

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

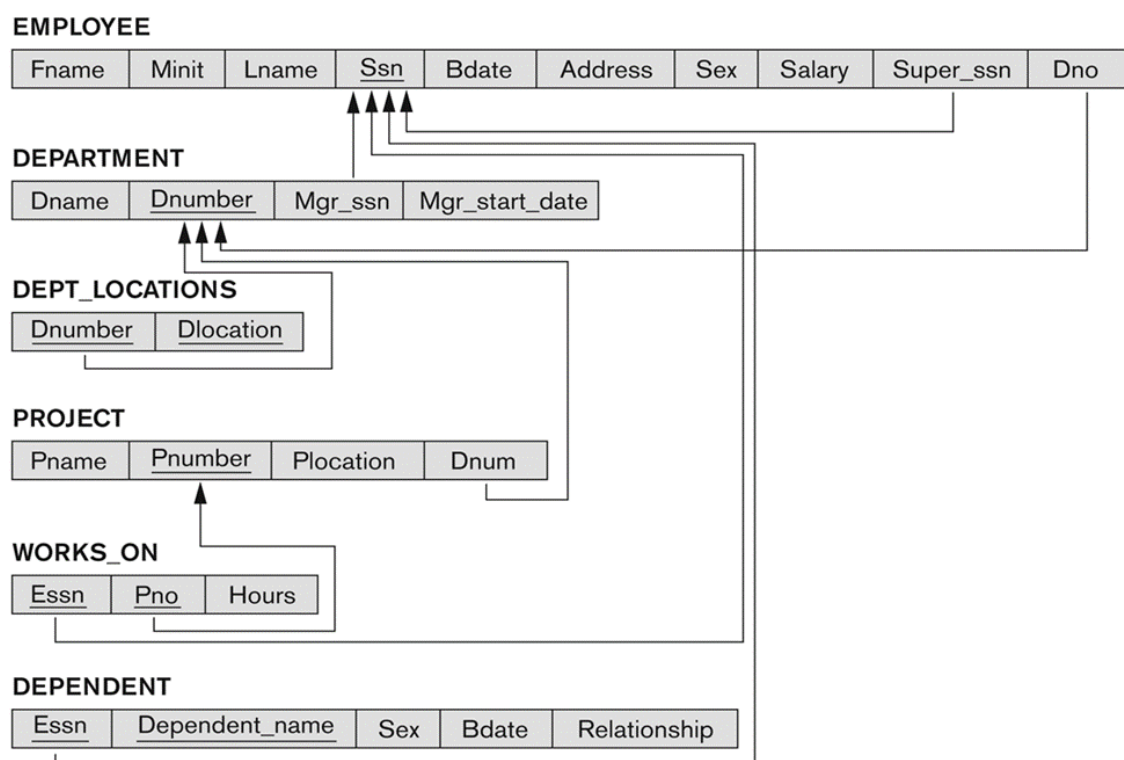
E.g.: SELECT, INSERT, UPDATE, DELETE

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	

ALTER:

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	

DROP:

Used to delete the entire table structure.

SYNTAX:

DROP table <table_name>;

EXAMPLE:

DROP table WORKS_ON;

TRUNCATE:

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	

CHECK:

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

SYNTAX:

```
col_name datatype(size) CHECK (condition);
```

NOT NULL:

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

SYNTAX:

```
col_name datatype(size) NOT NULL;
```

UNIQUE:

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
Filter...	Filter..	Filter...	Filter	Filter...	Filter...	Filter...	Filter.	Filter...	Filte
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
abc Filter...	abc Filter...	abc Filter...	abc 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
abc Filter...	abc Filter...	abc Filter...	abc 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
abc Filter...	abc Filter...	abc Filter...	abc Filter...	abc Filter...	abc Filter...	abc Filter...	abc Filter...	abc Filter...	abc Filter...
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
abc Filter...	abc Filter...	abc Filter...	abc Filter...	abc Filter...	abc Filter...	abc Filter...	abc Filter...	abc Filter...	abc Filter...
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
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

DELETE:

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
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
abc Filter..	abc Filter	abc Filter...	abc Filter...	abc Filter...	abc Filter...	abc Filter...	abc Filter.	abc Filter...	abc 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
abc Filter..	abc Filter	abc Filter..	abc Filt	abc Filter	abc Filter...	abc Filter...	abc Filter.	abc Filter...	abc Filte
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.

COMMIT:

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...
abc Filter...	abc Filter.	abc Filter...	abc Filter...	abc Filter...	abc Filter...	abc Filter...	abc Filter.	abc Filter...	abc 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
abc Filter.	abc Filter	abc Filter...	abc Filter	abc Filter..	abc Filter...	abc Filter...	abc Filter..	abc Filter...	abc Filti
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
abc Filter..	abc Filter	abc Filter..	abc Filt	abc Filter	abc Filter...	abc Filter...	abc Filter.	abc Filter...	abc Filti
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
abc Filter...	abc Filter.	abc Filter..	abc Filter..	abc Filter..	abc Filter...	abc Filter...	abc Filter.	abc Filter...	abc Filti
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
abc Filter...	abc Filter	abc Filter.	abc Filter.	abc Filter...	abc Filter...	abc Filter...	abc Filter.	abc Filter...	abc Filt
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
abc Filter..	abc Filter	abc Filter..	abc Filter.	abc Filter...	abc Filter...	abc Filter...	abc Filter..	abc Filter...	abc 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
abc Filter.	abc Filter.	abc Filter.	abc Filter..	abc Filter...	abc Filter...	abc Filter...	abc Filter.	abc Filter...	abc 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
abc Filter.	abc Filter	abc Filter...	abc Filter..	abc Filter...	abc Filter...	abc Filter...	abc Filter..	abc Filter...	abc Filtr
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
abc Filter..	abc Filter	abc Filter.	abc Filter	abc Filter	abc Filter...	abc Filte	abc Fi	abc Filtr	abc Filtr
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
abc Filter.	abc Filter	abc Filter...	abc Filter...	abc Filter...	abc Filter...	abc Filter...	abc Filter.	abc Filter...	abc Filtr
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
abc Filter..	abc Filter	abc Filter.	abc Filter...	abc Filter...	abc Filter...	abc Filter...	abc Filter..	abc Filter...	abc Filtr
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
abc Filter...	abc Filter	abc Filter...	abc Filtr	abc Filter.	abc Filter...	abc Filter...	abc Filter..	abc Filter...	abc Filtr
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
abc Filter...	abc Filter...	abc Filter...	abc Filter...	abc Filter...	abc Filter...	abc Filter...	abc Filter...	abc Filter...	abc Filter...
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston TX	M	35000	333445555	5

ORDER BY:

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
abc Filter...	abc Filter...	abc Filter...	abc Filter...	abc Filter...	abc Filter...	abc Filter...	abc Filter...	abc Filter...	abc Filter...
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
abc Filter...	abc Filter...	abc Filter...	abc Filter...	abc Filter...	abc Filter...	abc Filter...	abc Filter...	abc Filter...	abc Filter...
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...	 Filter...
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...	 Filter...
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

Date & Time Functions:

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

Math Functions:

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
abc Filter...	abc Filter...	abc 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
abc Filter...	abc Filter...	abc 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
abc Filter...	abc Filter...	abc 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
abc Filter...	abc 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
abc Filter...	abc Filter...	abc 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
abc Filter...	abc Filter...
5	4

ROLLUP Operator:

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
abc Filter...	abc Filter...	abc 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
abc Filter...	abc 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
abc Filter...	abc Filter...	abc 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
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

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
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

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

USING Clause:

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
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

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
abc Filter...	abc Filter...	abc 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
abc Filter...	abc Filter...	abc 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
abc Filter...	abc Filter...	abc 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
abc Filter...	abc 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 DROPPING 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
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
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
abc Filter...	abc Filter...	abc 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
abc Filter...	abc 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
abc Filter...	abc Filter...	abc 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
abc Filter...	abc Filter...	abc 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
abc Filter...	abc Filter...	abc 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.

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(
```

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(

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(

Calling a Stored Procedure:

1. -- 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...	 Fil...	 Filter...	 Filter...	 Filter...	 Filter...	 Filter...	 Filt...
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. -- Updating salary for a specific employee

CALL UpdateEmployeeSalary(000000000, 50000);

SELECT * FROM EMPLOYEE;

Fname	Minit	Lname	ssn	Bdate	Address	Gender	Salary	super_ssn	Dno
<input type="text" value="a Filter..."/>	<input type="text" value="a Filter..."/>	<input type="text" value="a Filter..."/>	<input type="text" value="a Filter..."/>	<input type="text" value="a Filter..."/>	<input type="text" value="a Filter..."/>	<input type="text" value="a Filter..."/>	<input type="text" value="a Filter..."/>	<input type="text" value="a Filter..."/>	<input type="text" value="a Filter..."/>
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. -- Deleting an employee

CALL DeleteEmployee(123456789);

SELECT * FROM EMPLOYEE;

Fname ↑	Minit	Lname	ssn	Bdate	Address	Gender	Salary	super_ssn	Dno
<input type="text" value="a Filter..."/>	<input type="text" value="a Filter..."/>	<input type="text" value="a Filter..."/>	<input type="text" value="a Filter..."/>	<input type="text" value="a Filter..."/>	<input type="text" value="a Filter..."/>	<input type="text" value="a Filter..."/>	<input type="text" value="a Filter..."/>	<input type="text" value="a Filter..."/>	<input type="text" value="a Filter..."/>
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

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:

```
-- 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

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
abc Filter...	abc Filter	abc Filter...	abc Filter	abc Filter..	abc Filter...	abc Filter...	abc Filter.	abc Filter...	abc 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