

Create Database:-

Create database bakery;

Create Table:-

```
CREATE TABLE customer(id int PRIMARY KEY AUTO_INCREMENT,name
varchar(255),email varchar(255),password varchar (255),mobile_no bigint(11)
);
```

Value Insert:-

```
INSERT INTO customer(id, name, email, password, mobile_no) VALUES
(1,"Jil","jp@gmail.com",'jil23554',9876543210),
(2,"Dhruvin","dp@gmail.com",'dp3554',9876543211),
(3,"Zainil","zp@gmail.com",'zp23554',9876543215);
```

```
CREATE TABLE seller(id int PRIMARY KEY AUTO_INCREMENT,name varchar(255),email
varchar(255),password varchar (255),mobile_no bigint(11)
);
```

```
INSERT INTO seller(id, name, email, password, mobile_no) VALUES
(1,"Jil","jp56@gmail.com",'jil235854',9876543270),
(2,"Dhruvin","dp45@gmail.com",'dp35j54',9876543220),
(3,"Zainil","zp2@gmail.com",'zp235ap54',9876543265)
(4,"Ket","kp56@gmail.com",'kp235854',5876543270);
```

```
CREATE TABLE product(p_id int PRIMARY KEY AUTO_INCREMENT,p_name
varchar(255),p_mfd int
);
```

Database:

```
CREATE DATABASE cake_shop;
```

Table:

```
CREATE TABLE customer (id int PRIMARY KEY AUTO_INCREMENT,name  
varchar(255),mobile_no bigint(11),email varchar(255));
```

```
INSERT INTO customer(name, mobile_no, email) VALUES  
("Jil",'9876543210','jp@gmail.com'),('Dhruvin','9876543214','dp@gmail.com'),('Za  
inil','9876543215','zp@gmail.com');
```

Foreign Key:

```
CREATE TABLE feedback (id int PRIMARY KEY AUTO_INCREMENT,cust_id int(11),  
FOREIGN KEY (cust_id) REFERENCES customer(id),feedback varchar(255));
```

```
INSERT INTO feedback(cust_id, feedback) VALUES (1,"Good Service"),(2,"Average  
Service");
```

Alter Table:

```
ALTER TABLE customer add(purchase_date date);
```

```
ALTER TABLE customer CHANGE COLUMN email email_id VARCHAR(255);
```

Update:

```
UPDATE customer set name="Jil Patel"where id=1;
```

JOINTS:-

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

Types of join:3 Types

1) inner Join/ Join ==> This is Main.

2) outer join ==> Left Outer Join/Right Outer Join/Full Join

3) cross join

1) Inner Join/Join ==> This is Main.- Very Important

- customer		Feedabck
cust_id	PK	fed_id PK
cust_name		cust_id FK
pass		msg

- select * from customer join feedback on customer.id = feedback.cust_id

- select feedback.*,customers.name from customer join feedback on
customer.id = feedback.cust_id

customer	order	product
cust_id	order_id	prod_id
cust_name	cust_id	pro_name
pass	prod_id	pro_price

- select * from order join customer on order.cust_id=customer.id
join product on order.prod_id=product.prod_id

2) Outer Join

- Left Outer Join

select * from customer left outer join feedback on customer.id =feedback.uid

- Right Outer Join

select * from customer right outer join feedback on customer.id =feedback.uid

- Full join

select * from user_tbl full join feedback

select * from customer full join feedback

3) Cross Join

- select * from customer cross join feedback

Distinct:-

- The SELECT DISTINCT statement is used to return only distinct (different) values.
- SELECT DISTINCT name FROM customer; //It show only name

ORDER BY:-

- The ORDER BY keyword is used to sort the result-set in ascending or descending order.
- SELECT * FROM Products ORDER BY Price;
- SELECT * FROM customer ORDER BY email_id; //Show alphabetic order A-Z

AND Operator:-

- The WHERE clause can contain one or many AND operators.
- The AND operator is used to filter records based on more than one condition, like if you want to return all customers from Spain that starts with the letter 'G':
- SELECT * FROM Customers WHERE Country = 'Spain' AND CustomerName LIKE 'G%';
- SELECT * FROM customer WHERE email_id = 'jp@gmail.com' AND name LIKE 'j%';

OR Operator:-

- The WHERE clause can contain one or more OR operators.
- The OR operator is used to filter records based on more than one condition, like if you want to return all customers from Germany but also those from Spain:
- SELECT * FROM Customers WHERE Country = 'Germany' OR Country = 'Spain';
- SELECT * FROM customer WHERE email_id = 'jp@gmail.com' OR name LIKE 'j%';

NOT Operator:-

- The NOT operator is used in combination with other operators to give the opposite result, also called the negative result.
- In the select statement below we want to return all customers that are NOT from Spain:
- `SELECT * FROM Customers WHERE NOT Country = 'Spain';`
- `SELECT * FROM customer WHERE NOT email_id = "jp@gmail.com" ;`
- `SELECT * FROM customer WHERE name NOT LIKE 'A%';`
- `SELECT * FROM customer WHERE name NOT LIKE 'Z%';`
- `SELECT * FROM customer WHERE NOT id > 4;`
- `SELECT * FROM customer WHERE NOT id < 4;`
- `SELECT * FROM customer WHERE id NOT BETWEEN 3 AND 5 ;`

Limit:-

- `SELECT column_name(s) FROM table_name WHERE condition LIMIT number;`
- `SELECT * FROM Customers LIMIT 3;`
- `SELECT * FROM customer LIMIT 2;`

Aggregate Functions:-

- `MIN()` - returns the smallest value within the selected column
- `MAX()` - returns the largest value within the selected column
- `COUNT()` - returns the number of rows in a set
- `SUM()` - returns the total sum of a numerical column
- `AVG()` - returns the average value of a numerical column
- `SELECT COUNT(name) FROM customer;`
- `SELECT SUM(price) FROM customer;`

IN Operator

- The IN operator allows you to specify multiple values in a WHERE clause.

-The IN operator is a shorthand for multiple OR conditions.

- SELECT * FROM Customers WHERE Country IN ('Germany', 'France', 'UK')

LIKE Operator:-

- The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.

- There are two wildcards often used in conjunction with the LIKE operator:

- The percent sign % represents zero, one, or multiple characters

- The underscore sign _ represents one, single character

- For Use Search

- SELECT * FROM customer WHERE name LIKE 'j%';

- SELECT * FROM customer WHERE name LIKE '%l';

- SELECT * FROM customer WHERE name LIKE '%l%';

BETWEEN Operator

-The BETWEEN operator selects values within a given range. The values can be numbers, text, or dates.

- The BETWEEN operator is inclusive: begin and end values are included.

- SELECT * FROM Products WHERE Price BETWEEN 10 AND 20;

- SELECT * FROM customer WHERE id BETWEEN 1 AND 5;

Aliases

- SQL aliases are used to give a table, or a column in a table, a temporary name.

- Aliases are often used to make column names more readable.

- An alias only exists for the duration of that query.

- An alias is created with the AS keyword.

- SELECT CustomerID AS ID FROM Customers;

- SELECT id AS Customer_Id FROM customer;

Views (Security Concept/ sub menu virtual table) Exa: (BANK DUPLICATE TABLE):-

-In SQL, a view is a virtual table based on the result-set of an SQL statement.

-A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.

-You can add SQL functions, WHERE, and JOIN statements to a view and present the data as if the data were coming from one single table.

-View is a sub-table. If any changes in this table then changes applied automatically in main table.

-CREATE VIEW customer_view AS SELECT id,mobile_no,email_id FROM customer;

id	mobile_no	email_id
1	9876543210	jp1@gmail.com
2	9876543214	dp@gmail.com
3	9876543215	zp@gmail.com
4	1234567892	kp@gmail.com
5	9876543225	jack@gmail.com
9	1234567891	jay@gmail.com
10	123456781	jayp@gmail.com

Procedure :

-A stored procedure is a prepared SQL code that you can save, so the code can be reused over and over again. So if you have an SQL query that you write over and over again, save it as a stored procedure, and then just call it to execute it.

-Write delimiter // compulsory

-CREATE PROCEDURE insert_cus(

IN name varchar(255),

IN mobile_no bigint(11),

IN email_id varchar(255)

)

BEGIN

insert into customer(name,mobile_no,email_id) values(name,mobile_no,email_id);

END

-CALL insert_cus('jay','1234567891','jay@gmail.com');

id	name	mobile_no	email_id	purchase_date
1	Jil Patel	9876543210	jp1@gmail.com	2025-04-03
2	Dhruvin	9876543214	dp@gmail.com	NULL
3	Zainil	9876543215	zp@gmail.com	NULL
4	Ket	1234567892	kp@gmail.com	2025-03-01
5	Jack	9876543225	jack@gmail.com	2025-01-01
9	jay	1234567891	jay@gmail.com	NULL
10	jay patel	123456781	jayp@gmail.com	NULL

Trigger :

-A MySQL trigger is a stored program (with queries) which is executed automatically to respond to a specific event such as insertion, updation or deletion occurring in a table.

-BEFORE INSERT – activated before data is inserted into the table.

-AFTER INSERT- activated after data is inserted into the table.

-BEFORE UPDATE – activated before data in the table is updated.

-AFTER UPDATE - activated after data in the table is updated.

-BEFORE DELETE – activated before data is removed from the table.

-AFTER DELETE – activated after data is removed from the table

Insert Trigger:

```
-create table customer_log( name varchar(255),mobile_no bigint(11),email_id
varchar(255),entry_date_time datetime);
```




```
CREATE TRIGGER insert_trigger_cuslog BEFORE INSERT ON customer FOR EACH ROW
BEGIN
```

```
insert into customer_log(name,mobile_no,email_id,Entry_date_time) values
(new.name,new.mobile_no,new.email_id,now());
```

```
END//
```

```
-CALL insert_cus("jay patel",'123456781',"jayp@gmail.com");
```

name	mobile_no	email_id	entry_date_time
jay patel	123456781	jayp@gmail.com	2025-05-07 12:10:25

Name	Time	Event
<input type="checkbox"/> insert_trigger_cuslog	BEFORE	INSERT  Edit  Export  Drop

Delete Trigger:

- create table customer_delete_log(name varchar(255),mobile_no
bigint(11),email_id varchar(255),entry_date_time datetime);

CREATE TRIGGER delete_trigger_cuslog BEFORE DELETE ON customer FOR EACH
ROW

BEGIN

insert into customer_delete_log(name,mobile_no,email_id,Entry_date_time) values
(old.name,old.mobile_no,old.email_id,now());

END//

name	mobile_no	email_id	entry_date_time
jay patel	123456781	jayp@gmail.com	2025-05-07 12:20:55

- CREATE TRIGGER after_delete_trigger_cuslog AFTER DELETE ON customer FOR
EACH ROW

BEGIN

insert into customer_delete_log(name,mobile_no,email_id,Entry_date_time) values
(old.name,old.mobile_no,old.email_id,now());

END//

name	mobile_no	email_id	entry_date_time
jay patel	123456781	jayp@gmail.com	2025-05-07 12:20:55
jay	1234567891	jay@gmail.com	2025-05-07 12:25:09
jay	1234567891	jay@gmail.com	2025-05-07 12:25:09

Update Trigger:

- create table customer_update_log(name varchar(255),mobile_no
bigint(11),email_id varchar(255),entry_date_time datetime);

CREATE TRIGGER before_update_trigger_cuslog BEFORE UPDATE ON customer FOR
EACH ROW

BEGIN

INSERT INTO customer_update_log(name, mobile_no, email_id, entry_date_time)
VALUES (OLD.name, OLD.mobile_no, OLD.email_id, NOW());

END//

name	mobile_no	email_id	entry_date_time
Jack	9876543225	jack@gmail.com	2025-05-07 12:29:08

id	name	mobile_no	email_id	purchase_date
1	Jil Patel	9876543210	jp1@gmail.com	2025-04-03
2	Dhruvin	9876543214	dp@gmail.com	NULL
3	Zainil	9876543215	zp@gmail.com	NULL
4	Ket	1234567892	kp@gmail.com	2025-03-01
5	Jack Peter	9876543225	jack@gmail.com	2025-01-01

After Update:

- CREATE TRIGGER after_update_trigger_cuslog AFTER UPDATE ON customer FOR EACH ROW

BEGIN

INSERT INTO customer_update_log(name, mobile_no, email_id, entry_date_time)
VALUES (NEW.name, NEW.mobile_no, NEW.email_id, NOW());

END//

name	mobile_no	email_id	entry_date_time
Jack	9876543225	jack@gmail.com	2025-05-07 12:29:08
Jack Peter	9876543225	jack@gmail.com	2025-05-07 12:31:39
Jack	9876543225	jack@gmail.com	2025-05-07 12:31:39