EXPERIMENT-1

Aim:

Introduction to MySQL Workbench. How to use MySQL Workbench to run SQL Statements.

MySQL:

- It is an open-source relational database
- It is cross-platform which means it runs on number of different platforms such as Windows, Linux and MacOs etc.

MySQL Workbench:

- Database designing
- SQL development
- Server administration

- Mysql workbench is a Visual database designing and modelling access tool for MySQL server relational database.
- It is an integrated development environment for MySQL server.
- It has utilities for database modelling, designing and server administration.
- It supports all objects such as tables, views, store procedures, triggers etc that make up a database.

SQL:

SQL stands for Structured Query Language which is a computer language for storing, manipulating and retrieving data stored in relational database.

Using MySQL Workbench to run SQL Statements:

Before any SQL statements can be executed on a database, the MySQL Workbench tool must first establish a connection to the target database server. This may take the form of a local server that is running on the same host as the workbench, or a server running on a remote system.

Developers use structured query language (SQL) commands, which are specific keywords or SQL statements, to work with data stored in relational databases. The following are categories for SQL commands.

- Data Definition Language (DDL)
- Data Manipulation Language (DML)
- Data Control Language (DCL)

DDL Commands:

These commands are used to create or modify the tables in SQL.
E.g.: CREATE, ALTER, TRUNCATE and DROP.

DML Commands:

These commands are used to change the data present in the SQL database
E.g.: SELECT, INSERT, UPDATE, DELETE

DCL Commands:

These are used to control access to data stored in a database
E.g.: GRANT, REVOKE

Company Relational Database Schema

EXPERIMENT-2

Aim:

Implementation of

i. DDL Commands: CREATE , ALTER, DROP and TRUNCATE.
ii. Constraints PRIMARY KEY, FOREIGN KEY,CHECK,NOT NULL,UNIQUE.

CREATE:

It is used to create a new table /a database / other object in the database.

SYNTAX:

CREATE database database_name
CREATE table table_name(col1 datatype(size), col2 datatype(size), col3 datatype(size),…… colndatatype(size));

EXAMPLE-1:

```
CREATE DATABASE 22501A0557;

USE 22501A0557;

CREATE TABLE EMPLOYEE (

   fname varchar(20) NOT NULL,

   minit varchar(20),

   lname varchar(20),

   ssn bigint,

   Bdate date,

   address varchar(40),

   gender char,

   salary bigint,

   super_ssn bigint references EMPLOYEE(ssn),

   dno int references DEPARTMENT(Dnumber)

);

DESC EMPLOYEE;
```

Field	Type	Null	Key	Default	Extra
Filter...	Filter...	Filter...	Filter...	Filter...	Filter...
fname	varchar(20)	NO		NULL	
minit	varchar(20)	YES		NULL	
lname	varchar(20)	YES		NULL	
ssn	bigint	YES		NULL	
Bdate	date	YES		NULL	
address	varchar(40)	YES		NULL	
gender	char(1)	YES		NULL	
salary	bigint	YES		NULL	
super_ssn	bigint	YES		NULL	
dno	int	YES		NULL	

EXAMPLE-2:

```
CREATE TABLE DEPARTMENT(
   Dname varchar(30) NOT NULL,
   Dnumber int,
   Mgr_ssn bigint NOT NULL references EMPLOYEE(ssn) ,
   Mgr_start_date date
);
DESC DEPARTMENT;
```

Field	Type	Null	Key	Default	Extra
Filter...	Filter...	Filter...	Filter...	Filter...	Filter...
Dname	varchar(30)	NO		NULL	
Dnumber	int	NO		NULL	
Mgr_ssn	bigint	NO		NULL	
Mgr_start_date	date	YES		NULL	

<u>ALTER</u>:

It is used to add a new column/a key to the table

SYNTAX-1:

ALTER table<table_name> add COLUMN <col_name><data_size>(size);

EXAMPLE:

1. alter table EMPLOYEE add column ph_num int(10);
   desc EMPLOYEE;

Field	Type	Null	Key	Default	Extra
Filter...	Filter...	Filter...	Filter...	Filter...	Filter...
fname	varchar(20)	NO		NULL	
minit	varchar(20)	YES		NULL	
lname	varchar(20)	YES		NULL	
ssn	bigint	YES		NULL	
Bdate	date	YES		NULL	
address	varchar(40)	YES		NULL	
gender	char(1)	YES		NULL	
salary	bigint	YES		NULL	
super_ssn	bigint	YES		NULL	
dno	int	YES		NULL	
ph_num	int	YES		NULL	

2. ALTER table <table_name> add PRIMARY KEY(col);
   alter table EMPLOYEE add primary key(Ssn);
   desc EMPLOYEE;

Field	Type	Null	Key	Default	Extra
Filter...	Filter...	Filter...	Filter...	Filter...	Filter...
fname	varchar(20)	NO		NULL	
minit	varchar(20)	YES		NULL	
lname	varchar(20)	YES		NULL	
ssn	bigint	NO	PRI	NULL	
Bdate	date	YES		NULL	
address	varchar(40)	YES		NULL	
gender	char(1)	YES		NULL	
salary	bigint	YES		NULL	
super_ssn	bigint	YES		NULL	
dno	int	YES		NULL	
ph_num	int	YES		NULL	

3. ALTER table<table_name> add FOREIGNKEY(col) REFERENCES<table><col_name>;
   alter table EMPLOYEE add foreign key(Dno) references DEPARTMENT(Dnumber);
   desc EMPLOYEE;

Field	Type	Null	Key	Default	Extra
Filter...	Filter...	Filter...	Filter...	Filter...	Filter...
fname	varchar(20)	NO		NULL	
minit	varchar(20)	YES		NULL	
lname	varchar(20)	YES		NULL	
ssn	bigint	NO	PRI	NULL	
Bdate	date	YES		NULL	
address	varchar(40)	YES		NULL	
gender	char(1)	YES		NULL	
salary	bigint	YES		NULL	
super_ssn	bigint	YES		NULL	
dno	int	YES	MUL	NULL	
ph_num	int	YES		NULL	

4. alter table DEPARTMENT add primary key(Dnumber);
   desc DEPARTMENT;

Field	Type	Null	Key	Default	Extra
Filter...	Filter...	Filter...	Filter...	Filter...	Filter...
Dname	varchar(30)	NO		NULL	
Dnumber	int	NO	PRI	NULL	
Mgr_ssn	bigint	NO		NULL	
Mgr_start_date	date	YES		NULL	

5. alter table DEPARTMENT add foreign key(Mgr_ssn) references EMPLOYEE(Ssn);
   desc DEPARTMENT;

Field	Type	Null	Key	Default	Extra
Filter...	Filter...	Filter...	Filter...	Filter...	Filter...
Dname	varchar(30)	NO		NULL	
Dnumber	int	NO	PRI	NULL	
Mgr_ssn	bigint	NO	MUL	NULL	
Mgr_start_date	date	YES		NULL	

SYNTAX-2:

Used to delete a column from the table.

Alter table table_name drop column column_name

EXAMPLE:

Alter table EMPLOYEE drop column ph_num;

desc EMPLOYEE;

Field	Type	Null	Key	Default	Extra
Filter...	Filter...	Filter...	Filter...	Filter...	Filter...
fname	varchar(20)	NO		NULL	
minit	varchar(20)	YES		NULL	
lname	varchar(20)	YES		NULL	
ssn	bigint	NO	PRI	NULL	
Bdate	date	YES		NULL	
address	varchar(40)	YES		NULL	
gender	char(1)	YES		NULL	
salary	bigint	YES		NULL	
super_ssn	bigint	YES		NULL	
dno	int	YES	MUL	NULL	

SYNTAX-3:

It is used to modify the size of the existing column.

ALTER table<table_name> modify COLUMN <column_name>(new_size);

EXAMPLE:

Alter table EMPLOYEE modify Address varchar(50);
desc EMPLOYEE;

Field	Type	Null	Key	Default	Extra
Filter...	Filter...	Filter...	Filter...	Filter...	Filter...
fname	varchar(20)	NO		NULL	
minit	varchar(20)	YES		NULL	
lname	varchar(20)	YES		NULL	
ssn	bigint	NO	PRI	NULL	
Bdate	date	YES		NULL	
Address	varchar(50)	YES		NULL	
gender	char(1)	YES		NULL	
salary	bigint	YES		NULL	
super_ssn	bigint	YES		NULL	
dno	int	YES	MUL	NULL	

SYNTAX-4:

Used to modify the datatype of the existing column.
ALTER table <table_name> modify COLUMN <column_size>(size);

EXAMPLE:

alter table EMPLOYEE modify gender char(2);
desc EMPLOYEE;

Field	Type	Null	Key	Default	Extra
Filter...	Filter...	Filter...	Filter...	Filter...	Filter...
fname	varchar(20)	NO		NULL	
minit	varchar(20)	YES		NULL	
lname	varchar(20)	YES		NULL	
ssn	bigint	NO	PRI	NULL	
Bdate	date	YES		NULL	
Address	varchar(50)	YES		NULL	
gender	char(2)	YES		NULL	
salary	bigint	YES		NULL	
super_ssn	bigint	YES		NULL	
dno	int	YES	MUL	NULL	

<u>DROP</u>:

Used to delete the entire table structure.

SYNTAX:

DROP table <table_name>;

EXAMPLE:

DROP table WORKS_ON;

<u>TRUNCATE</u>:

Deletes the data inside a table, but not the table itself.

SYNTAX:

TRUNCATE table <table_name>;

EXAMPLE:

TRUNCATE table PROJECT

PRIMARY KEY:

A primary key is used to ensure that data in the specific column is unique. A column cannot have NULL values.

SYNTAX:

col_name datatype PRIMARYKEY;

FOREIGN KEY:

A foreign key constraint is a key used to link two tables together. A foreign key is a field (or collection of fields) in one table that refers to the PRIMARY KEY in another table.

SYNTAX:

<col_name>(datatype) REFERENCES <col_name>(datatype);

EXAMPLE:

1. Creation of departments locations table.

```
CREATE TABLE DEPT_LOCATIONS(
   Dnumber int NOT NULL references DEPARTMENT(Dnumber) ,
   Dlocation varchar(30) NOT NULL,
   primary key(Dnumber,Dlocation)
);
DESC DEPT_LOCATIONS;
```

Field	Type	Null	Key	Default	Extra
Filter...	Filter...	Filter...	Filter...	Filter...	Filter...
Dnumber	int	NO	PRI	NULL	
Dlocation	varchar(30)	NO	PRI	NULL	

2. Creation of project table.

```
CREATE TABLE PROJECT(
   Pname varchar(30) NOT NULL,
   Pnumber int NOT NULL primary key,
   Plocation varchar(30),
   Dnum int NOT NULL references DEPARTMENT(Dnumber)
);
DESC PROJECT;
```

Field	Type	Null	Key	Default	Extra
Filter...	Filter...	Filter...	Filter...	Filter...	Filter...
Pname	varchar(30)	NO		NULL	
Pnumber	int	NO	PRI	NULL	
Plocation	varchar(30)	YES		NULL	
Dnum	int	NO		NULL	

3. Creation of Works_In table:

```
CREATE TABLE WORKS_ON(
   Essn bigint NOT NULL references EMPLOYEE(ssn),
   Pno int NOT NULL references PROJECT(Pnumber),
   Hours Decimal(3,1) NOT NULL,
   primary key(Essn, Pno)
);
DESC WORKS_ON;
```

Field	Type	Null	Key	Default	Extra
Filter...	Filter...	Filter...	Filter...	Filter...	Filter...
Essn	bigint	NO	PRI	NULL	
Pno	int	NO	PRI	NULL	
Hours	decimal(3,1)	NO		NULL	

4. Creation of dependents table:

```
CREATE TABLE DEPENDENT(
   Essn bigint NOT NULL references EMPLOYEE(ssn) ,
   Dependent_name varchar(30) NOT NULL,
   Gender char,
   Bdate date,
   Relationship varchar(8),
   primary key(Essn, Dependent_name)
);
DESC DEPENDENT;
```

Field	Type	Null	Key	Default	Extra
Filter...	Filter...	Filter...	Filter...	Filter...	Filter...
Essn	bigint	NO	PRI	NULL	
Dependent_name	varchar(30)	NO	PRI	NULL	
Gender	char(1)	YES		NULL	
Bdate	date	YES		NULL	
Relationship	varchar(8)	YES		NULL	

<u>CHECK</u>:

The CHECK constraint ensures that all values in a column meet a specific condition.

SYNTAX:

col_name datatype(size) CHECK (condition);

<u>NOT NULL</u>:

The not null constraint in a column means that the column cannot store NULL values

SYNTAX:

col_name datatype(size) NOT NULL;

<u>UNIQUE</u>:

The unique constraint ensures that all values in a column are different. A PRIMARY KEY constraint automatically has a UNIQUE constraint.

SYNTAX:

<col_name>datatype NOT NULL UNIQUE;

EXPERIMENT-3

Aim:

Implementation of

i. DML Commands: INSERT, UPDATE, DELETE.
ii. DCL Commands: COMMIT, ROLLBACK and SAVEPOINT.

INSERT:

INSERT is used to add new records to a database table.

SYNTAX:

INSERT INTO table_name (column1, column2, ...) VALUES (value1, value2, ...);

EXAMPLE:

1. USE 22501A0557;
INSERT INTO EMPLOYEE VALUES

('John','B','Smith',123456789,'1965-01-09','731 Fondren, Houston TX','M',30000,333445555,5),

('Franklin','T','Wong',333445555,'1965-12-08','638 Voss, Houston TX','M',40000,888665555,5),

('Alicia','J','Zelaya',999887777,'1968-01-19','3321 Castle, Spring TX','F',25000,987654321,4),

('Jennifer','S','Wallace',987654321,'1941-06-20','291 Berry, Bellaire TX','F',43000,888665555,4),

('Ramesh','K','Narayan',666884444,'1962-09-15','975 Fire Oak, Humble TX','M',38000,333445555,5),

('Joyce','A','English',453453453,'1972-07-31','5631 Rice, Houston TX','F',25000,333445555,5),

('Ahmad','V','Jabbar',987987987,'1969-03-29','980 Dallas, Houston TX','M',25000,987654321,4),

('James','E','Borg',888665555,'1937-11-10','450 Stone, Houston TX','M',55000,null,1);

SELECT * FROM EMPLOYEE;

Fname	Minit	Lname	ssn	Bdate	Address	Gender	Salary	super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1965-12-08	638 Voss, Houston TX	M	40000	888665555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston TX	F	25000	333445555	5
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble TX	M	38000	333445555	5
James	E	Borg	888665555	1937-11-10	450 Stone, Houston TX	M	55000	NULL	1
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire TX	F	43000	888665555	4
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston TX	M	25000	987654321	4
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring TX	F	25000	987654321	4

2. INSERT INTO DEPARTMENT
   VALUES

   ('Research',5,333445555,'1988-05-22'),

   ('Administration',4,987654321,'1995-01-01'),

   ('Headquarters',1,888665555,'1981-06-19');

   SELECT * FROM DEPARTMENT;

Dname	Dnumber	Mgr_ssn	Mgr_start_date
Filter...	Filter...	Filter...	Filter...
Headquarters	1	888665555	1981-06-19
Administration	4	987654321	1995-01-01
Research	5	333445555	1988-05-22

3. INSERT INTO PROJECT

   VALUES

   ('ProductX',1,'Bellaire',5),

   ('ProductY',2,'Sugarland',5),

   ('ProductZ',3,'Houston',5),

   ('Computerization',10,'Stafford',4),

   ('Reorganization',20,'Houston',1),

   ('Newbenefits',30,'Stafford',4);

   SELECT * FROM PROJECT;

Pname	Pnumber	Plocation	Dnum
Filter...	Filter...	Filter...	Filter...
ProductX	1	Bellaire	5
ProductY	2	Sugarland	5
ProductZ	3	Houston	5
Computerization	10	Stafford	4
Reorganization	20	Houston	1
Newbenefits	30	Stafford	4

4. INSERT INTO WORKS_ON

VALUES

(123456789,1,32.5),

(123456789,2,7.5),

(666884444,3,40.0),

(453453453,1,20.0),

(453453453,2,20.0),

(333445555,2,10.0),

(333445555,3,10.0),

(333445555,10,10.0),

(333445555,20,10.0),

(999887777,30,30.0),

(999887777,10,10.0),

(987987987,10,35.0),

(987987987,30,5.0),

(987654321,30,20.0),

(987654321,20,15.0),

(888665555,20,NULL);

SELECT * FROM WORKS_ON;

Essn	Pno	Hours
Filter...	Filter...	Filter...
123456789	1	32.5
123456789	2	7.5
333445555	2	10.0
333445555	3	10.0
333445555	10	10.0
333445555	20	10.0
453453453	1	20.0
453453453	2	20.0
666884444	3	40.0
888665555	20	NULL
987654321	20	15.0
987654321	30	20.0
987987987	10	35.0
987987987	30	5.0
999887777	10	10.0
999887777	30	30.0

5. INSERT INTO DEPENDENT

   VALUES

   (333445555,'Alice','F','1986-04-04','Daughter'),

   (333445555,'Theodore','M','1983-10-25','Son'),

   (333445555,'Joy','F','1958-05-03','Spouse'),

   (987654321,'Abner','M','1942-02-28','Spouse'),

   (123456789,'Michael','M','1988-01-04','Son'),

   (123456789,'Alice','F','1988-12-30','Daughter'),

   (123456789,'Elizabeth','F','1967-05-05','Spouse');

   SELECT * FROM DEPENDENT;

Essn	Dependent_name	Gender	Bdate	Relationship
Filter...	Filter...	Filter...	Filter...	Filter...
123456789	Alice	F	1988-12-30	Daughter
123456789	Elizabeth	F	1967-05-05	Spouse
123456789	Michael	M	1988-01-04	Son
333445555	Alice	F	1986-04-04	Daughter
333445555	Joy	F	1958-05-03	Spouse
333445555	Theodore	M	1983-10-25	Son
987654321	Abner	M	1942-02-28	Spouse

6. INSERT INTO DEPT_LOCATIONS

   VALUES

   (1,'Houston'),

   (4,'Stafford'),

   (5,'Bellaire'),

   (5,'Sugarland'),

   (5,'Houston');

   SELECT * FROM DEPT_LOCATIONS;

Dnumber	Dlocation
Filter...	Filter...
1	Houston
4	Stafford
5	Bellaire
5	Houston
5	Sugarland

UPDATE:

The UPDATE statement is used to modify the existing column in a table.

SYNTAX:

- UPDATE table_name

  SET column1 = value1, column2 = value2, ...

  WHERE condition;

- UPDATE table_name

  SET column1 = value1;

EXAMPLE:

Creating a temporary table to apply operations of update on it

CREATE TABLE EMPLOYEE1 AS SELECT * FROM EMPLOYEE;

1. UPDATE EMPLOYEE1
   SET Salary='50000';
   SELECT * FROM EMPLOYEE1;

Fname	Minit	Lname	ssn	Bdate	Address	Gender	Salary	super_ssn	Dno
Filter...	Filter...	Filter...	Filter...	Filter...	Filter...	Filter...	Filter...	Filter...	Filt
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston TX	M	50000	333445555	5
Franklin	T	Wong	333445555	1965-12-08	638 Voss, Houston TX	M	50000	888665555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston TX	F	50000	333445555	5
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble TX	M	50000	333445555	5
James	E	Borg	888665555	1937-11-10	450 Stone, Houston TX	M	50000	NULL	1
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire TX	F	50000	888665555	4
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston TX	M	50000	987654321	4
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring TX	F	50000	987654321	4

2. UPDATE EMPLOYEE1

   SET Address='7345 venice Bellaire TX'

   where Ssn=123456789;

   select * from EMPLOYEE1;

Fname	Minit	Lname	ssn	Bdate	Address	Gender	Salary	super_ssn	Dno
Filter...	Filter	Filter...	Filter.	Filter...	Filter...	Filter...	Filter...	Filter...	Filte
John	B	Smith	123456789	1965-01-09	7345 venice Bellaire TX	M	50000	333445555	5
Franklin	T	Wong	333445555	1965-12-08	638 Voss, Houston TX	M	50000	888665555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston TX	F	50000	333445555	5
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble TX	M	50000	333445555	5
James	E	Borg	888665555	1937-11-10	450 Stone, Houston TX	M	50000	NULL	1
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire TX	F	50000	888665555	4
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston TX	M	50000	987654321	4
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring TX	F	50000	987654321	4

3. UPDATE EMPLOYEE1

   SET Salary='55000'

   where Super_ssn=333445555;

   select * from EMPLOYEE1;

Fname	Minit	Lname	ssn	Bdate	Address	Gender	Salary	super_ssn	Dno
Filter.	Filter..	Filter...	Filter	Filter..	Filter...	Filter...	Filter.	Filter...	Filte
John	B	Smith	123456789	1965-01-09	7345 venice Bellaire TX	M	55000	333445555	5
Franklin	T	Wong	333445555	1965-12-08	638 Voss, Houston TX	M	50000	888665555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston TX	F	55000	333445555	5
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble TX	M	55000	333445555	5
James	E	Borg	888665555	1937-11-10	450 Stone, Houston TX	M	50000	NULL	1
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire TX	F	50000	888665555	4
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston TX	M	50000	987654321	4
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring TX	F	50000	987654321	4

<u>DELETE</u>:

SQL DELETE is a basic SQL operation used to delete data in a database. This SQL DELETE operation is important for database size management, data accuracy, and integrity.

SYNTAX :

- DELETE FROM table_name;
- DELETE FROM table_name WHERE some_condition;

EXAMPLE:

1. DELETE
   FROM EMPLOYEE1
   WHERE Fname = 'Alicia';
   SELECT * FROM EMPLOYEE1;

Fname	Minit	Lname	ssn	Bdate	Address	Gender	Salary	super_ssn	Dno
Filter.	Filter	Filter..	Filter..	Filter...	Filter...	Filter...	Filter.	Filter...	Filte
John	B	Smith	123456789	1965-01-09	7345 venice Bellaire TX	M	55000	333445555	5
Franklin	T	Wong	333445555	1965-12-08	638 Voss, Houston TX	M	50000	888665555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston TX	F	55000	333445555	5
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble TX	M	55000	333445555	5
James	E	Borg	888665555	1937-11-10	450 Stone, Houston TX	M	50000	NULL	1
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire TX	F	50000	888665555	4
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston TX	M	50000	987654321	4

2. DELETE

   FROM EMPLOYEE1

   WHERE gender = 'F';

   SELECT * FROM EMPLOYEE1;

Fname	Minit	Lname	ssn	Bdate	Address	Gender	Salary	super_ssn	Dno
Filter..	Filter	Filter...	Filter...	Filter...	Filter...	Filter...	Filter.	Filter...	Filte
John	B	Smith	123456789	1965-01-09	7345 venice Bellaire TX	M	55000	333445555	5
Franklin	T	Wong	333445555	1965-12-08	638 Voss, Houston TX	M	50000	888665555	5
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble TX	M	55000	333445555	5
James	E	Borg	888665555	1937-11-10	450 Stone, Houston TX	M	50000	NULL	1
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston TX	M	50000	987654321	4

3. DELETE

   FROM EMPLOYEE1;

   SELECT * FROM EMPLOYEE1;

Fname	Minit	Lname	ssn	Bdate	Address	Gender	Salary	super_ssn	Dno
Filter..	Filter	Filter..	Filt	Filter	Filter...	Filter...	Filter.	Filter...	Filt
									No data

DCL (Data Control Language)

DCL includes commands such as GRANT and REVOKE which mainly deal with the rights, permissions, and other controls of the database system.

<u>COMMIT</u>:

Saves all changes made during the transaction

SYNTAX:

COMMIT;

EXAMPLE:

START TRANSACTION;

UPDATE EMPLOYEE

SET Salary = Salary + 5000

WHERE ssn = '123456789';

COMMIT;

SELECT * FROM EMPLOYEE WHERE Ssn = '123456789';

Fname	Minit	Lname	ssn	Bdate	Address	Gender	Salary	super_ssn	D...
Filter...	Filter.	Filter...	Filter...	Filter...	Filter...	Filter...	Filter.	Filter...	Filt
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston TX	M	35000	333445555	5

ROLLBACK:

Undoes all changes made during the transaction

SYNTAX:

ROLLBACK;

EXAMPLE:

START TRANSACTION;

INSERT INTO EMPLOYEE

VALUES ('Sarah', 'M', 'Green', 777889999, '1975-09-17', '456 Oak, Houston TX', 'F', 32000, 987654321, 4);

SELECT * FROM EMPLOYEE WHERE Ssn = 777889999;

Fname	Minit	Lname	ssn	Bdate	Address	Gender	Salary	super_ssn	Dno
Filter.	Filter	Filter...	Filter	Filter..	Filter...	Filter...	Filter..	Filter...	Filt
Sarah	M	Green	777889999	1975-09-17	456 Oak, Houston TX	F	32000	987654321	4

ROLLBACK;

Fname ↑	Minit	Lname	ssn	Bdate	Address	Gender	Salary	super_ssn	Dno
Filter..	Filter	Filter..	Filt	Filter	Filter...	Filter...	Filter.	Filter...	Filt
									No data

SAVEPOINT:

Creates a savepoint within the current transaction

SYNTAX:

SAVEPOINT savepoint_name;

EXAMPLE:

START TRANSACTION;

INSERT INTO EMPLOYEE

VALUES ('Lucas', 'W', 'White', 666777888, '1990-02-23', '789 Pine, Houston TX', 'M', 31000, 333445555, 5);

SAVEPOINT sp1;

SELECT * FROM EMPLOYEE WHERE ssn = 666777888;

Fname	Minit	Lname	ssn	Bdate	Address	Gender	Salary	super_ssn	Dno
Filter...	Filter.	Filter..	Filter..	Filter..	Filter...	Filter...	Filter.	Filter...	Filt
Lucas	W	White	666777888	1990-02-23	789 Pine, Houston TX	M	31000	333445555	5

INSERT INTO EMPLOYEE

VALUES ('Olivia', 'D', 'Davis', 111223344, '1985-06-14', '135 Elm, Houston TX', 'F', 29000, 987654321, 4);

select * from EMPLOYEE;

Fname	Minit	Lname	ssn	Bdate	Address	Gender	Salary	super_ssn	Dno
Filter...	Filter	Filter.	Filter.	Filter...	Filter...	Filter...	Filter.	Filter...	Filte
Olivia	D	Davis	111223344	1985-06-14	135 Elm, Houston TX	F	29000	987654321	4
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston ...	M	35000	333445555	5
Franklin	T	Wong	333445555	1965-12-08	638 Voss, Houston TX	M	40000	888665555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston TX	F	25000	333445555	5
Lucas	W	White	666777888	1990-02-23	789 Pine, Houston TX	M	31000	333445555	5
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble TX	M	38000	333445555	5
James	E	Borg	888665555	1937-11-10	450 Stone, Houston TX	M	55000	NULL	1
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire TX	F	43000	888665555	4
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston TX	M	25000	987654321	4
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring TX	F	25000	987654321	4

ROLLBACK TO sp1;
select * from EMPLOYEE;

Fname	Minit	Lname	ssn	Bdate	Address	Gender	Salary	super_ssn	Dno
Filter..	Filter	Filter..	Filter.	Filter...	Filter...	Filter...	Filter..	Filter...	Filt
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston ...	M	35000	333445555	5
Franklin	T	Wong	333445555	1965-12-08	638 Voss, Houston TX	M	40000	888665555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston TX	F	25000	333445555	5
Lucas	W	White	666777888	1990-02-23	789 Pine, Houston TX	M	31000	333445555	5
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble TX	M	38000	333445555	5
James	E	Borg	888665555	1937-11-10	450 Stone, Houston TX	M	55000	NULL	1
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire TX	F	43000	888665555	4
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston TX	M	25000	987654321	4
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring TX	F	25000	987654321	4

COMMIT;

EXPERIMENT-4

Aim:

Retrieving the data from a single table using

i) SELECT statement
ii) SELECT statement with where clause (Comparison Operators, AND, OR, NOT, IN, BETWEEN, LIKE)
iii) ORDER BY clause (sort by column name)
iv) LIMIT clause

SELECT:

A basic SELECT statement retrieves all rows and columns from a specified table.

SYNTAX:

SELECT * FROM table_name;

EXAMPLE:

SELECT * FROM EMPLOYEE;

Fname	Minit	Lname	ssn	Bdate	Address	Gender	Salary	super_ssn	Dno
Filter.	Filter.	Filter.	Filter..	Filter...	Filter...	Filter...	Filter.	Filter...	Filt
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston ...	M	35000	333445555	5
Franklin	T	Wong	333445555	1965-12-08	638 Voss, Houston TX	M	40000	888665555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston TX	F	25000	333445555	5
Lucas	W	White	666777888	1990-02-23	789 Pine, Houston TX	M	31000	333445555	5
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble TX	M	38000	333445555	5
James	E	Borg	888665555	1937-11-10	450 Stone, Houston TX	M	55000	NULL	1
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire TX	F	43000	888665555	4
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston TX	M	25000	987654321	4
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring TX	F	25000	987654321	4

SELECT Statement with WHERE Clause:

You can filter data using the WHERE clause. Common comparison operators include:

- = : Equal to
- != or <> : Not equal to
- > : Greater than
- < : Less than
- >= : Greater than or equal to
- <= : Less than or equal to

SYNTAX:

SELECT * FROM table_name WHERE condition;

EXAMPLE:

1. **Using Comparison Operators**: Retrieve employees with a salary greater than 30,000:

   SELECT * FROM EMPLOYEE WHERE Salary > 30000;

Fname	Minit	Lname	ssn	Bdate	Address	Gender	Salary	super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston TX	M	35000	333445555	5
Franklin	T	Wong	333445555	1965-12-08	638 Voss, Houston TX	M	40000	888665555	5
Lucas	W	White	666777888	1990-02-23	789 Pine, Houston TX	M	31000	333445555	5
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble TX	M	38000	333445555	5
James	E	Borg	888665555	1937-11-10	450 Stone, Houston TX	M	55000	NULL	1
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire TX	F	43000	888665555	4

2. **Using AND, OR**: Retrieve male employees who earn more than 30,000 or female employees who earn more than 40,000:
   SELECT * FROM EMPLOYEE WHERE (Gender = 'M' AND Salary > 30000) OR (Gender = 'F' AND Salary > 40000);

Fname	Minit	Lna...	ssn	Bd...	Address	Ge...	S..	s...	Dno
John	B	Smith	1234567...	1965-01...	731 Fondren, Houst...	M	35000	33344...	5
Franklin	T	Wong	3334455...	1965-12...	638 Voss, Houston TX	M	40000	88866...	5
Lucas	W	White	6667778...	1990-02...	789 Pine, Houston TX	M	31000	33344...	5
Ramesh	K	Narayan	6668844...	1962-09...	975 Fire Oak, Humb...	M	38000	33344...	5
James	E	Borg	8886655...	1937-11...	450 Stone, Houston...	M	55000	NULL	1
Jennifer	S	Wallace	9876543...	1941-06...	291 Berry, Bellaire TX	F	43000	88866...	4

3. **Using NOT**: Retrieve employees who are not in department 5:
   SELECT * FROM EMPLOYEE WHERE NOT Dno = 5;

Fname	Minit	Lname	ssn	Bdate	Address	Gender	Salary	super_ssn	Dno
James	E	Borg	888665555	1937-11-10	450 Stone, Houston TX	M	55000	NULL	1
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire TX	F	43000	888665555	4
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston TX	M	25000	987654321	4
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring TX	F	25000	987654321	4

4. **Using IN**: Retrieve employees who work in departments 1, 4, or 5:
   SELECT * FROM EMPLOYEE WHERE Dno IN (1, 4);

Fname	Minit	Lname	ssn	Bdate	Address	Gender	Salary	super_ssn	Dno
James	E	Borg	888665555	1937-11-10	450 Stone, Houston TX	M	55000	NULL	1
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire TX	F	43000	888665555	4
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston TX	M	25000	987654321	4
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring TX	F	25000	987654321	4

5. **Using BETWEEN**: Retrieve employees born between 1960 and 1970:
   SELECT * FROM EMPLOYEE WHERE Bdate BETWEEN '1960-01-01' AND '1970-12-31';

Fname	Minit	Lname	ssn	Bdate	Address	Gender	Salary	super_ssn	Dno
John	B	Smith	12345...	1965-01-...	731 Fondren, Houston TX	M	35000	333445555	5
Franklin	T	Wong	33344...	1965-12-...	638 Voss, Houston TX	M	40000	888665555	5
Ramesh	K	Narayan	66688...	1962-09-...	975 Fire Oak, Humble TX	M	38000	333445555	5
Ahmad	V	Jabbar	98798...	1969-03-...	980 Dallas, Houston TX	M	25000	987654321	4
Alicia	J	Zelaya	99988...	1968-01-...	3321 Castle, Spring TX	F	25000	987654321	4

6. **Using LIKE**: Retrieve employees whose last name starts with 'S':
   SELECT * FROM EMPLOYEE WHERE Lname LIKE 'S%';

Fname	Minit	Lname	ssn	Bdate	Address	Gender	Salary	super_ssn	Dno
Filter...	Filter.	Filter...	Filter.	Filter..	Filter...	Filter...	Filter.	Filter...	Filt
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston TX	M	35000	333445555	5

<u>ORDER BY</u>:

You can sort the result set using the ORDER BY clause. The default sorting order is ascending (ASC), but you can also sort in descending order (DESC).

SYNTAX:

SELECT * FROM table_name ORDER BY column_name [ASC|DESC];

EXAMPLE:

1. Sort employees by salary in ascending order:
   SELECT * FROM EMPLOYEE ORDER BY Salary ASC;

Fname	Minit	Lname	ssn	Bdate	Address	Gender	Salary	super_ssn	Dno
Filter...	Filter	Filter..	Filter..	Filter...	Filter...	Filter..	Filter.	Filter...	Filt
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston TX	M	25000	987654321	4
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring TX	F	25000	987654321	4
Lucas	W	White	666777888	1990-02-23	789 Pine, Houston TX	M	31000	333445555	5
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston TX	M	35000	333445555	5
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble TX	M	38000	333445555	5
Franklin	T	Wong	333445555	1965-12-08	638 Voss, Houston TX	M	40000	888665555	5
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire TX	F	43000	888665555	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston TX	M	55000	NULL	1

2. Sort employees by last name in descending order:
   SELECT * FROM EMPLOYEE ORDER BY Lname DESC;

Fname	Minit	Lname	ssn	Bdate	Address	Gender	Salary	super_ssn	Dno
Filter.	Filter.	Filter...	Filter...	Filter...	Filter...	Filter...	Filter...	Filter...	Filt
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring TX	F	25000	987654321	4
Franklin	T	Wong	333445555	1965-12-08	638 Voss, Houston TX	M	40000	888665555	5
Lucas	W	White	666777888	1990-02-23	789 Pine, Houston TX	M	31000	333445555	5
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire TX	F	43000	888665555	4
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston TX	M	35000	333445555	5
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble TX	M	38000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston TX	M	25000	987654321	4
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston TX	F	25000	333445555	5
James	E	Borg	888665555	1937-11-10	450 Stone, Houston TX	M	55000	NULL	1

LIMIT:

The LIMIT clause is used to specify the number of records you want to retrieve from the result set.

SYNTAX:

SELECT * FROM table_name LIMIT number_of_rows;

EXAMPLE:

1. Retrieve the first 5 employees:

   SELECT * FROM EMPLOYEE LIMIT 5;

Fname	Minit	Lname	ssn	Bdate	Address	Gender	Salary	super_ssn	Dno
Filter...	Filter	Filter..	Filter...	Filter...	Filter...	Filter...	Filter.	Filter...	Filte
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston TX	M	35000	333445555	5
Franklin	T	Wong	333445555	1965-12-08	638 Voss, Houston TX	M	40000	888665555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston TX	F	25000	333445555	5
Lucas	W	White	666777888	1990-02-23	789 Pine, Houston TX	M	31000	333445555	5
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble TX	M	38000	333445555	5

2. Retrieve the top 3 highest-paid employees:
   SELECT * FROM EMPLOYEE ORDER BY Salary DESC LIMIT 3;

Fname	Minit	Lname	ssn	Bdate	Address	Gender	Salary	super_ssn	Dno
Filter..	Filter.	Filter...	Filter..	Filter..	Filter...	Filter...	Filter.	Filter...	Filte
James	E	Borg	888665555	1937-11-10	450 Stone, Houston TX	M	55000	NULL	1
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire TX	F	43000	888665555	4
Franklin	T	Wong	333445555	1965-12-08	638 Voss, Houston TX	M	40000	888665555	5

EXPERIMENT-5

Aim:

Implementation of

i. Strings
ii. Numeric
iii. Date
iv. Time
v. Other Functions

String Functions:

These functions allow you to manipulate string values in SQL. Each query performs a specific action like concatenation, trimming, reversing strings, etc.

EXAMPLE:

1. **Concatenate Strings**:
   Adds two or more expressions together.
   SELECT CONCAT('My','S','QL') AS CONCAT;

CONCAT
abc Filter...
MySQL

2. **Insert a Substring:**
   Inserts a string within a string at the specified position and for a certain number of characters.
   SELECT INSERT('Quadratic', 3, 4, 'What') ISRT;

ISRT
abc Filter...
QuWhattic

3. **Find Substring Position:**
   Returns the position of the first occurrence of a string in another string.
   SELECT INSTR('foobarbar', 'bar') INSTR;

INSTR
abc Filter...
4

4. **Convert to Lowercase:**
   Converts a string to lower-case.
   SELECT LOWER('QUADRATICALLY') LOWER;

LOWER
abc Filter...
quadratically

5. **Find String Length:**
   Returns the length of a string. (in bytes)
   SELECT LENGTH('text') LEN;

LEN
abc Filter...
4

6. **Left Pad String:**
   Left-pads a string with another string, to a certain length.
   SELECT LPAD('hi', 4, '??') LPAD;

LPAD
abc Filter...
??hi

7. **Right Pad String:**
   Right-pads a string with another string, to a certain length.
   SELECT RPAD('hi', 4, '??') RPAD;

RPAD
abc Filter...
hi??

8. **Trim Leading Spaces:**
   Removes leading spaces from a string.
   SELECT LTRIM(' barbar') LTRIM;

LTRIM
abc Filter...
barbar

9. **Trim Trailing Spaces:**
   Removes trailing spaces from a string.
   SELECT RTRIM('barbar   ') RTRIM57;

RTRIM57
abc Filter...
barbar

10. **Reverse String:**
    Reverses a string and returns the result.
    SELECT REVERSE('abcd') REVERSE57;

REVERSE57
abc Filter...
dcba

11. **String Comparison (Equal):**
    Compares two strings. (returns 0 if s1=s2,1 if s1>s2 , -1 if s1<s2)
    SELECT STRCMP('MOHD', 'MOHD') STRCMP57;

STRCMP57
abc Filter...
0

12. **String Comparison (Not Equal):**
    SELECT STRCMP('AMOHD', 'MOHD') STRCMP57;

STRCMP57
abc Filter...
-1

13. **Substring Extraction:**
    Return a substring from string str starting at position pos.
    SELECT SUBSTRING('Quadratically', 5) SUBSTRING57;

SUBSTRING57
abc Filter...
ratically

14. **Substring Extraction (with Length):**
    Return a substring len characters long from string str, starting at position pos.
    SELECT SUBSTRING('Quadratically', 5, 6) SUBSTRING57;

SUBSTRING57
abc Filter...
ratica

<u>Date & Time Functions</u>:

Date and time functions are used to manipulate and format date and time values.

DATE_FORMAT(date,format); Formats the date value according to the format string.

The following specifiers may be used in the format string. The .%. character is required before format specifier characters.

- %a Abbreviated weekday name (Sun..Sat)
- %b Abbreviated month name (Jan..Dec)
- %c Month, numeric (0..12)
- %D Day of the month with English suffix (0th, 1st, 2nd, 3rd, .)
- %d Day of the month, numeric (00..31)
- %j Day of year (001..366)
- %M Month name (January..December)
- %m Month, numeric (00..12)
- %W Weekday name (Sunday..Saturday)
- %w Day of the week (0 = Sunday..6 = Saturday)

SELECT func_name (date/time);

- HOUR Returns the hour part for a given date
- MONTH Returns the month part for a given date
- MONTHNAME Returns the name of the month for a given date
- WEEK Returns the week number for a given date
- WEEKDAY Returns the weekday number for a given date
- WEEKOFYEAR Returns the week number for a given date
- YEAR Returns the year part for a given date
- YEARWEEK Returns the year and week number for a given date

EXAMPLE:

1. **Get Current Date**:
   Returns the current date as a value in 'YYYY-MM-DD' format
   SELECT CURDATE() CURDATE57;

CURDATE57
abc Filter...
2024-09-16

2. **Get Current Time**:
   Returns the current time as a value in 'HH:MM:SS format.
   SELECT CURTIME() CURTIME57;

CURTIME57
abc Filter...
09:09:06

3. **Get Day of the Month**:
   Returns the day of the month for a given date.
   SELECT DAYOFMONTH('2005-07-25') DAYOFMONTH57;

DAYOFMONTH57
abc Filter...
25

4. **Get Day of the Week**:
   Returns the weekday index for a given date (1 = Sunday, 2 = Monday, ., 7 = Saturday).
   SELECT DAYOFWEEK('2005-07-25') DAYOFWEEK57;

DAYOFWEEK57
abc Filter...
2

5. **Get Day of the Year**:
   Returns the day of the year for a given date, in the range 1 to 366.
   SELECT DAYOFYEAR('2005-07-25') DAYOFYEAR57;

DAYOFYEAR57
abc Filter...
206

6. **Format Date to Custom String**:
   Formats the date value according to the format string.
   SELECT DATE_FORMAT('2024-07-31 15:45:00', '%a %M %D') DATE_FORMAT57;

DATE_FORMAT57
abc Filter...
Mon July 25th

7. **Format Date to Custom Numeric String**:
   SELECT DATE_FORMAT('2024-07-31 15:45:00', '%m-%d-%Y') DATE_FORMAT57;

DATE_FORMAT57
abc Filter...
07-31-2024

8. **Format Date with Day of the Year and Week Number**:
   SELECT DATE_FORMAT('2024-07-31 15:45:00', '%j %W %u') DATE_FORMAT57;

DATE_FORMAT57
abc Filter...
213 Wednesday 31

9. **Get Week, Weekday, and Hour**:
   SELECT CONCAT(WEEK('2024-07-31 15:45:00'), ' ', WEEKDAY('2024-07-31 15:45:00'), ' ', HOUR('2024-07-31 15:45:00')) Time57;

Time57
abc Filter...
30 2 15

10. **Get Year, Month Name, and Year Week**:

    SELECT CONCAT(YEAR('2024-07-31 15:45:00'), ' ', MONTHNAME('2024-07-31 15:45:00'), ' ', YEARWEEK('2024-07-31 15:45:00')) Time57;

Time57
abc Filter...
2024 July 202430

<u>Math Functions</u>:

Math functions allow you to perform calculations and operations on numeric data.

EXAMPLE:

1. **Find the Greatest Value**:
   Returns the largest value in a number set.
   SELECT GREATEST(10000, 2) GREATEST57;

GREATEST57
abc Filter...
10000

2. **Find the Least Value**:
   Returns the smallest value in a number set.
   SELECT LEAST(10000, 123, -123123) LEAST57;

LEAST57
abc Filter...
-123123

3. **Floor Division (Rounds Down)**:
   Returns the closest integer value less than x.
   SELECT FLOOR(10/3) FLOOR57;

FLOOR57
abc Filter...
3

4. **Ceiling Division (Rounds Up)**:
   Returns the closest integer value greater than x.
   SELECT CEILING(10/3) CEILING57;

CEILING57
abc Filter...
4

5. **Round a Number**:
   Returns x rounded off to the closest integer, with y decimal places.
   SELECT ROUND(10.523132) ROUND57;

ROUND57
abc Filter...
11

6. **Truncate a Number to 3 Decimal Places**:
   Returns the result of truncating x to y decimal places.
   SELECT TRUNCATE(1.785927343, 3);

TRUNCATE57
abc Filter...
1.785

7. **Find Absolute Value**:
   Returns the absolute value of the x.
   SELECT ABS(-124);

ABS57
abc Filter...
124

8. **Square Root of a Number**:
   Returns square root of the x.
   SELECT SQRT(64) SQRT57;

SQRT57
abc Filter...
8

9. **Power of a Number**:
   Returns x power y.
   SELECT POW(2, 2);

POW57
abc Filter...
4

10. **Exponential Function**:
    Returns the e power x.
    SELECT EXP(0) EXP57;

EXP57
abc Filter...
1

11. **Modulus Operation**:
    Returns the modulo(Remainder) of x/y.
    SELECT MOD(5, 2) MOD57;

MOD57
abc Filter...
1

EXPERIMENT-6

Aim:

Implementation of

i. Aggregate Functions
ii. GROUP BY and Having Clause
iii. ROLLUP Operator

Aggregate Functions:

Aggregate functions in SQL perform calculations on a set of values and return a single value. These are commonly used with the `GROUP BY` clause to group rows that have the same values in specified columns.

EXAMPLE:

1. **SUM:**
   Calculates the total sum of a numeric column.
   SELECT SUM(salary) AS Total_Salary57 FROM EMPLOYEE;

Total_Salary57
abc Filter...
317000

2. **AVG:**
   Calculates the average value of a numeric column.
   SELECT AVG(salary) AS Average_Salary57 FROM employee;

Average_Salary57
abc Filter...
35222.2222

3. **COUNT:**
   Returns the number of rows in a result set.
   SELECT COUNT(ssn) AS Total_Employees57 FROM employee;

Total_Employees57
abc Filter...
9

4. **MAX:**
   Returns the highest value in a column.
   SELECT MAX(salary) AS Max_Salary57 FROM employee;

Max_Salary57
abc Filter...
55000

5. **MIN:**
   Returns the lowest value in a column.
   SELECT MIN(salary) AS Min_Salary57 FROM employee;

Min_Salary57
abc Filter...
25000

GROUP BY:

The GROUP BY clause groups rows that have the same values into summary rows, like "total salary" or "number of employees per department."

SYNTAX:

SELECT column_name, AGGREGATE_FUNCTION(column_name)

FROM table_name

GROUP BY column_name;

EXAMPLE:

1. **Finding out average salary of each department**
   SELECT Dno, AVG(Salary) AS AvgSalary
   FROM EMPLOYEE
   GROUP BY Dno;

Dno	AvgSalary57
abc Filter...	abc Filter...
5	33800.0000
1	55000.0000
4	31000.0000

2. **Count the number of employees in each department**
   SELECT dno, COUNT(*) AS Size57
   FROM EMPLOYEE
   GROUP BY dno
   ORDER BY dno;

dno	Size57
abc Filter...	abc Filter...
1	1
4	3
5	5

3. **For each department retrieve the department no, the no of employees in the department and their average salary**

```
SELECT dno, COUNT(*) as employees57, AVG(salary) as avg_salary57
FROM EMPLOYEE
GROUP BY dno
ORDER BY dno;
```

dno	employees57	avg_salary57
Filter...	Filter...	Filter...
1	1	55000.0000
4	3	31000.0000
5	5	33800.0000

4. **For each project retrieve the project no, project name and the no of employees working on that project**

```
SELECT pnumber, pname, count(essn) as workers57
FROM PROJECT, WORKS_ON
WHERE pnumber = pno
GROUP BY pnumber
ORDER BY pnumber;
```

pnumber	pname	workers57
Filter...	Filter...	Filter...
1	ProductX	2
2	ProductY	3
3	ProductZ	2
10	Computerization	3
20	Reorganization	3
30	Newbenefits	3

5. **For each project retrieve the project no, name and the no of employees from department no = 5 who work on the project**

```
SELECT pnumber, pname, count(essn) as workers57
FROM PROJECT P, WORKS_ON W
WHERE dnum = 5 and pnumber = pno
GROUP BY pnumber
ORDER BY pnumber;
```

pnumber	pname	workers57
Filter...	Filter...	Filter...
1	ProductX	2
2	ProductY	3
3	ProductZ	2

HAVING:

The HAVING clause is used to filter groups based on a condition. It is used with the GROUP BY clause.

SYNTAX:

```
SELECT column_name, AGGREGATE_FUNCTION(column_name)
FROM table_name
GROUP BY column_name
HAVING condition;
```

EXAMPLE:

1. **Departments with Average Salary is greater than 32000**

```
SELECT Dno, AVG(Salary) AS AvgSalary57
FROM EMPLOYEE
GROUP BY Dno
HAVING AVG(Salary) > 32000;
```

Dno	AvgSalary57
Filter...	Filter...
5	33800.0000
1	55000.0000

2. **For each Project on which more than 2 employees work, retrieve project no, project name and the no of employees who work on the project**

```
SELECT pnumber, pname, COUNT(essn) AS workers57
FROM PROJECT P
JOIN WORKS_ON W ON P.pnumber = W.pno
GROUP BY pnumber, pname
HAVING COUNT(essn) > 2
ORDER BY pnumber;
```

pnumber	pname	workers57
Filter...	Filter...	Filter...
2	ProductY	3
10	Computerization	3
20	Reorganization	3
30	Newbenefits	3

3. **Count the total number of employees whose salary exceed 40,000 in each department but only for the department where more than 5 employees work**
   SELECT dno, COUNT(*) AS employees57
   FROM EMPLOYEE
   WHERE salary > 25000
   GROUP BY dno
   HAVING COUNT(*) > 2;

dno	employees57
Filter...	Filter...
5	4

<u>ROLLUP Operator</u>:

The ROLLUP operator is used to perform hierarchical aggregation and generate subtotals and grand totals in a single query.

SYNTAX:

SELECT column1, column2, AGGREGATE_FUNCTION(column3)

FROM table_name

GROUP BY column1, column2 WITH ROLLUP;

EXAMPLE:

1. Simple Query using Rollup
   SELECT Dno, Gender, COUNT(*) AS NumEmployees57
   FROM EMPLOYEE
   GROUP BY Dno, Gender WITH ROLLUP;

Dno	Gender	NumEmployees57
Filter...	Filter...	Filter...
1	M	1
1	NULL	1
4	F	2
4	M	1
4	NULL	3
5	F	1
5	M	4
5	NULL	5
NULL	NULL	9

2. **Total Salary of Employees in each Department**
   SELECT Dno, SUM(salary) AS Total_Salary57
   FROM employee
   GROUP BY Dno WITH ROLLUP;

Dno	Total_Salary57
Filter...	Filter...
1	55000
4	93000
5	169000
NULL	317000

3. **Total Salary of each department and job**
   SELECT dno, ssn, SUM(salary) AS Total_Salary
   FROM employee
   GROUP BY dno, ssn WITH ROLLUP;

dno	ssn	Total_Salary
Filter...	Filter...	Filter...
1	888665555	55000
1	NULL	55000
4	987654321	43000
4	987987987	25000
4	999887777	25000
4	NULL	93000
5	123456789	35000
5	333445555	40000
5	453453453	25000
5	666777888	31000
5	666884444	38000
5	NULL	169000
NULL	NULL	317000

EXPERIMENT-7

Aim:

Implementation of

i. INNER JOIN
ii. OUTER JOIN
iii. USING Clause
iv. NATURAL JOIN

JOIN:

A **JOIN** in SQL is used to combine rows from two or more tables based on a related column between them. The purpose of a join is to fetch data from multiple tables that share a logical relationship.

Types of JOINS:

1. **INNER JOIN**: Returns rows when there is a match in both tables.
2. **LEFT JOIN (or LEFT OUTER JOIN)**: Returns all rows from the left table, and matched rows from the right table. If there is no match, the result is NULL on the right side.
3. **RIGHT JOIN (or RIGHT OUTER JOIN)**: Returns all rows from the right table, and matched rows from the left table. If there is no match, the result is NULL on the left side.
4. **FULL OUTER JOIN**: Returns all rows when there is a match in either table. Rows that do not have a match in one of the tables are returned with NULL values.

SYNTAX:

```
SELECT column_list

FROM table1

JOIN_TYPE table2

ON table1.common_column = table2.common_column;
```

EXAMPLE:

```
SELECT Customers.CustomerName, Orders.Product

FROM Customers

INNER JOIN Orders ON Customers.CustomerID = Orders.CustomerID;
```

INNER JOIN:

The INNER JOIN returns only the rows that have matching values in both tables. If no match is found, those rows are excluded from the result set.

EXAMPLE:

1. **Joining EMPLOYEE and DEPARTMENT on Dno and Dnumber:**

   SELECT E.Fname, E.Lname, D.Dname

   FROM EMPLOYEE E

   INNER JOIN DEPARTMENT D ON E.Dno = D.Dnumber;

Fname	Lname	Dname
Filter...	Filter...	Filter...
John	Smith	Research
Franklin	Wong	Research
Joyce	English	Research
Lucas	White	Research
Ramesh	Narayan	Research
James	Borg	Headquarters
Jennifer	Wallace	Administration
Ahmad	Jabbar	Administration
Alicia	Zelaya	Administration

OUTER JOIN:

There are three types of outer joins: LEFT JOIN, RIGHT JOIN, and FULL OUTER JOIN. They include rows from one or both tables even if there are no matches.

EXAMPLE:

1. **LEFT JOIN:** The LEFT JOIN returns all records from the left table (EMPLOYEE), and the matched records from the right table (DEPARTMENT). If no match is found, NULL values are returned for the right table's columns.

   SELECT E.Fname, E.Lname, D.Dname

   FROM EMPLOYEE E

   LEFT JOIN DEPARTMENT D ON E.Dno = D.Dnumber;

Fname	Lname	Dname
Filter...	Filter...	Filter...
John	Smith	Research
Franklin	Wong	Research
Joyce	English	Research
Lucas	White	Research
Ramesh	Narayan	Research
James	Borg	Headquarters
Jennifer	Wallace	Administration
Ahmad	Jabbar	Administration
Alicia	Zelaya	Administration

2. **RIGHT JOIN:** The RIGHT JOIN returns all records from the right table (DEPARTMENT), and the matched records from the left table (EMPLOYEE). If no match is found, NULL values are returned for the left table's columns.
   SELECT E.Fname, E.Lname, D.Dname
   FROM EMPLOYEE E
   RIGHT JOIN DEPARTMENT D ON E.Dno = D.Dnumber;

Fname	Lname	Dname
abc Filter...	abc Filter...	abc Filter...
James	Borg	Headquarters
Alicia	Zelaya	Administration
Ahmad	Jabbar	Administration
Jennifer	Wallace	Administration
Ramesh	Narayan	Research
Lucas	White	Research
Joyce	English	Research
Franklin	Wong	Research
John	Smith	Research

<u>USING Clause</u>:

The USING clause is used to specify the column to join on when the column names in both tables are the same.

EXAMPLE:

1. **Joining EMPLOYEE and DEPARTMENT on Dno:**

   ALTER TABLE DEPARTMENT CHANGE COLUMN Dnumber Dno INT;

   SELECT E.Fname, E.Lname, D.Dname

   FROM EMPLOYEE E

   INNER JOIN DEPARTMENT D USING (Dno);

Fname	Lname	Dname
abc Filter...	abc Filter...	abc Filter...
John	Smith	Research
Franklin	Wong	Research
Joyce	English	Research
Lucas	White	Research
Ramesh	Narayan	Research
James	Borg	Headquarters
Jennifer	Wallace	Administration
Ahmad	Jabbar	Administration
Alicia	Zelaya	Administration

NATURAL JOIN:

The NATURAL JOIN automatically joins two tables based on all columns with the same name and data type in both tables.

Example:

1. **Natural Join EMPLOYEE and DEPARTMENT**

   SELECT E.Fname, E.Lname, D.Dname

   FROM EMPLOYEE E

   NATURAL JOIN DEPARTMENT D;

Fname	Lname	Dname
Filter...	Filter...	Filter...
John	Smith	Research
Franklin	Wong	Research
Joyce	English	Research
Lucas	White	Research
Ramesh	Narayan	Research
James	Borg	Headquarters
Jennifer	Wallace	Administration
Ahmad	Jabbar	Administration
Alicia	Zelaya	Administration

EXPERIMENT-8

Aim:

Implementation of SUB/SUMMARY Queries using

i. IN
ii. ANY
iii. SOME
iv. ALL
v. EXISTS
vi. NOT EXISTS

SUB QUERIES:

A subquery is a query nested inside another query. It is used to perform operations that require multiple steps or to filter results based on complex conditions. Subqueries can return a single value, a single row, or multiple rows, and can be used in various SQL clauses such as SELECT, WHERE, and FROM. They are also known as inner queries or nested queries.

SYNTAX:

A subquery can appear in several SQL clauses and has different forms depending on where it is used. Here's a general overview:

1. **In the SELECT Clause:**

   SELECT column1, (SELECT sub_column FROM sub_table WHERE condition) AS alias_name

   FROM main_table;

2. **In the WHERE Clause:**
   SELECT column1
   FROM main_table
   WHERE column2 = (SELECT sub_column FROM sub_table WHERE condition);
3. **In the FROM Clause:**
   SELECT column1
   FROM (SELECT sub_column1, sub_column2 FROM sub_table WHERE condition) AS sub_query
   WHERE condition;
4. **In the HAVING Clause:**
   SELECT column1, COUNT(*)
   FROM main_table
   GROUP BY column1
   HAVING COUNT(*) > (SELECT AVG(count_column) FROM sub_table WHERE condition);

EXAMPLE:

1. **Subquery in SELECT Clause:**

```
-- Find the employee with the highest salary
SELECT Fname, Lname, Salary,
   (SELECT MAX(Salary) FROM EMPLOYEE) AS Highest_Salary
FROM EMPLOYEE;
```

Fname	Lname	Salary	Highest_Salary
abc Filter...	abc Filter...	abc Filter...	abc Filter...
John	Smith	35000	55000
Franklin	Wong	40000	55000
Joyce	English	25000	55000
Lucas	White	31000	55000
Ramesh	Narayan	38000	55000
James	Borg	55000	55000
Jennifer	Wallace	43000	55000
Ahmad	Jabbar	25000	55000
Alicia	Zelaya	25000	55000

2. **Subquery in WHERE Clause:**

```
-- Find employees who work on project number 1
SELECT Fname, Lname
FROM EMPLOYEE
WHERE ssn IN (
   SELECT Essn
   FROM WORKS_ON
   WHERE Pno = 1
);
```

Fname	Lname
abc Filter...	abc Filter...
John	Smith
Joyce	English

3. **Subquery in FROM Clause:**

```
-- Find the average salary of employees in departments located in 'Houston'
SELECT AVG(Salary) AS Avg_Salary57
FROM EMPLOYEE
WHERE Dno IN (
   SELECT Dno
   FROM dept_locations dl
   WHERE dl.Dlocation = 'Houston' and dl.Dnumber = EMPLOYEE.Dno
);
```

Avg_Salary
abc Filter...
37333.3333

4. **Subquery in HAVING Clause:**

```
-- Find departments where the total number of employees exceeds the average number of employees across all departments

SELECT Dname, COUNT(*) AS Number_Of_Employees

FROM EMPLOYEE

JOIN DEPARTMENT ON EMPLOYEE.Dno = DEPARTMENT.Dno

GROUP BY Dname

HAVING COUNT(*) > (

   SELECT AVG(Number_Of_Employees)

   FROM (

      SELECT Dno, COUNT(*) AS Number_Of_Employees

      FROM EMPLOYEE

      GROUP BY Dno

   ) AS Department_Employee_Count

);
```

Dname	Number_Of_Em...
abc Filter...	abc Filter...
Research	5

IN:

The IN operator is used to filter records based on a set of values. It is often used in subqueries.

EXAMPLE:

1. **Find employees who work on projects located in 'Houston'.**

```
SELECT Fname, Lname
FROM EMPLOYEE
WHERE ssn IN (
  SELECT Essn
  FROM WORKS_ON
  WHERE Pno IN (
    SELECT Pnumber
    FROM PROJECT
    WHERE Plocation = 'Houston'
  )
);
```

Fname	Lname
abc Filter...	abc Filter...
Franklin	Wong
Ramesh	Narayan
James	Borg
Jennifer	Wallace

ANY:

The ANY operator is used to compare a value to any value in another result set.

EXAMPLE:

Find employees whose salary is greater than the salary of any employee working on project number 10.

```
SELECT Fname, Lname, Salary
FROM EMPLOYEE
WHERE Salary > ANY (
  SELECT Salary
  FROM EMPLOYEE E
```

```
    JOIN WORKS_ON W ON E.ssn = W.Essn
    WHERE W.Pno = 10
);
```

Fname	Lname	Salary
Filter...	Filter...	Filter...
John	Smith	35000
Franklin	Wong	40000
Lucas	White	31000
Ramesh	Narayan	38000
James	Borg	55000
Jennifer	Wallace	43000

SOME:

The SOME operator is essentially synonymous with ANY and can be used interchangeably.

EXAMPLE:

Find employees whose salary is greater than the salary of some employees in department number 5.

```
SELECT Fname, Lname, Salary
FROM EMPLOYEE
WHERE Salary > SOME (
    SELECT Salary
    FROM EMPLOYEE
    WHERE Dno = 5
);
```

Fname	Lname	Salary
Filter...	Filter...	Filter...
John	Smith	35000
Franklin	Wong	40000
Lucas	White	31000
Ramesh	Narayan	38000
James	Borg	55000
Jennifer	Wallace	43000

ALL:

The ALL operator is used to compare a value to all values in another result set.

EXAMPLE:

Find employees whose salary is greater than all employees' salaries in department number 5.

```
SELECT Fname, Lname, Salary
FROM EMPLOYEE
WHERE Salary > ALL (
   SELECT Salary
   FROM EMPLOYEE
   WHERE Dno = 5
);
```

Fname	Lname	Salary
Filter...	Filter...	Filter...
James	Borg	55000
Jennifer	Wallace	43000

EXISTS:

The EXISTS operator is used to test for the existence of rows returned by a subquery. It returns true if the subquery returns one or more rows.

EXAMPLE:

Find employees who have at least one dependent.

```
SELECT Fname, Lname
FROM EMPLOYEE E
WHERE EXISTS (
   SELECT 1
   FROM DEPENDENT D
   WHERE E.ssn = D.Essn
);
```

Fname	Lname
Filter...	Filter...
John	Smith
Franklin	Wong
Jennifer	Wallace

NOT EXISTS:

The NOT EXISTS operator is used to test for the non-existence of rows returned by a subquery. It returns true if the subquery returns no rows.

EXAMPLE:

Find employees who do not have any dependents.

```
SELECT Fname, Lname
FROM EMPLOYEE E
WHERE NOT EXISTS (
   SELECT 1
   FROM DEPENDENT D
   WHERE E.ssn = D.Essn
);
```

Fname	Lname
abc Filter...	abc Filter...
Joyce	English
Lucas	White
Ramesh	Narayan
James	Borg
Ahmad	Jabbar
Alicia	Zelaya

EXPERIMENT-9

Aim:

Implementation of

i. Creating INDEXES and VIEWS
ii. INSERTING, DELETING and DROPING on VIEWs

INDEX:

Indexes improve the speed of data retrieval operations on a database table. They are particularly useful for speeding up queries involving WHERE clauses, joins, or sorting.

SYNTAX:

CREATE INDEX index_name

ON table_name (column_name);

EXAMPLE:

1. **Creating a Single-Column Index:**
   -- Create an index on the 'ssn' column of the EMPLOYEE table
   CREATE INDEX idx_ssn
   ON EMPLOYEE (ssn);
   SHOW INDEX FROM EMPLOYEE;

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type
Filte	Filter...	Filter...	Filter...	Filter...	Filter...	Filter...	Filter...	Filter...	Filte	Filter...
employee	0	PRIMARY	1	ssn	A	10	NULL	NULL		BTREE
employee	1	idx_ssn	1	ssn	A	9	NULL	NULL		BTREE

2. **Creating a Composite Index:**
   -- Create an index on both 'Dno' and 'Salary' columns of the EMPLOYEE table
   CREATE INDEX idx_dno_salary
   ON EMPLOYEE (Dno, Salary);
   SHOW INDEX FROM EMPLOYEE;

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type
Filte	Filter...	Filter...	Filter...	Filter...	Filter...	Filter...	Filter...	Filter...	Filte	Filter...
employee	0	PRIMARY	1	ssn	A	10	NULL	NULL		BTREE
employee	1	idx_ssn	1	ssn	A	9	NULL	NULL		BTREE
employee	1	idx_dno_salary	1	Dno	A	3	NULL	NULL	YES	BTREE
employee	1	idx_dno_salary	2	Salary	A	8	NULL	NULL	YES	BTREE

VIEWS:

A view is a virtual table based on the result of an SQL query. It can simplify complex queries, provide a way to present data to users, and restrict access to certain data.

SYNTAX:

```
CREATE VIEW view_name AS

SELECT column1, column2, ...

FROM table_name

WHERE condition;
```

EXAMPLE:

1. **Creating a Simple View:**

```
-- Create a view to display employee names and their departments
CREATE VIEW EmployeeDept AS
SELECT E.Fname, E.Lname, D.Dname
FROM EMPLOYEE E
JOIN DEPARTMENT D ON E.Dno = D.Dno;
SELECT * FROM EmployeeDept;
```

Fname	Lname	Dname
Filter...	Filter...	Filter...
James	Borg	Headquarters
Ahmad	Jabbar	Administration
Alicia	Zelaya	Administration
Jennifer	Wallace	Administration
Joyce	English	Research
Lucas	White	Research
John	Smith	Research
Ramesh	Narayan	Research
Franklin	Wong	Research

2. **Creating a View with Aggregation:**

```
-- Create a view to display the average salary by department
CREATE VIEW DeptAvgSalary AS
SELECT D.Dname, AVG(E.Salary) AS Avg_Salary
FROM EMPLOYEE E
JOIN DEPARTMENT D ON E.Dno = D.Dno
GROUP BY D.Dname;
SELECT * FROM DeptAvgSalary;
```

Dname	Avg_Salary
Filter...	Filter...
Headquarters	55000.0000
Administration	31000.0000
Research	33800.0000

Inserting values into views:

You can insert data into a view if the view is updatable (i.e., it is based on a single table and does not involve complex operations like joins).

SYNTAX:

```
INSERT INTO view_name (column1, column2, ...)
VALUES (value1, value2, ...);
```

EXAMPLE:

```
-- Create a view based on a single base table
CREATE VIEW EmployeeView AS
SELECT ssn, Fname, Salary
FROM EMPLOYEE;
SELECT * FROM EmployeeView;
```

ssn	Fname	Salary
Filter...	Filter...	Filter...
123456789	John	35000
333445555	Franklin	40000
453453453	Joyce	25000
666777888	Lucas	31000
666884444	Ramesh	38000
888665555	James	55000
987654321	Jennifer	43000
987987987	Ahmad	25000
999887777	Alicia	25000

```
-- Insert into the view
INSERT INTO EmployeeView (ssn, Fname, Salary)
VALUES ('888888888', 'Rohith', 60000);
SELECT * FROM EmployeeView;
```

ssn	Fname	Salary
Filter...	Filter...	Filter...
123456789	John	35000
333445555	Franklin	40000
453453453	Joyce	25000
666777888	Lucas	31000
666884444	Ramesh	38000
888665555	James	55000
888888888	Rohith	60000
987654321	Jennifer	43000
987987987	Ahmad	25000
999887777	Alicia	25000

Deleting values from views:

You can delete rows from a view if it is updatable. The deletion will affect the underlying base table.

SYNTAX:

```
DELETE FROM view_name
WHERE condition;
```

EXAMPLE:

```
-- Delete from the view assuming it's updatable and based on a single table
DELETE FROM EmployeeView
WHERE Fname = 'Rohith';
SELECT * FROM EmployeeDept;
```

Fname	Lname	Dname
Filter...	Filter...	Filter...
James	Borg	Headquarters
Ahmad	Jabbar	Administration
Alicia	Zelaya	Administration
Jennifer	Wallace	Administration
Joyce	English	Research
Lucas	White	Research
John	Smith	Research
Ramesh	Narayan	Research
Franklin	Wong	Research

Dropping a view:

Dropping a view removes it from the database. It does not affect the underlying tables.

SYNTAX:

```
DROP VIEW view_name;
```

EXAMPLE:

```
-- Drop the view 'EmployeeDept'
DROP VIEW EmployeeView;
SELECT * FROM EmployeeView;
```

Query with errors. Please, check the error below.

```
Table '22501a0557.employeeview' doesn't exist
```

CONSOLE RE-RUN QUERY EXPORT OPEN

EXPERIMENT-10

Aim:

Examples on

i. Creating and Calling STORED PROCEDURE (IN, OUT, INOUT Parameters), Dropping a STORED PROCEDURE.
ii. Creating, calling and Dropping a FUNCTION.
iii. Creating and Dropping a TRIGGER

STORED PROCEDURE:

A stored procedure is a set of SQL statements that can be executed as a single unit. Stored procedures can have parameters of type IN, OUT, or INOUT.

We can also change the delimiter in MySQL

To change the delimiter : DELIMITER $$;

To resume ";" as a delimiter : DELIMITER ; $$

SYNTAX:

```
CREATE PROCEDURE procedure_name (IN param1 datatype, OUT param2 datatype,
INOUT param3 datatype)
BEGIN
   -- SQL statements
END;
```

EXAMPLE:

1. -- Creating a procedure with IN parameters (for inserting a new employee)

```
DELIMITER //
CREATE PROCEDURE InsertEmployee(IN ssn BIGINT, IN Fname VARCHAR(20), IN
Salary BIGINT)
BEGIN
   INSERT INTO EMPLOYEE (ssn, Fname, Salary)
   VALUES (ssn, Fname, Salary);
END //
DELIMITER ;
```

✔	4	08:25:46	CREATE PROCEDURE InsertEmployee(

-- Inserting a new employee

CALL InsertEmployee(000000000,'Klee', 35000);

SELECT * FROM EMPLOYEE;

Fname	Minit	Lname	ssn	Bdate	Address	Gender	Salary	super_ssn	Dno
Filter...	Filter..	Filter...	Filt	Filter.	Filter...	Filter...	Filter..	Filter...	Filte
Klee	NULL	NULL	0	NULL	NULL	NULL	35000	NULL	NULL
John	B	Smith	12345...	1965-01-...	731 Fondren, Houston TX	M	35000	333445555	5
Franklin	T	Wong	33344...	1965-12-...	638 Voss, Houston TX	M	40000	888665555	5
Joyce	A	English	45345...	1972-07-...	5631 Rice, Houston TX	F	25000	333445555	5
Lucas	W	White	66677...	1990-02-...	789 Pine, Houston TX	M	31000	333445555	5
Ramesh	K	Narayan	66688...	1962-09-...	975 Fire Oak, Humble TX	M	38000	333445555	5
James	E	Borg	88866...	1937-11-...	450 Stone, Houston TX	M	55000	NULL	1
Jennifer	S	Wallace	98765...	1941-06-...	291 Berry, Bellaire TX	F	43000	888665555	4
Ahmad	V	Jabbar	98798...	1969-03-...	980 Dallas, Houston TX	M	25000	987654321	4
Alicia	J	Zelaya	99988...	1968-01-...	3321 Castle, Spring TX	F	25000	987654321	4

2. -- Creating a procedure to UPDATE an employee's salary

```
DELIMITER //
CREATE PROCEDURE UpdateEmployeeSalary(IN empSSN BIGINT, IN newSalary
BIGINT)
BEGIN
   UPDATE EMPLOYEE
   SET Salary = newSalary
   WHERE ssn = empSSN;
END //
DELIMITER ;
```

✔	6	08:27:54	CREATE PROCEDURE UpdateEmployeeSalary(

-- Updating salary for a specific employee

CALL UpdateEmployeeSalary(000000000, 50000);

SELECT * FROM EMPLOYEE;

Fname	Minit	Lname	ssn	Bdate	Address	Gender	Salary	super_ssn	Dno
Filter..	Filter	Filter...	Filter..	Filter...	Filter...	Filter...	Filter..	Filter...	Filte
Klee	NULL	NULL	0	NULL	NULL	NULL	50000	NULL	NULL
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston TX	M	35000	333445555	5
Franklin	T	Wong	333445555	1965-12-08	638 Voss, Houston TX	M	40000	888665555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston TX	F	25000	333445555	5
Lucas	W	White	666777888	1990-02-23	789 Pine, Houston TX	M	31000	333445555	5
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble TX	M	38000	333445555	5
James	E	Borg	888665555	1937-11-10	450 Stone, Houston TX	M	55000	NULL	1
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire TX	F	43000	888665555	4
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston TX	M	25000	987654321	4
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring TX	F	25000	987654321	4

3. -- Creating a procedure to DELETE an employee by SSN

```
DELIMITER //
CREATE PROCEDURE DeleteEmployee(IN empSSN BIGINT)
BEGIN
        DELETE FROM EMPLOYEE WHERE ssn = empSSN;

END //

DELIMITER ;
```

7 08:30:35 CREATE PROCEDURE DeleteEmployee(

```
-- Deleting an employee

CALL DeleteEmployee(123456789);

SELECT * FROM EMPLOYEE;
```

Fname	Minit	Lname	ssn	Bdate	Address	Gender	Salary	super_ssn	Dno
Filter...	Filter	Filter...	Filter...	Filter...	Filter...	Filter...	Filter...	Filter...	Filt
Klee	NULL	NULL	0	NULL	NULL	NULL	50000	NULL	NULL
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston TX	M	35000	333445555	5
Franklin	T	Wong	333445555	1965-12-08	638 Voss, Houston TX	M	40000	888665555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston TX	F	25000	333445555	5
Lucas	W	White	666777888	1990-02-23	789 Pine, Houston TX	M	31000	333445555	5
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble TX	M	38000	333445555	5
James	E	Borg	888665555	1937-11-10	450 Stone, Houston TX	M	55000	NULL	1
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire TX	F	43000	888665555	4
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston TX	M	25000	987654321	4
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring TX	F	25000	987654321	4

Dropping a Stored Procedure:

```
DROP PROCEDURE IF EXISTS InsertEmployee;
DROP PROCEDURE IF EXISTS UpdateEmployeeSalary;
DROP PROCEDURE IF EXISTS DeleteEmployee;

CALL DeleteEmployee(000000000)
```

Query with errors. Please, check the error below.

```
PROCEDURE 22501a0557.DeleteEmployee does not exist
```

CONSOLE RE-RUN QUERY EXPORT OPEN

4. This procedure takes an employee's SSN as input and returns their salary using an OUT parameter.

```
DELIMITER //
CREATE PROCEDURE GetEmployeeSalary(IN empSSN BIGINT, OUT firstName VARCHAR(30), OUT lastName VARCHAR(30) , OUT empSalary BIGINT)
BEGIN
   SELECT Fname, Lname, Salary INTO firstName, lastName, empSalary
   FROM EMPLOYEE
   WHERE ssn = empSSN;
END //
DELIMITER ;
```

3 22:06:15 CREATE PROCEDURE GetEmployeeSalary(

```
CALL GetEmployeeSalary(123456789, @firstName, @lastName ,@salary);
SELECT @firstName, @lastName , @salary;
```

@firstName	@lastName	@salary
abc Filter...	abc Filter...	abc Filter...
John	Smith	35000

5. This procedure increases an employee's salary by a certain percentage and returns the updated salary using an INOUT parameter.

```
DELIMITER //
CREATE PROCEDURE IncreaseEmployeeSalary(INOUT empSSN BIGINT, IN percentage DOUBLE)
BEGIN
   DECLARE currentSalary BIGINT;

   -- Get current salary
   SELECT Salary INTO currentSalary
   FROM EMPLOYEE
   WHERE SSN = empSSN;

   -- Update salary by the given percentage
   UPDATE EMPLOYEE
   SET Salary = currentSalary + (currentSalary * percentage / 100)
   WHERE SSN = empSSN;

   -- Return the updated employee details (Fname, Lname, and Salary)
   SELECT Fname, Lname, Salary
   FROM EMPLOYEE
   WHERE SSN = empSSN;
END //
DELIMITER ;
```

45 22:22:05 CREATE PROCEDURE IncreaseEmployeeSalary(

```
-- Define a user variable to hold SSN
SET @empSSN = 123456789;
```

```
-- Increasing salary by 10% and getting the updated details
CALL IncreaseEmployeeSalary(@empSSN, 10);
```

Fname	Lname	Salary
abc Filter...	abc Filter...	abc Filter...
John	Smith	38500

6. This procedure checks if an employee's salary is below a specified threshold. If it is, the salary is increased to the threshold amount.

```
DELIMITER //
CREATE PROCEDURE CheckAndUpdateSalary(IN empSSN BIGINT, IN threshold
BIGINT)
BEGIN
   DECLARE currentSalary BIGINT;

   -- Get the current salary of the employee
   SELECT Salary INTO currentSalary
   FROM EMPLOYEE
   WHERE SSN = empSSN;

   -- IF condition to check if the salary is below the threshold
   IF currentSalary < threshold THEN
      UPDATE EMPLOYEE
      SET Salary = threshold
      WHERE SSN = empSSN;
   END IF;

   -- Return the employee's first name, last name, and updated salary
   SELECT Fname, Lname, Salary
   FROM EMPLOYEE
   WHERE SSN = empSSN;
END //
DELIMITER ;
```

57 22:28:22 CREATE PROCEDURE CheckAndUpdateSalary(

```
-- Checking and updating salary if it is below the threshold (40000)
CALL CheckAndUpdateSalary(123456789, 40000);
```

Fname	Lname	Salary
abc Filter...	abc Filter...	abc Filter...
John	Smith	40000

CURSOR:

Cursors in MySQL allow sequential traversal of records returned by a query. They are useful for performing row-by-row operations, especially when batch processing of result sets is required.

SYNTAX:

1. **Declare a Cursor**
   DECLARE cursor_name CURSOR FOR select_statement;
2. **Open a Cursor**
   OPEN cursor_name;
3. **Fetch Data into Variables**
   FETCH cursor_name INTO var1, var2, ...;
4. **Close the Cursor**
   CLOSE cursor_name;
5. **Declare a Handler for NOT FOUND**
   DECLARE CONTINUE HANDLER FOR NOT FOUND SET var_name = TRUE;

EXAMPLE:

We will create a stored procedure tt35 that processes employees with a salary less than 35,000 and increases their salary by 10%.

```
CREATE PROCEDURE tt35()

BEGIN

   DECLARE sal BIGINT;  -- Variable to store the salary

   DECLARE eid BIGINT;  -- Variable to store the employee SSN

   DECLARE salary_status VARCHAR(30) DEFAULT 'Updated';  -- Variable to store
status

   DECLARE r_f TINYINT DEFAULT FALSE;  -- Flag to check end of cursor


   -- Declare a cursor to fetch employees with salary less than 35,000

   DECLARE cur_sal CURSOR FOR

   SELECT SSN, Salary FROM EMPLOYEE WHERE Salary < 35000;


   -- Declare a handler to set the flag when no more rows are available

   DECLARE CONTINUE HANDLER FOR NOT FOUND SET r_f = TRUE;


   -- Open the cursor
```

```
    OPEN cur_sal;
    -- Loop through each row from the cursor
    WHILE r_f = FALSE DO
        -- Fetch the SSN and Salary of the employee
        FETCH cur_sal INTO eid, sal;

        -- If the employee exists, update their salary by 10%
        IF r_f = FALSE THEN
            UPDATE EMPLOYEE SET Salary = sal + (sal * 0.1) WHERE SSN = eid;
            SELECT eid AS Employee_SSN, sal AS Old_Salary, sal + (sal * 0.1) AS
New_Salary, salary_status AS Salary_Status;
        END IF;
    END WHILE;
    -- Close the cursor
    CLOSE cur_sal;
END //
DELIMITER ;
```

✓	63	22:42:23	CREATE PROCEDURE tt35()

```
-- Call the procedure to update the salary of employees with salary less than 35,000
CALL tt35();
```

Employee_SSN	Old_Salary	New_Salary	Salary_Status
abc Filter...	abc Filter...	abc Filter...	abc Filter...
111111111	21000	23100.0	Updated

```
-- Verify the update with a select query
SELECT * FROM EMPLOYEE WHERE Salary < 35000;
```

Fname	Minit	Lname	ssn	Bdate	Address	Gender	Salary	super_ssn	Dno
abc Filter.	abc Filter	abc Filter..	abc Filter...	abc Filter...	abc Filter...	abc Filter...	abc Filter..	abc Filter...	abc Filte
Test	T	User	111111111	1990-01-01	123 Test St, Test City	M	23100	NULL	1
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston TX	F	27500	333445555	5
Lucas	W	White	666777888	1990-02-23	789 Pine, Houston TX	M	34100	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston TX	M	27500	987654321	4
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring TX	F	27500	987654321	4

FUNCTIONS:

A function is a stored program that returns a single value. Functions can be used in SQL statements similarly to built-in functions.

SYNTAX:

```
CREATE FUNCTION function_name (param1 datatype, param2 datatype)
RETURNS datatype
BEGIN
   -- SQL statements
   RETURN value;
END;
```

EXAMPLE:

1. -- Create a function to calculate the yearly bonus (10% of salary)

```
DELIMITER //
CREATE FUNCTION fn_example(emp_id INT) RETURNS DECIMAL(10,2)
READS SQL DATA
BEGIN
   DECLARE emp_salary DECIMAL(10,2);
   SELECT salary INTO emp_salary FROM EMPLOYEE WHERE ssn = emp_id;
   RETURN emp_salary * 1.1;
END //
DELIMITER ;
```

15 09:22:20 CREATE FUNCTION fn_example(

```
SELECT fn_example(123456789) yearly_bonus57;
```

yearly_bonus57
abc Filter...
38500.00

```
DROP FUNCTION IF EXISTS fn_example;
```

Query with errors. Please, check the error below.

```
FUNCTION 22501a0557.fn_example does not exist
```

CONSOLE RE-RUN QUERY EXPORT OPEN

2. This function calculates the total salaries of employees belonging to department 5.

```
DELIMITER //

CREATE FUNCTION get_sal(deptno INT) RETURNS DECIMAL(9,2)
DETERMINISTIC
READS SQL DATA
BEGIN
   DECLARE sum_sal DECIMAL(9,2);
   SELECT SUM(Salary) INTO sum_sal FROM EMPLOYEE WHERE dno = deptno;
   RETURN sum_sal;
END //

DELIMITER ;
```

✓	65	22:52:52	CREATE FUNCTION get_sal

```
-- Example: Get total salary for department 5
SELECT get_sal(5) AS dept_sal;
```

dept_sal
abc Filter...
179600.00

3. This function counts the number of projects that a specified employee is working on based on the WORKS_ON table.

```
DELIMITER //

CREATE FUNCTION get_proj(eno BIGINT) RETURNS INT
DETERMINISTIC
READS SQL DATA
BEGIN
   DECLARE ecount INT;
   SELECT COUNT(pno) INTO ecount FROM WORKS_ON WHERE Essn = eno;
   RETURN ecount;
END //
DELIMITER ;
```

✓	68	22:57:38	CREATE FUNCTION get_proj

```
-- Example: Get project count for employee with SSN 123456789
SELECT get_proj(123456789) AS projects;
```

projects
abc Filter...
2

TRIGGER:

A trigger is a set of SQL statements that automatically execute in response to certain events on a particular table, such as INSERT, UPDATE, or DELETE.

SYNTAX:

- **Creating a Trigger:**

```
CREATE TRIGGER trigger_name
BEFORE/AFTER INSERT/UPDATE/DELETE
ON table_name
FOR EACH ROW
BEGIN
   -- SQL statements
END;
```

- **Dropping a Trigger:**

```
DROP TRIGGER IF EXISTS trigger_name;
```

EXAMPLE:

1. **Trigger when Inserting:**

```
DELIMITER //
CREATE TRIGGER before_insert_employee
BEFORE INSERT ON EMPLOYEE
FOR EACH ROW
BEGIN
   SET NEW.salary = NEW.salary + 1000;
END //
DELIMITER ;
INSERT INTO EMPLOYEE
VALUES ('Test','T','User',111111111,'1990-01-01','123 Test St, Test
City','M',20000,NULL,1);
```

16	09:39:10	CREATE TRIGGER before_insert_employee

```
SELECT * FROM EMPLOYEE WHERE ssn = 111111111;
```

Fname	Minit	Lname	ssn	Bdate	Address	Gender	Salary	super_ssn	Dno
Filter...	Filter	Filter...	Filter	Filter..	Filter...	Filter...	Filter.	Filter...	Filt
Test	T	User	111111111	1990-01-01	123 Test St, Test City	M	21000	NULL	1

2. **Trigger when Updating:**

```
DELIMITER //
CREATE TRIGGER ensure_non_negative_salary
BEFORE UPDATE ON EMPLOYEE
FOR EACH ROW
BEGIN
   IF NEW.salary < 0 THEN
      SET NEW.salary = OLD.salary;
   END IF;
END //
DELIMITER ;
```

20 09:43:54 CREATE TRIGGER ensure_non_negative_salary

```
-- Attempt to update the salary to a negative value
UPDATE EMPLOYEE SET salary = -5000 WHERE ssn = 123456789;
-- Check the salary to see if it was reset
SELECT Fname, Lname, salary FROM EMPLOYEE WHERE ssn = 123456789;
```

Fname	Lname	salary
abc Filter...	abc Filter...	abc Filter...
John	Smith	35000

3. **Dropping a Trigger:**

```
DROP TRIGGER IF EXISTS before_insert_employee;
DROP TRIGGER IF EXISTS ensure_non_negative_salary;
```

23 09:49:46 DROP TRIGGER IF EXISTS ensure_non_negative_salary

EXPERIMENT-11

Case Study - 1

Hospital Management System

Whenever a new patient is either admitted or comes for outdoor check-up, a uniquepatient id is generated after storing the name, address and date of birth of thepatient. For furthervisits, the patient uses his unique id. There are severaldepartments in the Hospital. Adepartment characterized by a unique id, name,floor number and total workers. Thedoctors have a unique employee id, and theirname, address, contact number, qualifications are stored with this id.Similarly other workers like nurses, ward boys, ambulance drivers also have aunique employee id. Also, each worker is characterized by name, address and type.Doctors and workers can be associated with multiple departments with differentschedules. Whenever patient is admitted in the hospital various details arerecorded. Thepatient id, name and address are stored, the department number andname in which thepatient is admitted along with the bed number and room numberis also stored. Also, forevery patient, a senior doctor and junior doctor areappointed. The details of the doctors, like name, id, contact number is recorded.The prescribed medicines are also stored for apatient. In case of outdoor checkups,the patient id is stored, along with the departmentnumber, the employee id of thedoctor, the prescription and the date of checkup. Foremergency duty every night,the employee_id of doctor and the nurse is stored along withdate.

1. Data Analysis:

The data required for the Hospital Management System can be categorized into the following entities:

1. **Patient**: Each patient has a unique ID, name, address, and date of birth. For admitted patients, additional data includes department number, bed number, room number, and assigned doctors.
2. **Department**: Each department has a unique ID, name, floor number, and total workers.
3. **Doctor**: Each doctor has a unique employee ID, name, address, contact number, qualifications, and is associated with one or more departments.

4. **Worker**: Workers (nurses, ward boys, ambulance drivers) also have a unique employee ID, name, address, and type. They can be associated with multiple departments.
5. **Admissions**: When a patient is admitted, details such as patient ID, department, room, and assigned doctors are recorded.
6. **Outdoor Checkup**: For outdoor checkups, patient ID, department, doctor ID, prescription, and date of checkup are stored.
7. **Emergency Duty**: Employee IDs of doctors and nurses are recorded along with the date for night duties.
8. **Prescriptions**: Each patient's prescription, which includes medications and associated information, is stored.

2. Conceptual Schema (ER Diagram):

3. Relational Schema and Normalization:

1. **Patient**:

Patient_id	Name	Address	Date of Birth

2. **Department:**

Department_id	Name	Floor	Total_Workers

3. **Doctor:**

Employee_id	Name	Address	Contact	Qualification

4. **Worker:**

Employee_id	Name	Address	Type

5. **Doc_Department:**

Employee_id	Department_id	Schedule

6. **Admission:**

Admission_id	Patient_id	Department_id	Bed_no	Room_no	Sr_doc	Jr_doc

7. **Outdoor_checkup:**

Checkup_id	Patient_id	Department	Doc_id	Prescription	Checkup_date

8. **Emergency_Duty:**

Duty_id	Doc_id	Nurse_id	Date

9. **Prescription:**

Prescription_id	Patient_id	Medication	Dosage	Frequency

3.2 Normalization Process:

The relational schema is normalized to the 3rd Normal Form (3NF).

1NF (First Normal Form): All tables have "**atomic"** columns with unique rows.

2NF (Second Normal Form): Each non-key attribute is "**fully functionally dependent"** on the primary key.

For example:doctor-department relationships are stored in a separate table, Doc_Department, to prevent partial dependency.

3NF (Third Normal Form): Transitive dependencies are removed. Each non-key attribute "**depends solely"** on the primary key of its table.

4. Implementing in MySQL:

Creating tables:

1. **Patient:**

   Query:CREATE TABLE Patient (Patient_ID INT PRIMARY KEY, Name VARCHAR(100), Address VARCHAR(255), DOB DATE);

   Output:

Result Grid | Filter Rows: | Export: | Wrap

Field	Type	Null	Key	Default	Extra
Patient_ID	int	NO	PRI	NULL	
Name	varchar(100)	YES		NULL	
Address	varchar(255)	YES		NULL	
DOB	date	YES		NULL	

2. **Department:**

   Query:CREATE TABLE Department (Department_ID INT PRIMARY KEY, Name VARCHAR(100), Floor INT, Total_Workers INT);

   Output:

Result Grid | Filter Rows: | Export: | Wrap Cell Con

Field	Type	Null	Key	Default	Extra
Department_ID	int	NO	PRI	NULL	
Name	varchar(100)	YES		NULL	
Floor	int	YES		NULL	
Total_Workers	int	YES		NULL	

3. **Doctor:**

   Query:CREATE TABLE Doctor (Employee_ID INT PRIMARY KEY, Name VARCHAR(100), Address VARCHAR(255), Contact VARCHAR(15), Qualification VARCHAR(100));

   Output:

Result Grid | Filter Rows: | Export: | Wrap

Field	Type	Null	Key	Default	Extra
Employee_ID	int	NO	PRI	NULL	
Name	varchar(100)	YES		NULL	
Address	varchar(255)	YES		NULL	
Contact	varchar(15)	YES		NULL	
Qualification	varchar(100)	YES		NULL	

4. **Worker:**

   Query: CREATE TABLE Worker (Employee_ID INT PRIMARY KEY, Name VARCHAR(100), Address VARCHAR(255), Type VARCHAR(50));

   Output:

Result Grid | Filter Rows: | Export: | Wrap C

Field	Type	Null	Key	Default	Extra
Employee_ID	int	NO	PRI	NULL	
Name	varchar(100)	YES		NULL	
Address	varchar(255)	YES		NULL	
Type	varchar(50)	YES		NULL	

5. **Doc_Department:**

   Query:CREATE TABLE Doc_Department(Employee_ID INT, Department_ID INT, Schedule VARCHAR(255), FOREIGN KEY (Employee_ID) REFERENCES Doctor(Employee_ID), FOREIGN KEY (Department_ID) REFERENCES Department(Department_ID), PRIMARY KEY (Employee_ID, Department_ID));

   Output:

Result Grid | Filter Rows: | Export: | Wrap Cell Conte

Field	Type	Null	Key	Default	Extra
Employee_ID	int	NO	PRI	NULL	
Department_ID	int	NO	PRI	NULL	
Schedule	varchar(255)	YES		NULL	

6. **Admission:**

   Query: CREATE TABLE Admission (Admission_ID INT PRIMARY KEY,Patient_ID INT, Department_ID INT,Bed_Number INT,Room_Number INT,Senior_Doctor INT, Junior_Doctor INT,FOREIGN KEY (Patient_ID) REFERENCES Patient(Patient_ID), FOREIGN KEY (Department_ID) REFERENCES Department(Department_ID),

FOREIGN KEY (Senior_Doctor) REFERENCES Doctor(Employee_ID),FOREIGN KEY (Junior_Doctor) REFERENCES Doctor(Employee_ID));

Output:

Result Grid | Filter Rows: | Export:

Field	Type	Null	Key	Default	Ext
Admission_ID	int	NO	PRI	NULL	
Patient_ID	int	YES	MUL	NULL	
Department_ID	int	YES	MUL	NULL	
Bed_Number	int	YES		NULL	
Room_Number	int	YES		NULL	
Senior_Doctor	int	YES	MUL	NULL	
Junior_Doctor	int	YES	MUL	NULL	

7. **Outdoor_checkup:**

Query: CREATE TABLE Outdoor_Checkup(Checkup_ID INT PRIMARY KEY, Patient_ID INT, Department_ID INT, Doctor_ID INT, Prescription TEXT, Checkup_Date DATE, FOREIGN KEY (Patient_ID) REFERENCES Patient(Patient_ID), FOREIGN KEY (Department_ID) REFERENCES Department(Department_ID), FOREIGN KEY (Doctor_ID) REFERENCES Doctor(Employee_ID));

Output:

Result Grid | Filter Rows: | Export:

Field	Type	Null	Key	Default	Extra
Checkup_ID	int	NO	PRI	NULL	
Patient_ID	int	YES	MUL	NULL	
Department_ID	int	YES	MUL	NULL	
Doc_ID	int	YES	MUL	NULL	
Prescription	text	YES		NULL	
Checkup_Date	date	YES		NULL	

8. **Emergency_Duty:**

Query: CREATE TABLE Emergency_Duty(Duty_ID INT PRIMARY KEY, Doctor_ID INT, Nurse_ID INT, Date DATE, FOREIGN KEY (Doctor_ID) REFERENCESDoctor(Employee_ID), FOREIGN KEY (Nurse_ID) REFERENCES Worker(Employee_ID));

Output:

Field	Type	Null	Key	Default	Extra
Duty_ID	int	NO	PRI	NULL	
Doctor_ID	int	YES	MUL	NULL	
Nurse_ID	int	YES	MUL	NULL	
Date	date	YES		NULL	

9. **Prescription:**

Query:CREATE TABLE Prescription (Prescription_ID INT PRIMARY KEY, Patient_ID INT, Medication VARCHAR(255), Dosage VARCHAR(50), Frequency VARCHAR(50), FOREIGN KEY (Patient_ID) REFERENCES Patient(Patient_ID));

Output:

Result Grid | Filter Rows: | Export: | Wrap Cell Con

Field	Type	Null	Key	Default	Extra
Prescription_ID	int	NO	PRI	NULL	
Patient_ID	int	YES	MUL	NULL	
Medication	varchar(255)	YES		NULL	
Dosage	varchar(50)	YES		NULL	
Frequency	varchar(50)	YES		NULL	

Inserting data into tables:

1. **Patient**

Query: INSERT INTO Patient (Patient_ID, Name, Address, DOB) VALUES (1, 'John Doe', '123 Main St, City A', '1985-06-12'),

(2, 'Jane Smith', '456 Oak St, City B', '1990-03-25');

Output:

Result Grid | Filter Rows: | Edit:

Patient_ID	Name	Address	DOB
1	John Doe	123 Main St, City A	1985-06-12
2	Jane Smith	456 Oak St, City B	1990-03-25
NULL	NULL	NULL	NULL

2. **Department**

Query: INSERT INTO Department (Department_ID, Name, Floor, Total_Workers)VALUES (101, 'Cardiology', 2, 20),(102, 'Neurology', 3, 15);

Output:

Result Grid | Filter Rows: | Edit

Department_ID	Name	Floor	Total_Workers
101	Cardiology	2	20
102	Neurology	3	15
NULL	NULL	NULL	NULL

3. **Doctor**

Query: INSERT INTO Doctor (Employee_ID, Name, Address, Contact, Qualification)

VALUES (201, 'Dr. Alan Turner', '789 Pine St, City A', '1234567890', 'MD Cardiology'),

(202, 'Dr. Emily Clark', '321 Cedar St, City B', '0987654321', 'MD Neurology');

Output:

Result Grid | Filter Rows: | Edit: | Export/Import:

Employee_ID	Name	Address	Contact	Qualification
201	Dr. Alan Turner	789 Pine St, City A	1234567890	MD Cardiology
202	Dr. Emily Clark	321 Cedar St, City B	0987654321	MD Neurology
NULL	NULL	NULL	NULL	NULL

4. **Worker**

Query: INSERT INTO Worker (Employee_ID, Name, Address, Type)

VALUES (301, 'Nurse Amy Green', '567 Birch St, City C', 'Nurse'),

(302, 'Ward Boy Mike Blue', '890 Maple St, City D', 'Ward Boy');

Output:

Result Grid | Filter Rows: | Edit:

Employee_ID	Name	Address	Type
301	Nurse Amy Green	567 Birch St, City C	Nurse
302	Ward Boy Mike Blue	890 Maple St, City D	Ward Boy
NULL	NULL	NULL	NULL

5. **Doc_Department:**

Query: INSERT INTO Doc_Department (Employee_ID, Department_ID, Schedule)

VALUES (201, 101, 'Mon-Fri, 9am-5pm'),(202, 102, 'Mon-Wed, 10am-6pm');

Output:

Result Grid | Filter Rows: | Edit:

Employee_ID	Department_ID	Schedule
201	101	Mon-Fri, 9am-5pm
202	102	Mon-Wed, 10am-6pm
NULL	NULL	NULL

6. **Admission**

Query: INSERT INTO Admission (Admission_ID, Patient_ID, Department_ID, Bed_Number, Room_Number, Senior_Doctor, Junior_Doctor)

VALUES (1, 1, 101, 12, 101, 201, 202),(2, 2, 102, 8, 102, 202, 201);

Output:

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Conten

Admission_ID	Patient_ID	Department_ID	Bed_Number	Room_Number	Senior_Doctor	Junior_Doctor
1	1	101	12	101	201	202
2	2	102	8	102	202	201
NULL	NULL	NULL	NULL	NULL	NULL	NULL

7. **Outdoor_Checkup**

Query: INSERT INTO Outdoor_Checkup (Checkup_ID, Patient_ID, Department_ID, Doc_ID, Prescription, Checkup_Date)VALUES (1, 1, 101, 201, 'Medication A', '2024-09-10'),(2, 2, 102, 202, 'Medication B', '2024-09-11');

Output:

Result Grid | Filter Rows: | Edit: | Export/Import:

Checkup_ID	Patient_ID	Department_ID	Doc_ID	Prescription	Checkup_Date
1	1	101	201	Medication A	2024-09-10
2	2	102	202	Medication B	2024-09-11
NULL	NULL	NULL	NULL	NULL	NULL

8. **Emergency_Duty:**

Query:INSERT INTO Emergency_Duty (Duty_ID, Doctor_ID, Nurse_ID, Date) VALUES (1, 201, 301, '2024-09-12'),(2, 202, 301, '2024-09-13');

Output:

Result Grid | Filter Rows: | Edit:

Duty_ID	Doctor_ID	Nurse_ID	Date
1	201	301	2024-09-12
2	202	301	2024-09-13
NULL	NULL	NULL	NULL

9. **Prescription:**

Query:INSERT INTO Prescription (Prescription_ID, Patient_ID, Medication, Dosage, Frequency)VALUES (1, 1, 'Medication A', '500mg', 'Twice a Day'),

(2, 2, 'Medication B', '250mg', 'Once a Day');

Output:

Result Grid | Filter Rows: | Edit:

Prescription_ID	Patient_ID	Medication	Dosage	Frequency
1	1	Medication A	500mg	Twice a Day
2	2	Medication B	250mg	Once a Day
NULL	NULL	NULL	NULL	NULL

5. Query the Database:

1. Retrive all Patients

Query: select * from Patients;

Output:

Result Grid | Filter Rows: | Edit:

Patient_ID	Name	Address	DOB
1	John Doe	123 Main St, City A	1985-06-12
2	Jane Smith	456 Oak St, City B	1990-03-25
NULL	NULL	NULL	NULL

2. List all Doctors working in the Cardiology Department (Department_ID = 101)

Query: SELECT D.Name, D.QualificationFROM Doctor DJOIN Doc_Department DD ON D.Employee_ID = DD.Employee_IDWHERE DD.Department_ID = 101;

Output:

Result Grid | Filter Rows:

Name	Qualification
Dr. Alan Turner	MD Cardiology

3. Find all Admissions for a specific Patient (Patient_ID = 1)

Query: SELECT * FROM AdmissionWHERE Patient_ID = 1;

Output:

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Con

Admission_ID	Patient_ID	Department_ID	Bed_Number	Room_Number	Senior_Doctor	Junior_Doctor
1	1	101	12	101	201	202
NULL	NULL	NULL	NULL	NULL	NULL	NULL

4. Retrieve all Checkups for a specific Patient (Patient_ID = 1)

Query: SELECT * FROM Outdoor_CheckupWHERE Patient_ID = 1;

Output:

Result Grid | Filter Rows: | Edit: | Export/Import:

Checkup_ID	Patient_ID	Department_ID	Doc_ID	Prescription	Checkup_Date
1	1	101	201	Medication A	2024-09-10
NULL	NULL	NULL	NULL	NULL	NULL

5. List all Emergency Duties of a specific Doctor (Doctor_ID = 201)

Query: SELECT * FROM Emergency_DutyWHERE Doctor_ID = 201;

Output:

Result Grid | Filter Rows:

Duty_ID	Doctor_ID	Nurse_ID	Date
1	201	301	2024-09-12
NULL	NULL	NULL	NULL

6. Find doctors who are on emergency duty on a specific date

Query: SELECT D.Name, D.Employee_IDFROM Doctor DJOIN Emergency_Duty ED ON D.Employee_ID = ED.Doctor_IDWHERE ED.Date = '2024-09-12';

Output:

Result Grid | Filter Rows:

Name	Employee_ID
Dr. Alan Turner	201

7. Retrieve the names of all patients admitted to a specific department along with their senior and junior doctors' names

Query: SELECT P.Name AS Patient_Name, D1.Name AS Senior_Doctor, D2.Name AS Junior_DoctorFROM Admission AJOIN Patient P ON A.Patient_ID = P.Patient_ID JOIN Doctor D1 ON A.Senior_Doctor = D1.Employee_IDJOIN Doctor D2 ON A.Junior_Doctor = D2.Employee_IDWHERE A.Department_ID = 101;

Output:

Result Grid | Filter Rows: | Exp

Patient_Name	Senior_Doctor	Junior_Doctor
John Doe	Dr. Alan Turner	Dr. Emily Clark

8. Find the total number of checkups and admissions for each patient

Query: SELECT P.Patient_ID, P.Name, (SELECT COUNT(*) FROM Admission A WHERE A.Patient_ID = P.Patient_ID) AS Total_Admissions,(SELECT COUNT(*) FROM Outdoor_Checkup OC WHERE OC.Patient_ID = P.Patient_ID) AS Total_CheckupsFROM Patient P;

Output:

Result Grid | Filter Rows: | Export: | Wrap

Patient_ID	Name	Total_Admissions	Total_Checkups
1	John Doe	1	1
2	Jane Smith	1	1

Case Study 2

Airline Reservation System

There are 6 different airlines in 6 different countries: Canada – AirCan, USA - USAir, UK - BritAir, France - AirFrance, Germany - LuftAir, Italy - ItalAir.

Their flights involve the following 12 cities: Toronto and Montreal in Canada, New York and Chicago in US, London and Edinburgh in UK, Paris and Nice in France, Bonn and Berlin in Germany, Rome and Naples in Italy.

In each of the 12 cities, there is a (single) booking office.

You are going to design a central air-reservation database to be used by all booking offices.

The flight has a unique flight number, airline code, business class indicator, smoking allowed indicator.

Flight availability has flight number, date + time of departure, number of total seats available in business class, number of booked seats in business class, number of total seats available in economy class, and number of booked seats in economy class.

The customers may come from any country, not just the 6 above, and from any province/state, and from any city. Customer has first & last name, mailing address, zero or more phone numbers, zero or more fax numbers, and zero or more email addresses.

Mailing address has street, city, province or state, postal code and country. Phone/fax number has country code, area code and local number.

Email address has only one string, and no structure is assumed. A customer can book one or more flights.

Two or more customers may have same mailing address and/or same phone number(s) and/or same fax number(s). But the email address is unique for each customer. First and last names do not have to be unique.

Booking has an unique booking number, booking city, booking date, flight number, date + time of departure (in local time, and time is always in hours and minutes), date + time of arrival (in local time), class indicator, total price (airport tax in origin + airport tax in

destination + flight price – in local currency. The flight price for business class is 1.5 times of the listed flight price), status indicator (three types: booked. Canceled – the customer canceled the booking, scratched – the customer had not paid in full 30 days prior to the departure), customer who is responsible for payment, amount-paid-so far (in local currency), outstanding balance (in local currency), the first & last names to be printed on the ticket.

The airport taxes must be stored in local currencies (i.e. Canadian Master Thesis – Weiguang Zhang McMaster University- Computing and Software 26 dollars, US dollars, British Pounds, French francs, German marks, and Italian Liras).

Since the exchange rates change daily, they also must be stored for calculations of all prices involved. Though France, Germany, and Italy have had a common currency for a while, we used the names of their original currencies to involve in this exercise currency exchange rates and their changes.

1. Data Analysis:

1. Airline ↔ Flight (Operates)

- Airline: Total participation (Every airline must operate at least one flight).
- Flight: Partial participation (A flight must be operated by one airline, but not all flights are necessarily operated by every airline).

2. City ↔ BookingOffice (LocatedIn)

- City: Total participation (Every city must have one booking office).
- BookingOffice: Total participation (Every booking office must be located in one city).

3. Flight ↔ FlightAvailability (Scheduled)

- Flight: Total participation (Every flight must have at least one availability record).
- FlightAvailability: Total participation (Every flight availability record must be associated with one flight).

4. Customer ↔ MailingAddress (ResidesAt)

- Customer: Partial participation (Not every customer must have a mailing address).
- MailingAddress: Partial participation (Not every mailing address must be associated with a customer).

5. Customer ↔ PhoneFaxNumber (HasContact)

- Customer: Partial participation (Not every customer must have a phone/fax number).
- PhoneFaxNumber: Total participation (Every phone/fax number must be associated with one customer).

6. Flight ↔ Booking (BookedBy)

- Flight: Partial participation (Not every flight must be booked).
- Booking: Total participation (Every booking must be associated with one flight).

7. Customer ↔ Booking (Makes)

- Customer: Partial participation (Not every customer must make a booking).
- Booking: Total participation (Every booking must be made by one customer).

8. City ↔ AirportTax (Imposes)

- City: Total participation (Every city must impose an airport tax).
- AirportTax: Total participation (Every airport tax must be associated with one city).

9. Country ↔ Airline (Belongs)

- Country: Partial participation (Not every country must have an airline).
- Airline: Total participation (Every airline must belong to one country).

10. Country ↔ City (LocatedIn)

- Country: Total participation (Every country must have at least one city).
- City: Total participation (Every city must be located in one country).

11. Country ↔ ExchangeRate (Gives)

- Country: Partial participation (Not every country must have an exchange rate with another country).
- ExchangeRate: Total participation (Every exchange rate must be associated with one country)

2. Conceptual Schema (ER Diagram):

3. Relational Schema and Normalization:

Output:

4. Implementing in MySQL:

CREATION OF THE TABLES

1. **City Table**

Query:

```
CREATE TABLE City (
   CityName VARCHAR(50) PRIMARY KEY,
   Country VARCHAR(50)
);
desc city;
```

Output:

Result Grid | Filter Rows: | Export:

Field	Type	Null	Key	Default	Extra
CityName	varchar(50)	NO	PRI	NULL	
Country	varchar(50)	YES		NULL	

2. **Mailing Address Table**

Query:

```
CREATE TABLE MailingAddress (

   MailingAddressID INT PRIMARY KEY AUTO_INCREMENT,

   Street VARCHAR(100),

   CityName VARCHAR(50),

   ProvinceState VARCHAR(50),

   PostalCode VARCHAR(20),

   Country VARCHAR(50),

   FOREIGN KEY (CityName) REFERENCES City(CityName)

);
```

Output:

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Field	Type	Null	Key	Default	Extra
MailingAddressID	int	NO	PRI	NULL	auto_increment
Street	varchar(100)	YES		NULL	
CityName	varchar(50)	YES	MUL	NULL	
ProvinceState	varchar(50)	YES		NULL	
PostalCode	varchar(20)	YES		NULL	
Country	varchar(50)	YES		NULL	

3. **Customer Table**

Query :

```
CREATE TABLE Customer (
    CustomerID INT PRIMARY KEY AUTO_INCREMENT,
    FirstName VARCHAR(50),
    LastName VARCHAR(50),
    MailingAddressID INT,
    Email VARCHAR(100) UNIQUE,
    FOREIGN KEY (MailingAddressID) REFERENCES MailingAddress(MailingAddressID)
);

desc Customer;
```

Output:

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Field	Type	Null	Key	Default	Extra
CustomerID	int	NO	PRI	NULL	auto_increment
FirstName	varchar(50)	YES		NULL	
LastName	varchar(50)	YES		NULL	
MailingAddressID	int	YES	MUL	NULL	
Email	varchar(100)	YES	UNI	NULL	

4. **PhoneFaxNumber Table**

Query :

```
CREATE TABLE PhoneFaxNumber (
    PhoneFaxID INT PRIMARY KEY AUTO_INCREMENT,
    CustomerID INT,
    CountryCode VARCHAR(5),
    AreaCode VARCHAR(5),
    LocalNumber VARCHAR(15),
    FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID)
);
```

Output:

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Field	Type	Null	Key	Default	Extra
PhoneFaxID	int	NO	PRI	NULL	auto_increment
CustomerID	int	YES	MUL	NULL	
CountryCode	varchar(5)	YES		NULL	
AreaCode	varchar(5)	YES		NULL	
LocalNumber	varchar(15)	YES		NULL	

5. **Airline Table**

Query :

```
CREATE TABLE Airline (
    AirlineCode VARCHAR(10) PRIMARY KEY,
    AirlineName VARCHAR(50),
    Country VARCHAR(50)
);
desc airline;
```

Output:

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Field	Type	Null	Key	Default	Extra
AirlineCode	varchar(10)	NO	PRI	NULL	
AirlineName	varchar(50)	YES		NULL	
Country	varchar(50)	YES		NULL	

6. **Flight Table**

Query :

```
CREATE TABLE Flight (
    FlightNumber VARCHAR(10) PRIMARY KEY,
    AirlineCode VARCHAR(10),
    BusinessClassIndicator BOOLEAN,
    SmokingAllowedIndicator BOOLEAN,
    FOREIGN KEY (AirlineCode) REFERENCES Airline(AirlineCode)
);
desc flight;
```

Output:

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Field	Type	Null	Key	Default	Extra
FlightNumber	varchar(10)	NO	PRI	NULL	
AirlineCode	varchar(10)	YES	MUL	NULL	
BusinessClassIndicator	tinyint(1)	YES		NULL	
SmokingAllowedIndicator	tinyint(1)	YES		NULL	

7. **FlightAvailability Table**

Query :

```
CREATE TABLE FlightAvailability (
    FlightNumber VARCHAR(10),
    DepartureDateTime DATETIME,
    TotalSeatsBusiness INT,
    BookedSeatsBusiness INT,
    TotalSeatsEconomy INT,
    BookedSeatsEconomy INT,
    PRIMARY KEY (FlightNumber, DepartureDateTime),
    FOREIGN KEY (FlightNumber) REFERENCES Flight(FlightNumber)
);
desc FlightAvailability;
```

Output:

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Field	Type	Null	Key	Default	Extra
FlightNumber	varchar(10)	NO	PRI	NULL	
DepartureDateTime	datetime	NO	PRI	NULL	
TotalSeatsBusiness	int	YES		NULL	
BookedSeatsBusiness	int	YES		NULL	
TotalSeatsEconomy	int	YES		NULL	
BookedSeatsEconomy	int	YES		NULL	

8. **BookingOffice Table**

Query :

```
CREATE TABLE BookingOffice (
    OfficeID INT PRIMARY KEY,
    CityName VARCHAR(50),
    FOREIGN KEY (CityName) REFERENCES City(CityName)
```

```
);

desc BookingOffice;
```

Output:

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Field	Type	Null	Key	Default	Extra
OfficeID	int	NO	PRI	NULL	
CityName	varchar(50)	YES	MUL	NULL	

9. **Booking Table**

Query :

```
CREATE TABLE Booking (
    BookingNumber INT PRIMARY KEY AUTO_INCREMENT,
    BookingCity VARCHAR(50),
    BookingDate DATE,
    FlightNumber VARCHAR(10),
    DepartureDateTime DATETIME,
    ArrivalDateTime DATETIME,
    ClassIndicator VARCHAR(10),
    TotalPrice DECIMAL(10, 2),
    StatusIndicator VARCHAR(10),
    CustomerID INT,
    AmountPaid DECIMAL(10, 2),
    OutstandingBalance DECIMAL(10, 2),
    TicketFirstName VARCHAR(50),
    TicketLastName VARCHAR(50),
    FOREIGN KEY (FlightNumber) REFERENCES Flight(FlightNumber),
```

```
    FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID)
);
desc Booking;
```

Output:

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Field	Type	Null	Key	Default	Extra
BookingNumber	int	NO	PRI	NULL	auto_increment
BookingCity	varchar(50)	YES		NULL	
BookingDate	date	YES		NULL	
FlightNumber	varchar(10)	YES	MUL	NULL	
DepartureDateTime	datetime	YES		NULL	
ArrivalDateTime	datetime	YES		NULL	
ClassIndicator	varchar(10)	YES		NULL	
TotalPrice	decimal(10,2)	YES		NULL	
StatusIndicator	varchar(10)	YES		NULL	
CustomerID	int	YES	MUL	NULL	
AmountPaid	decimal(10,2)	YES		NULL	
OutstandingBalance	decimal(10,2)	YES		NULL	
TicketFirstName	varchar(50)	YES		NULL	
TicketLastName	varchar(50)	YES		NULL	

10. **AirportTax Table**

Query :

```
CREATE TABLE AirportTax (
    CityName VARCHAR(50),
    Country VARCHAR(50),
    TaxAmount DECIMAL(10, 2),
    PRIMARY KEY (CityName),
    FOREIGN KEY (CityName) REFERENCES City(CityName)
);
desc AirportTax;
```

Output:

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Field	Type	Null	Key	Default	Extra
CityName	varchar(50)	NO	PRI	NULL	
Country	varchar(50)	YES		NULL	
TaxAmount	decimal(10,2)	YES		NULL	

11. **ExchangeRateTable**

Query :

```
CREATE TABLE ExchangeRate (
    Currency VARCHAR(10),
    Rate DECIMAL(10, 4),
    Date DATE,
    PRIMARY KEY (Currency, Date)
);
desc ExchangeRate;
```

Output:

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Field	Type	Null	Key	Default	Extra
Currency	varchar(10)	NO	PRI	NULL	
Rate	decimal(10,4)	YES		NULL	
Date	date	NO	PRI	NULL	

INSERTION OF DATA INTO TABLES

1. **Airline Table**

Query:

```
INSERT INTO Airline (AirlineCode, AirlineName, Country)
VALUES
('AC', 'AirCan', 'Canada'),
```

('US', 'USAir', 'USA'),

('BA', 'BritAir', 'UK'),

('AF', 'AirFrance', 'France'),

('LA', 'LuftAir', 'Germany'),

('IA', 'ItalAir', 'Italy');

Select * from Airline;

Output:

Result Grid | Filter Rows:

AirlineCode	AirlineName	Country
AC	AirCan	Canada
AF	AirFrance	France
BA	BritAir	UK
IA	ItalAir	Italy
LA	LuftAir	Germany
US	USAir	USA
NULL	NULL	NULL

2. **City Table**

Query:

INSERT INTO City (CityName, Country)

VALUES

('Toronto', 'Canada'),

('Montreal', 'Canada'),

('New York', 'USA'),

('Chicago', 'USA'),

('London', 'UK'),

('Edinburgh', 'UK'),

('Paris', 'France'),

('Nice', 'France'),

('Bonn', 'Germany'),

('Berlin', 'Germany'),

('Rome', 'Italy'),

('Naples', 'Italy');

select * from city;

Output:

Result Grid | Filter Rows:

CityName	Country
Berlin	Germany
Bonn	Germany
Chicago	USA
Edinburgh	UK
London	UK
Montreal	Canada
Naples	Italy
New York	USA
Nice	France
Paris	France
Rome	Italy
Toronto	Canada
NULL	NULL

3. **MailingAddress Table**

Query:

INSERT INTO MailingAddress (Street, CityName, ProvinceState, PostalCode, Country)

VALUES

('123 Queen St', 'Toronto', 'ON', 'M5V1Z4', 'Canada'),

('456 Rue St', 'Montreal', 'QC', 'H2Y1W9', 'Canada'),

('789 Broadway', 'New York', 'NY', '10003', 'USA'),

('101 Michigan Ave', 'Chicago', 'IL', '60611', 'USA'),

('102 Baker St', 'London', 'England', 'SW1A 2DR', 'UK'),

('103 Princes St', 'Edinburgh', 'Scotland', 'EH2 2ER', 'UK');

select * from MailingAddress;

Output:

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

MailingAddressID	Street	CityName	ProvinceState	PostalCode	Country
1	123 Queen St	Toronto	ON	M5V1Z4	Canada
2	456 Rue St	Montreal	QC	H2Y1W9	Canada
3	789 Broadway	New York	NY	10003	USA
4	101 Michigan Ave	Chicago	IL	60611	USA
5	102 Baker St	London	England	SW1A 2DR	UK
6	103 Princes St	Edinburgh	Scotland	EH2 2ER	UK
NULL	NULL	NULL	NULL	NULL	NULL

4. **Customer Table**

Query:

```
INSERT INTO Customer (FirstName, LastName, MailingAddressID, Email)

VALUES

('Alice', 'Smith', 1, 'alice.smith@example.com'),

('Bob', 'Jones', 2, 'bob.jones@example.com'),

('Charlie', 'Brown', 3, 'charlie.brown@example.com'),

('David', 'White', 4, 'david.white@example.com'),

('Eve', 'Black', 5, 'eve.black@example.com'),

('Frank', 'Green', 6, 'frank.green@example.com');

select * from Customer;
```

Output:

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

CustomerID	FirstName	LastName	MailingAddressID	Email
1	Alice	Smith	1	alice.smith@example.com
2	Bob	Jones	2	bob.jones@example.com
3	Charlie	Brown	3	charlie.brown@example.com
4	David	White	4	david.white@example.com
5	Eve	Black	5	eve.black@example.com
6	Frank	Green	6	frank.green@example.com
NULL	NULL	NULL	NULL	NULL

5. **PhoneFaxNumber Table**

Query:

INSERT INTO PhoneFaxNumber (CustomerID, CountryCode, AreaCode, LocalNumber) VALUES

(1, '+1', '416', '5551234'),

(2, '+1', '514', '5555678'),

(3, '+1', '212', '5557890'),

(4, '+1', '312', '5553456'),

(5, '+44', '20', '5556789'),

(6, '+44', '131', '5559876');

select * from PhoneFaxNumber;

Output:

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap

PhoneFaxID	CustomerID	CountryCode	AreaCode	LocalNumber
1	1	+1	416	5551234
2	2	+1	514	5555678
3	3	+1	212	5557890
4	4	+1	312	5553456
5	5	+44	20	5556789
6	6	+44	131	5559876
NULL	NULL	NULL	NULL	NULL

6. **BookingOffice Table**

Query:

INSERT INTO BookingOffice (OfficeID, CityName) VALUES

(1, 'Toronto'), (2, 'Montreal'), (3, 'New York'), (4, 'Chicago'), (5, 'London'),

(6, 'Edinburgh'), (7, 'Paris'), (8, 'Nice'), (9, 'Bonn'), (10, 'Berlin'), (11, 'Rome'),

(12, 'Naples');

select * from BookingOffice;

Output:

Result Grid | Filter Rows: | Edit:

OfficeID	CityName
10	Berlin
9	Bonn
4	Chicago
6	Edinburgh
5	London
2	Montreal
12	Naples
3	New York
8	Nice
7	Paris
11	Rome
1	Toronto
NULL	NULL

7. **Flight Table**

Query:

```
INSERT INTO Flight (FlightNumber, AirlineCode, BusinessClassIndicator,
SmokingAllowedIndicator)

VALUES

('AC101', 'AC', TRUE, FALSE),

('US202', 'US', TRUE, FALSE),

('BA303', 'BA', FALSE, FALSE),

('AF404', 'AF', TRUE, FALSE),

('LA505', 'LA', FALSE, FALSE),

('IA606', 'IA', TRUE, FALSE);

select * from Flight;
```

Output:

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap C

FlightNumber	AirlineCode	BusinessClassIndicator	SmokingAllowedIndicator
AC101	AC	1	0
AF404	AF	1	0
BA303	BA	0	0
IA606	IA	1	0
LA505	LA	0	0
US202	US	1	0
NULL	NULL	NULL	NULL

8. **Booking Table**

Query:

INSERT INTO Booking (BookingCity, BookingDate, FlightNumber, DepartureDateTime, ArrivalDateTime, ClassIndicator, TotalPrice, StatusIndicator, CustomerID, AmountPaid, OutstandingBalance, TicketFirstName, TicketLastName)

VALUES

('Toronto', '2024-09-01', 'AC101', '2024-09-20 08:00:00', '2024-09-20 12:00:00', 'Business', 750.00, 'Booked', 1, 750.00, 0.00, 'Alice', 'Smith'),

('Montreal', '2024-09-02', 'US202', '2024-09-21 09:00:00', '2024-09-21 13:00:00', 'Economy', 500.00, 'Booked', 2, 250.00, 250.00, 'Bob', 'Jones'),

('New York', '2024-09-03', 'BA303', '2024-09-22 10:00:00', '2024-09-22 14:00:00', 'Economy', 400.00, 'Scratched', 3, 100.00, 300.00, 'Charlie', 'Brown');

select * from Booking;

Output:

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

BookingNumber	BookingCity	BookingDate	FlightNumber	DepartureDateTime	ArrivalDateTime	ClassIndicator	TotalPrice	StatusIndicator	CustomerID	AmountPaid	OutstandingBalance	TicketFirstName	TicketLastName
1	Toronto	2024-09-01	AC101	2024-09-20 08:00:00	2024-09-20 12:00:00	Business	750.00	Booked	1	750.00	0.00	Alice	Smith
2	Montreal	2024-09-02	US202	2024-09-21 09:00:00	2024-09-21 13:00:00	Economy	500.00	Booked	2	250.00	250.00	Bob	Jones
3	New York	2024-09-03	BA303	2024-09-22 10:00:00	2024-09-22 14:00:00	Economy	400.00	Scratched	3	100.00	300.00	Charlie	Brown
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

9. **AirportTaxTable**

Query:

INSERT INTO AirportTax (CityName, Country, TaxAmount)

VALUES

('Toronto', 'Canada', 25.00),

('New York', 'USA', 30.00),

('London', 'UK', 20.00);

select * from AirportTax;

Output:

Result Grid | Filter Rows: | Edit:

CityName	Country	TaxAmount
London	UK	20.00
New York	USA	30.00
Toronto	Canada	25.00
NULL	NULL	NULL

5. Query the Database:

1. Display the airline code and airline name of all airlines in the system.

Query :

SELECT AirlineCode, AirlineName

FROM Airline;

Output:

Result Grid | Filter Rows: | Edit:

AirlineCode	AirlineName
AC	AirCan
AF	AirFrance
BA	BritAir
IA	ItalAir
LA	LuftAir
US	USAir
NULL	NULL

2. Find the names of all citie Query located in Canada.

Query :

SELECT CityName

FROM City

WHERE Country = 'Canada';

Output:

Result Grid

CityName
Montreal
Toronto
NULL

3. Retrieve all booking office IDs and the city they are located in.

Query :

SELECT OfficeID, CityName

FROM BookingOffice;

Output:

Result Grid | Filter Rows: | Edit:

OfficeID	CityName
10	Berlin
9	Bonn
4	Chicago
6	Edinburgh
5	London
2	Montreal
12	Naples
3	New York
8	Nice
7	Paris
11	Rome
1	Toronto
NULL	NULL

4. Find the first name, last name, and email address of all customers.

Query :

SELECT FirstName, LastName, Email

FROM Customer;

Output:

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

FirstName	LastName	Email
Alice	Smith	alice.smith@example.com
Bob	Jones	bob.jones@example.com
Charlie	Brown	charlie.brown@example.com
David	White	david.white@example.com
Eve	Black	eve.black@example.com
Frank	Green	frank.green@example.com

5. List all flights that are operated by 'AirCan'. Show the flight number and business class indicator.

Query :

```
SELECT FlightNumber, BusinessClassIndicator

FROM Flight

WHERE AirlineCode = 'AC';
```

Output:

Result Grid | Filter Rows: | Edit:

FlightNumber	BusinessClassIndicator
AC101	1
NULL	NULL

6. List all uni ue countries where airlines operate.

Query :

```
SELECT DISTINCT Country

FROM Airline;
```

Output:

Result Grid | Filter Rows: | Export:

CountryName
Canada
France
Germany
Italy
UK
USA

7. List the flight numbers for all flights operated by 'AirFrance'. Assume `AirlineCode` is referenced in `Flight`.

Query :

```
SELECT FlightNumber

FROM Flight

WHERE AirlineCode = (SELECT AirlineCode FROM Airline WHERE AirlineName = 'AirFrance');
```

Output:

Result Grid | Filter Rows: | Edit:

FlightNumber
AF404
NULL

8. Retrieve the first name, last name, and email of all customers who live in 'New York'. Assume `MailingAddressID` in `Customer` references the `MailingAddress` table.

Query :

SELECT FirstName, LastName, Email

FROM Customer

WHERE MailingAddressID = (SELECT MailingAddressID FROM MailingAddress WHERE CityName = 'New York');

Output:

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

FirstName	LastName	Email
Charlie	Brown	charlie.brown@example.com

9. Display the flight numbers where the number of booked seats in economy class i Query ess than the total available seats.

Query :

SELECT FlightNumber

FROM FlightAvailability

WHERE TotalSeatsEconomy > BookedSeatsEconomy;

Output:

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

FlightNumber
AC101
AF404
BA303
IA606
LA505
US202

10. List the first name and last name of all customers who have booked flight 'AC101'.

Query :

SELECT FirstName, LastName

FROM Customer

WHERE CustomerID IN (SELECT CustomerID FROM Booking WHERE FlightNumber = 'AC101');

Output:

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

FirstName	LastName
Alice	Smith