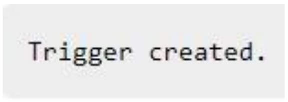
To write a program on PL/SQL Trigger

Program:

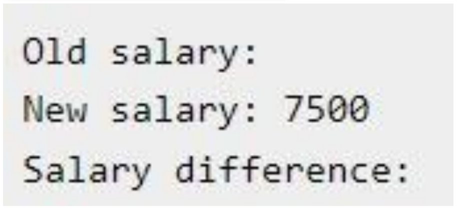
```
SQL> CREATE OR REPLACE TRIGGER display_salary_changes

BEFORE DELETE OR INSERT OR UPDATE ON emp

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;

/

Trigger created.

SQL> DECLARE

total_rows number(2);

BEGIN
```

```
UPDATE  emp

 SET salary = salary + 5000;

IF sql%notfound THEN

dbms_output.put_line('no customers updated');

ELSIF sql%found THEN

 total_rows := sql%rowcount;

dbms_output.put_line( total_rows || ' customers updated ');

END IF;

END;

/

Old salary: 155000

New salary: 160000

Salary difference: 5000

Old salary: 205000

New salary: 210000

Salary difference: 5000

Old salary: 45630

New salary: 50630

Salary difference: 5000

3 customers updated


PL/SQL procedure successfully completed.
```

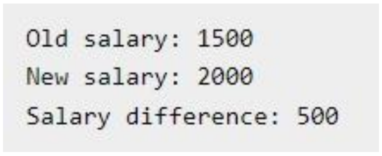
Output:



```
Trigger created.
```



```
Old salary:
New salary: 7500
Salary difference:
```



```
Old salary: 1500
New salary: 2000
Salary difference: 500
```

<u>Result:</u>

Thus, the above program is implemented successfully.

Ex. No: 9

Date:

MINI PROJECT

By working on Database Management System (DBMS) projects, students will:

- Learn to organize and clean raw data for efficient storage in databases.
- Gain experience in integrating data, standardizing it, and eliminating duplicates across multiple databases.
- Understand how to securely and quickly transfer data to external applications.
- Acquire skills in creating consistent and reliable datasets, improving data quality.
- Gain knowledge of data security and privacy measures to manage risks in data handling.

Upon completing DBMS projects, students will gain:

- A strong understanding of data types and database systems.
- Proficiency in using Microsoft SQL Server.
- Practical experience in creating databases, uploading/downloading data, and data manipulation.
- A solid foundation in database design and architecture.

DBMS Project Ideas

1. Customer Service Help Desk System
2. Fleet Management System
3. Online Ticketing System
4. Task Management System
5. Event Management System
6. Customer Relationship Management (CRM) System
7. Document Review and Approval System
8. Student Performance Analysis System
9. Online Classifieds System
10. Bank Accounts Management System

11. Supply Chain Management System
12. Smart City Database Management System
13. Real-time Stock Market Database Management
14. Real-time Chat Application
15. Library Database Management System
16. Smart Agriculture Database Management
17. Voting and Election Management System
18. Social Media Database Management System
19. Inventory Control Management
20. E-commerce Database Platform
21. Food Delivery System
22. Dairy Management System
23. Document Storage and Retrieval System
24. Restaurant Database Management
25. Volunteer Management System
26. Healthcare Appointment Scheduling System
27. Job Portal System
28. Blood Donation Management System
29. University Course Enrollment System
30. Train Reservation System
31. Real Estate Management System
32. Remote-server monitoring system
33. Parking Lot Management System
34. Library Database Management System
35. Pharmacy Database Management System
36. Payroll Database Management System
37. Bus Reservation System
38. Wholesaler Management System
39. Inventory Control Management
40. Cooking Recipe Database Management

41. Courier Service System
42. Online Learning Management System (LMS)
43. Bank Accounts Management System
44. Healthcare Inventory Management System
45. Train Reservation System
46. Subscription-Based Content Management System
47. Document Management System
48. Blood Donation Management System
49. E-commerce Database Platform
50. Parking Lot Management System