

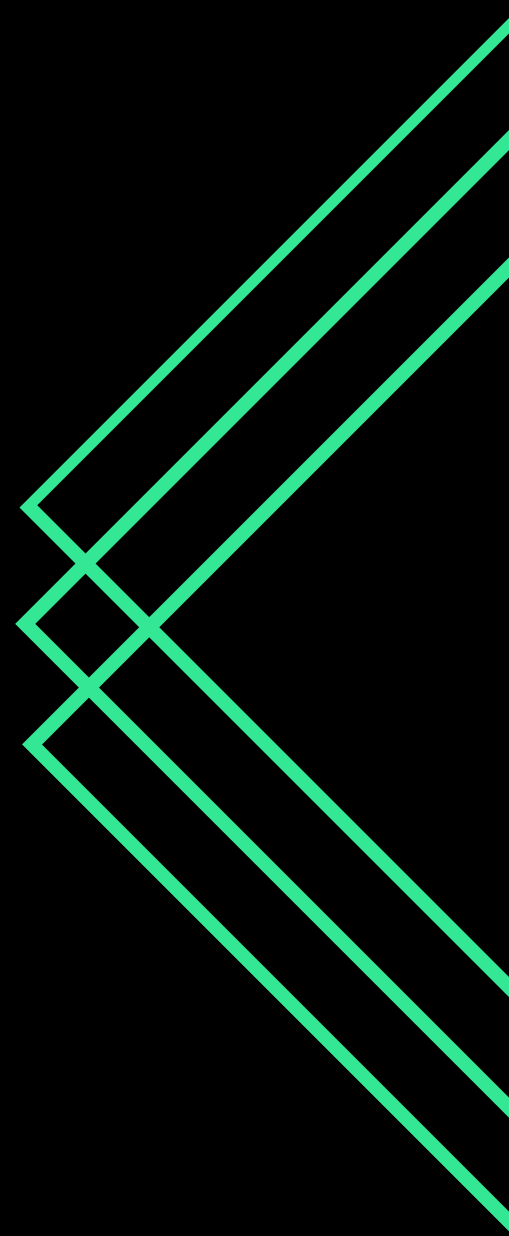
May 2022

DBMS FINAL PROJECT REPORT



GROUP 57

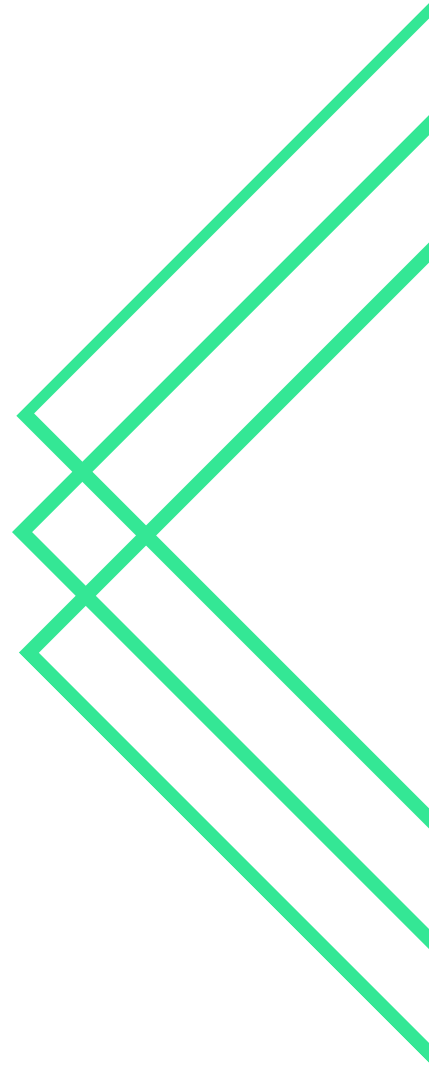
AAYUSH KUMAR - 2020008
ANKIT CHAURASIA- 2020027
DISHANT YADAV - 2020057
NITTIN YADAV - 2020093



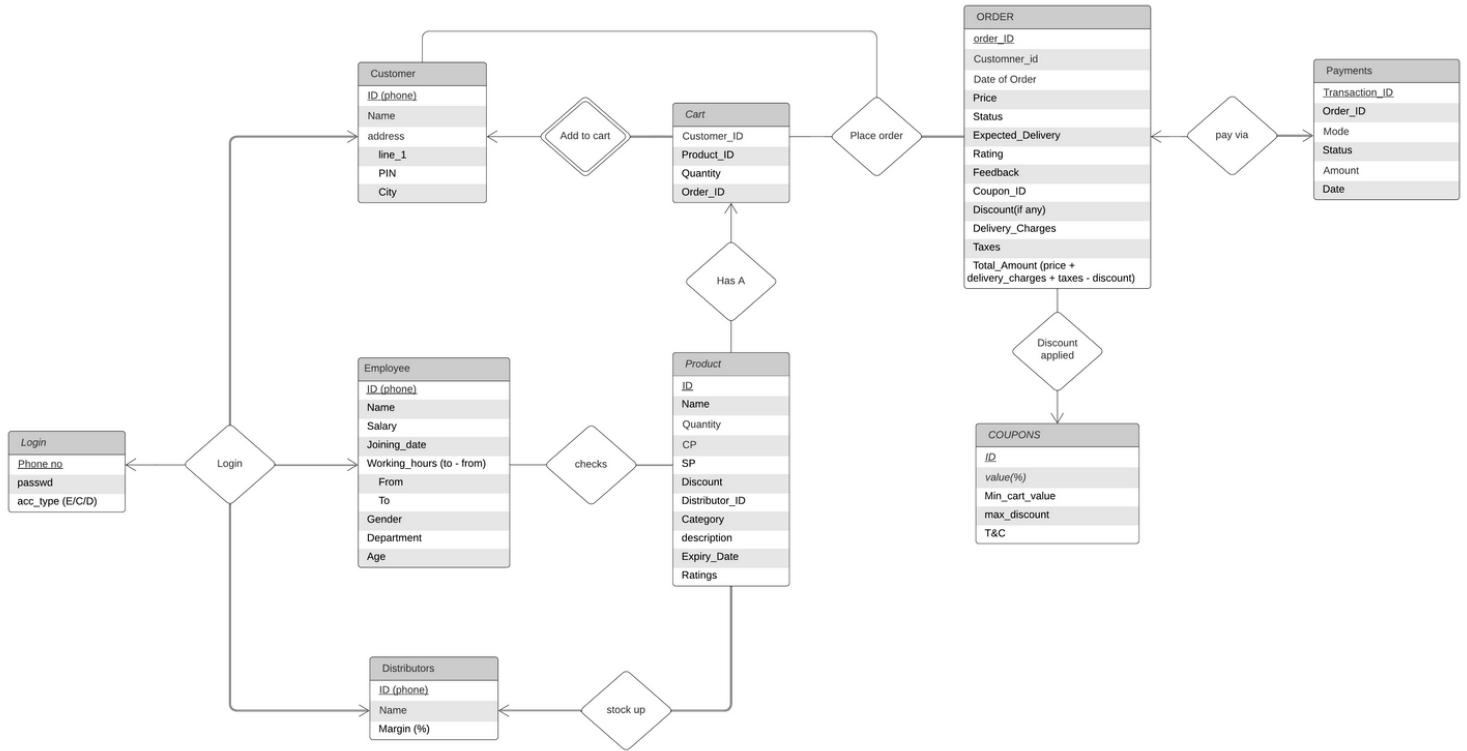
ABOUT


This report consists of the deliverables of the project assigned to us in course CSE-202 of an **Online Retail Store System**.

This is designed and created by Group 57 within the guidance of the course instructor Prof. Mukesh Mohania and the TA Gitansh Raj Satija.



ER DIAGRAM



 link -> [here](#)

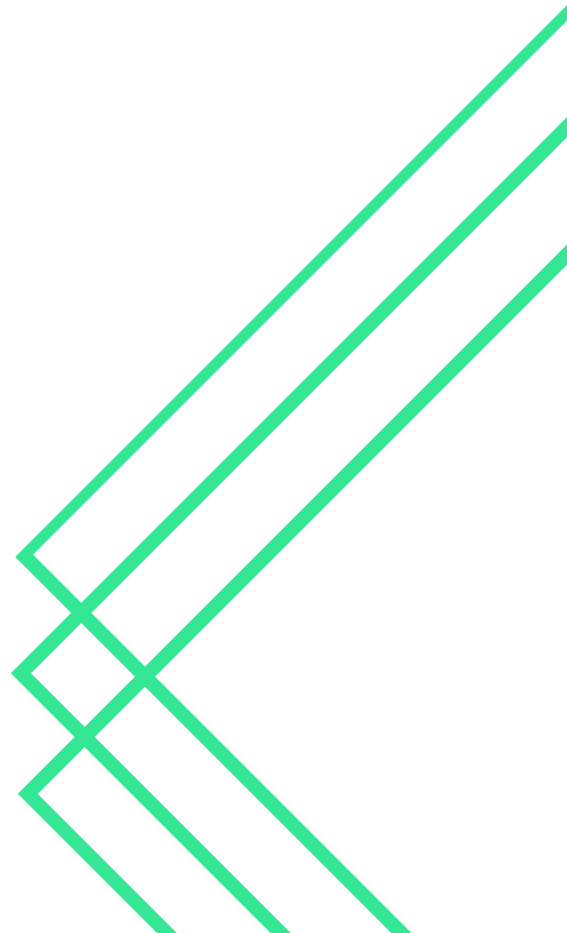
ER
Diagram

PROBLEM STATEMENT

Due to so much indulgence of online shopping in our daily life and because of free delivery at doorstep people tend to order even a single thing, which causes a drastic increase in no of orders due to which almost every street is flooded with delivery bikes and cars, which lead to pollution and traffic jams. Also, sometimes we forget some items while ordering, and we regret later, and then we place an order again, and the cycle continues. Also due to these large no of small orders delivery agents are just moving to and fro again and again and a large part of margin is lost in this travel cost.

STAKE HOLDERS

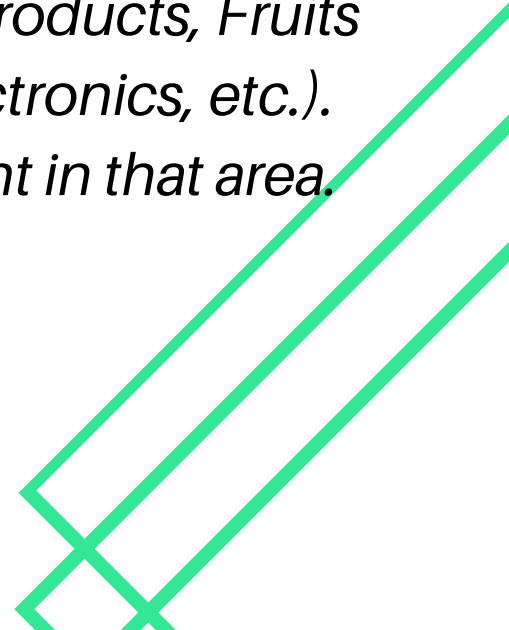
- Customer
- Distributors
- Employees
 - Area Dept/ Shipment Dept
 - Customer Care



SCOPE OF PROJECT

Database Management is the key for the effective and smooth working of Online Store Industry

Nowadays, everything is online, and everyone prefers to keep things in hand, but we already have lots of competition in goods supplies, so we aim to do something different from everyone and which can be profitable to us. Our idea is to deliver the whole goods store to the customer's area, so that not only the customer who ordered can buy but anyone in the area can get what they want at their doorsteps, we can have different carts for different categories of goods(for, e.g., Dairy products, Fruits and Veggies, Beauty Products, Electronics, etc.). We can notify our customers present in that area.

A decorative graphic consisting of several overlapping, parallel green lines that form a series of nested, elongated shapes, resembling a stylized 'X' or a series of parallel paths, located in the bottom right corner of the page.

TABLES!

This section consists of the tables that we have made along with their keys and constraints(if any)

- Product

Primary Key - product_ID , Foreign Key- distributor_ID

```
CREATE TABLE Product (  
  product_ID CHAR(14) PRIMARY KEY NOT NULL,  
  p_name VARCHAR(125) NOT NULL,  
  quantity INT,  
  CP INT NOT NULL,  
  SP INT NOT NULL,  
  discount INT,  
  distributor_ID CHAR(10) NOT NULL,  
  category VARCHAR(125) NOT NULL,  
  p_desc VARCHAR(511),  
  exp_date DATE,  
  rating INT,  
  CHECK(discount < CP),  
  CHECK(rating ≤ 5),  
  FOREIGN KEY (distributor_ID) REFERENCES Distributors(phone)  
);
```

- Distributor

Primary Key - ID or distributor_ID

```
CREATE TABLE Distributors (  
  phone CHAR(10) PRIMARY KEY NOT NULL,  
  d_name VARCHAR(125) NOT NULL,  
  margin INT NOT NULL DEFAULT(100),  
  CHECK(  
    margin > 0  
    AND margin ≤ 100  
  ),  
  FOREIGN KEY (phone) REFERENCES Login(phone)  
);
```

- Customer

Primary Key - ID , Foreign Key - Phone

```
CREATE TABLE Customer (  
  phone CHAR(10) PRIMARY KEY NOT NULL,  
  first_name VARCHAR(63) NOT NULL,  
  last_name VARCHAR(63),  
  full_name VARCHAR(126) AS (concat_ws(' ', first_name, last_name)),  
  line_1 VARCHAR(255) NOT NULL,  
  PIN INT NOT NULL,  
  city VARCHAR(63),  
  addrss VARCHAR(318) AS (concat_ws(' ', line_1, city)),  
  FOREIGN KEY (phone) REFERENCES Login(phone)  
);
```

- Cart

Foreign Key - customer_ID, order_ID

```
CREATE TABLE Cart (  
  customer_ID CHAR(10) NOT NULL,  
  product_ID INT NOT NULL,  
  quantity INT DEFAULT 1,  
  order_ID INT,  
  FOREIGN KEY (customer_ID) REFERENCES Customer(phone),  
  FOREIGN KEY (order_ID) REFERENCES Orders(order_ID)  
);
```

• Orders

Primary Key - order_ID , Foreign Key - Coupon_ID

```
CREATE TABLE Orders (
  order_ID INT PRIMARY KEY AUTO_INCREMENT NOT NULL,
  customer_id CHAR(10) NOT NULL,
  date_of_order DATE NOT NULL,
  price INT NOT NULL,
  order_status VARCHAR(31) NOT NULL,
  expected_delivery DATE NOT NULL,
  rating INT,
  feedback VARCHAR(255),
  coupon_ID VARCHAR(20),
  discount INT DEFAULT 0,
  delivery_charges INT NOT NULL,
  taxes INT NOT NULL,
  total_amount INT NOT NULL DEFAULT (price + delivery_charges + taxes - discount),
  CHECK(
    discount ≥ 0
    AND discount < price
  ),
  CHECK(
    delivery_charges ≥ 0
    AND delivery_charges ≤ 150
  ),
  CHECK(
    rating IS NULL
    OR (
      rating ≥ 0
      AND rating ≤ 5
    )
  ),
  FOREIGN KEY (coupon_ID) REFERENCES Coupons(ID),
  FOREIGN KEY (customer_id) REFERENCES Customer(phone)
);
```

• Employee

Primary Key - ID

```
CREATE TABLE Employee (
  phone CHAR(10) PRIMARY KEY NOT NULL,
  e_name VARCHAR(125) NOT NULL,
  salary INT NOT NULL,
  Joining_date DATE NOT NULL,
  work_from TIME NOT NULL,
  work_to TIME NOT NULL,
  working_hours INT AS (TIMEDIFF(work_to, work_from)),
  gender CHAR(7) NOT NULL,
  department VARCHAR(31) NOT NULL,
  age INT NOT NULL,
  CHECK(
    age ≥ 18
    and age < 60
  ),
  FOREIGN KEY (phone) REFERENCES Login(phone)
);
```


- Login

PK - Phone

```
CREATE TABLE Login (  
  phone CHAR(10) NOT NULL PRIMARY KEY,  
  passwd VARCHAR(15) NOT NULL,  
  acc_type CHAR(1) NOT NULL DEFAULT('C'),  
  CHECK(LENGTH(passwd) ≥ 6)  
);
```

- Coupons

PK - ID

```
CREATE TABLE Coupons (  
  ID VARCHAR(20) PRIMARY KEY NOT NULL,  
  discount_percent INT NOT NULL,  
  min_cart_value INT NOT NULL,  
  max_discount INT NOT NULL,  
  T_C VARCHAR(255),  
  CHECK(  
    discount_percent > 0  
    AND discount_percent < 100  
  ),  
  CHECK(min_cart_value > 0)  
);
```

- Payments

PK - transaction_ID, FK - order_ID

```
CREATE TABLE Payments (  
  transaction_ID INT PRIMARY KEY NOT NULL,  
  order_ID INT NOT NULL,  
  mode VARCHAR(16) NOT NULL,  
  payment_status VARCHAR(16) NOT NULL,  
  amount INT NOT NULL,  
  payment_date DATE NOT NULL,  
  FOREIGN KEY (order_ID) REFERENCES Orders(order_ID)  
);
```

JUSTIFICATION OF WEAK ENTITY/ TERNARY RELATION

Justification of Weak Entity

Since the Cart table doesn't have any attribute which can be used to identify each entry therefore it has to use foreign keys Customer ID and Product ID which can be unique for each entry. To get the items in the cart of customers we can use Customer ID to get a collection of all products in his/her cart. Cart is used to store relations between Customer and Product.

Justification of Ternary Relationship

Place order is a relationship with all the three entities related to it (customer, cart, order) because we'll need to relate customers to order through the items in his/her cart and transfer details from cart to order. This is why we need a ternary relationship.

LIST OF SQL QUERIES

1. Find details of products bought by a customer with given customer_ID

```
select p_name,SP,category,quantity,p_desc,rating
from Product
where product_ID in
(select product_ID from Cart where customer_ID='3810048166');|
```

2. Find 10 product_IDs and their Name with max SP and min CP (more margin)

```
Select product_ID,p_name
from Product
where (product_ID, SP-CP) in
(select product_ID,SP-CP as margin from Product order by margin
desc)
limit 10;
```

3. Find all the orders in which coupons with more than 10% discount are applied

```
select * from Orders
where coupon_ID in
(select ID from Coupons where discount_percent>10);
```

4. Find the price of the most expensive Order placed by every customer.

```
SELECT customer_id, MAX(price) AS p
FROM Orders
GROUP BY customer_id
ORDER BY p desc;
```

5. Find items to return to the certain distributor which are expired

```
select Product.p_name, Product.quantity from Product
where (exp_date < DATE("2022-05-12") and distributor_ID = '8350874341' );
```

6. Orders which are paid through Net Banking and the order value is above some value and only show SUCCESSFUL payments

```
SELECT * FROM orders
JOIN payments ON payments.order_ID = orders.order_ID
WHERE (mode='net-banking' AND payment_status='successful'
AND total_amount > 800);|
```

7. Find the Customers who got free delivery

```
select * from customer
join orders on phone = orders.Customer_ID
where orders.delivery_charges = 0;
```

8. For a min cart value between 800-950 , check available coupons and keep them in ascending order

```
select * from coupons where (min_cart_value between 800 and 900) order by
discount_percent;
```

9. Find the Nth highest priced item in the stock (for eg:- 5)

```
select * from product p1
where 5=(select count(distinct sp) from product p2 where
p2.sp>p1.sp) ;
```

10. Find the favorite products of the people (favorite product = count > 8)

```
select * from cart where product_id in
(select product_id from cart group by product_id having count(*)>8) ;
```

11. Delete any product from the cart

```
Delete from cart where customer_id = '6623243087' and order_id = 327  
product_id = 7775 ;|
```

12. Update address of any entry in the data

```
Update customer  
Set line_1='jahanpanah road', PIN=24, city='porter junction reo'  
Where phone= 0058067108;
```

13. Find details of distributor of N products whose quantity left in stock is min

```
SELECT p_name,d_name,quantity,product.distributor_ID  
from product JOIN distributors ON  
product.distributor_ID=distributors.distributor_ID  
ORDER BY product.quantity LIMIT 5;
```

14. How many customers have a product with a given id(say 5407) in their cart? Give details of the product and its corresponding customer.

```
SELECT  
product.p_name,product.exp_date,customer.full_name,customer.city,customer.  
addrss  
FROM customer,(product JOIN cart ON product.product_ID = Cart.product_ID)  
WHERE product.Product_ID = 5407;
```


15. Find top 'N'(say 5) Most bought products of a customer with given customer_id(say 3810048166) . Also Print the details of those products and customer name

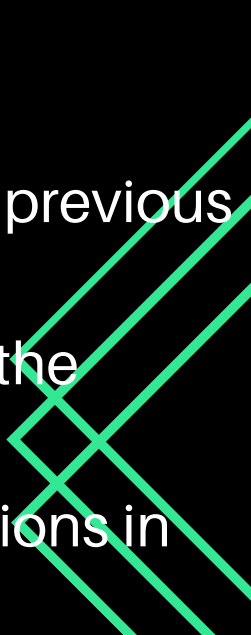
```
select customer.phone,s.p_name,s.p_desc,s.quantity
from customer,(select
product.p_name,product.p_desc,customer_ID,card.quantity from product
JOIN card on card.product_ID=product.product_ID
WHERE card.customer_ID=3810048166| order by card.quantity desc limit
5)
as s where customer.phone=s.customer_ID;
```

16. Print the total_amount of order if there exist any product in the cart of any customer whose quantity is greater than some value(say 6)

```
select total_amount from orders where customer_ID = ANY(select
customer_id from cart where quantity>6);
```



DESCRIPTION OF CHANGES INCORPORATED BASED ON MIDSEM REVIEW

- Replaced customer/employee/distributors id's with phone numbers as PK to reduce redundancy.
 - We added the functionality to find the previous orders placed by customers.
 - Added authentication functionality to the customer, employee, and distributors.
 - Fixed Discount applied, stock up, relations in the ER diagram.
- 

LIST OF DIFFERENT VIEWS CREATED

```
CREATE VIEW Cart_View AS
SELECT customer_id, p_name, Product.product_ID, Cart.quantity
FROM `Product`
INNER JOIN `Cart`
ON (Cart.product_ID = Product.product_ID
    AND Cart.order_ID IS NULL);
```

```
CREATE VIEW Order_View AS
SELECT Orders.customer_ID, Product.product_ID,
    Cart.quantity, SP*Cart.quantity AS price,
    date_of_order, order_status, expected_delivery
FROM `Orders`
INNER JOIN (`Cart`, `Product`)
ON (Orders.order_ID = Cart.order_ID
    AND Cart.product_ID = Product.product_ID
    AND Cart.order_ID IS NOT NULL);
```


LIST OF GRANTS CREATED FOR DIFFERENT ROLES

► Run SQL

```
CREATE ROLE 'employee', 'db_manager';
```

► Run SQL

```
GRANT ALL ON market.* TO 'db_manager';
```

► Run SQL

```
GRANT SELECT, UPDATE, INSERT, DELETE ON market.Coupons TO 'employee';
```

► Run SQL

```
GRANT SELECT ON market.Cart TO 'employee';
```

► Run SQL

```
GRANT SELECT ON market.Customer TO 'employee';
```

► Run SQL

```
GRANT SELECT ON market.Distributors TO 'employee';
```

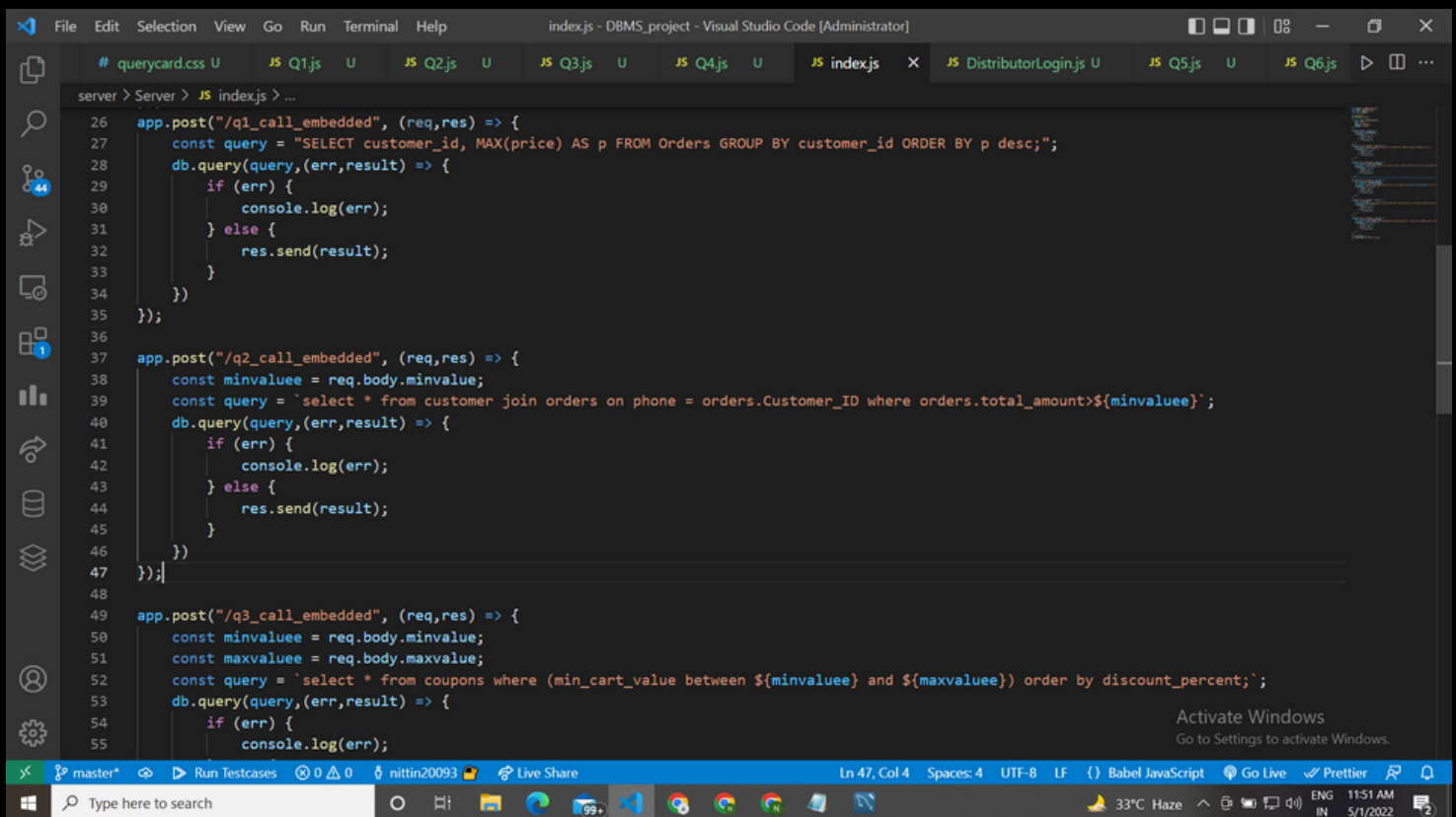
► Run SQL

```
GRANT SELECT ON market.Orders TO 'employee';
```

► Run SQL

```
GRANT SELECT, UPDATE, INSERT, DELETE ON market.Product TO 'employee';
```

LIST OF EMBEDDED SQL QUERIES



The screenshot shows a Visual Studio Code editor window titled "index.js - DBMS_project - Visual Studio Code [Administrator]". The editor displays three embedded SQL queries in JavaScript code, each using the `db.query` method. The queries are as follows:

```
26 app.post("/q1_call_embedded", (req,res) => {
27   const query = "SELECT customer_id, MAX(price) AS p FROM Orders GROUP BY customer_id ORDER BY p desc;";
28   db.query(query,(err,result) => {
29     if (err) {
30       console.log(err);
31     } else {
32       res.send(result);
33     }
34   })
35 });
36
37 app.post("/q2_call_embedded", (req,res) => {
38   const minvaluee = req.body.minvalue;
39   const query = `select * from customer join orders on phone = orders.Customer_ID where orders.total_amount>${minvaluee}`;
40   db.query(query,(err,result) => {
41     if (err) {
42       console.log(err);
43     } else {
44       res.send(result);
45     }
46   })
47 });
48
49 app.post("/q3_call_embedded", (req,res) => {
50   const minvaluee = req.body.minvalue;
51   const maxvaluee = req.body.maxvalue;
52   const query = `select * from coupons where (min_cart_value between ${minvaluee} and ${maxvaluee}) order by discount_percent`;
53   db.query(query,(err,result) => {
54     if (err) {
55       console.log(err);
```

The code is written in JavaScript and uses the `app.post` method to handle HTTP requests. The queries are embedded within the JavaScript code blocks. The first query is a simple `SELECT` statement. The second query is a `JOIN` query with a dynamic value. The third query is a `SELECT` statement with a dynamic range. The editor also shows a terminal window at the bottom with the command `server > Server > JS indexjs > ...`. The status bar at the bottom indicates the file is `index.js`, the language is `Babel JavaScript`, and the encoding is `UTF-8`. The system tray shows the date and time as `5/1/2022 11:51 AM`.

LIST OF EMBEDDED SQL QUERIES

```
server > Server > JS indexjs > ...
62 app.post("/q4_call_embedded", (req,res) => {
63   const N = req.body.n;
64   const query= `select * from product p1 where ${N}=(select count(distinct sp) from product p2 where p2.sp>p1.sp);`
65   db.query(query,(err,result) => {
66     if (err) {
67       console.log(err);
68     } else {
69       res.send(result);
70     }
71   })
72 });
73
74 app.post("/q5_call_embedded", (req,res) => {
75   const minvaluee = req.body.minvalue;
76   const query = `SELECT * FROM orders JOIN payments ON payments.order_ID = orders.order_ID WHERE (mode='net-banking' AND total_amount>${minvaluee}`
77   db.query(query,(err,result) => {
78     if (err) {
79       console.log(err);
80     } else {
81       res.send(result);
82     }
83   })
84 });
85
86 app.listen(3003, () => {
87   console.log('Port binded to 3003');
88 });
```

```
server > Server > JS indexjs > ...
47 });
48
49 app.post("/q3_call_embedded", (req,res) => {
50   const minvaluee = req.body.minvalue;
51   const maxvaluee = req.body.maxvalue;
52   const query = `select * from coupons where (min_cart_value between ${minvaluee} and ${maxvaluee}) order by discount_percent;`;
53   db.query(query,(err,result) => {
54     if (err) {
55       console.log(err);
56     } else {
57       res.send(result);
58     }
59   })
60 });
61
62 app.post("/q4_call_embedded", (req,res) => {
63   const N = req.body.n;
64   const query= `select * from product p1 where ${N}=(select count(distinct sp) from product p2 where p2.sp>p1.sp);`
65   db.query(query,(err,result) => {
66     if (err) {
67       console.log(err);
68     } else {
69       res.send(result);
70     }
71   })
72 });
73
74 app.post("/q5_call_embedded", (req,res) => {
75   const minvaluee = req.body.minvalue;
76   const query = `SELECT * FROM orders JOIN payments ON payments.order_ID = orders.order_ID WHERE (mode='net-banking' AND total_amount>${minvaluee}`
```

STEPS TAKEN FOR QUERY OPTIMIZATION

- We avoid selecting all columns with '*', unless we intend to use them all, selecting redundant columns may result in unnecessary performance degradation.
- Joining columns of different types/lengths in the same condition may cause performance degradation. The database will have to perform a cast for each of these values before performing the comparison, which may also prevent the use of an index for these columns. Make sure to alter the column types so that common comparisons will be done between two columns of the same type. (collapse)

- **Avoid Implicit Casts When Searching for Strings-**

When the database is required to cast values, an index can't be used to enhance performance. It's recommended not to compare string columns to numeric values, as it will result in a cast that will prevent index usage.

- Indexing tables with long entries so that while running the query it will give result fast and will use less computation.

-

LIST OF ALL INDICES CREATED

```
create index expdate on product(exp_date);
```

```
create index disc on product(discount,category);
```

```
create index mincart on coupons(min_cart_value);
```

```
create index sp on product(sp);
```

```
create index discper on coupons(discount_percent);
```

	Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type	Comment	Index_comment	Visible	Expression
▶	product	0	PRIMARY	1	product_ID	A	1141	<div>NULL</div>	<div>NULL</div>		BTREE			YES	<div>NULL</div>
	product	1	distributor_ID	1	distributor_ID	A	145	<div>NULL</div>	<div>NULL</div>		BTREE			YES	<div>NULL</div>
	product	1	disc	1	discount	A	579	<div>NULL</div>	<div>NULL</div>	YES	BTREE			YES	<div>NULL</div>
	product	1	disc	2	category	A	811	<div>NULL</div>	<div>NULL</div>		BTREE			YES	<div>NULL</div>
	product	1	expdate	1	exp_date	A	672	<div>NULL</div>	<div>NULL</div>	YES	BTREE			YES	<div>NULL</div>
	product	1	sp	1	SP	A	114	<div>NULL</div>	<div>NULL</div>		BTREE			YES	<div>NULL</div>

	Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type	Comment	Index_comment	Visible	Expression
▶	coupons	0	PRIMARY	1	ID	A	0	<div>NULL</div>	<div>NULL</div>		BTREE			YES	<div>NULL</div>
	coupons	1	mincart	1	min_cart_value	A	114	<div>NULL</div>	<div>NULL</div>		BTREE			YES	<div>NULL</div>
	coupons	1	discper	1	discount_percent	A	16	<div>NULL</div>	<div>NULL</div>		BTREE			YES	<div>NULL</div>

TRIGGERS

```
delimiter |
CREATE TRIGGER upd before update ON product
FOR EACH ROW
BEGIN
    IF new.quantity <= 0 THEN
        SET NEW.quantity = new.quantity + 40;
    END IF;
END;
|
delimiter ;
```

Description of the trigger:-

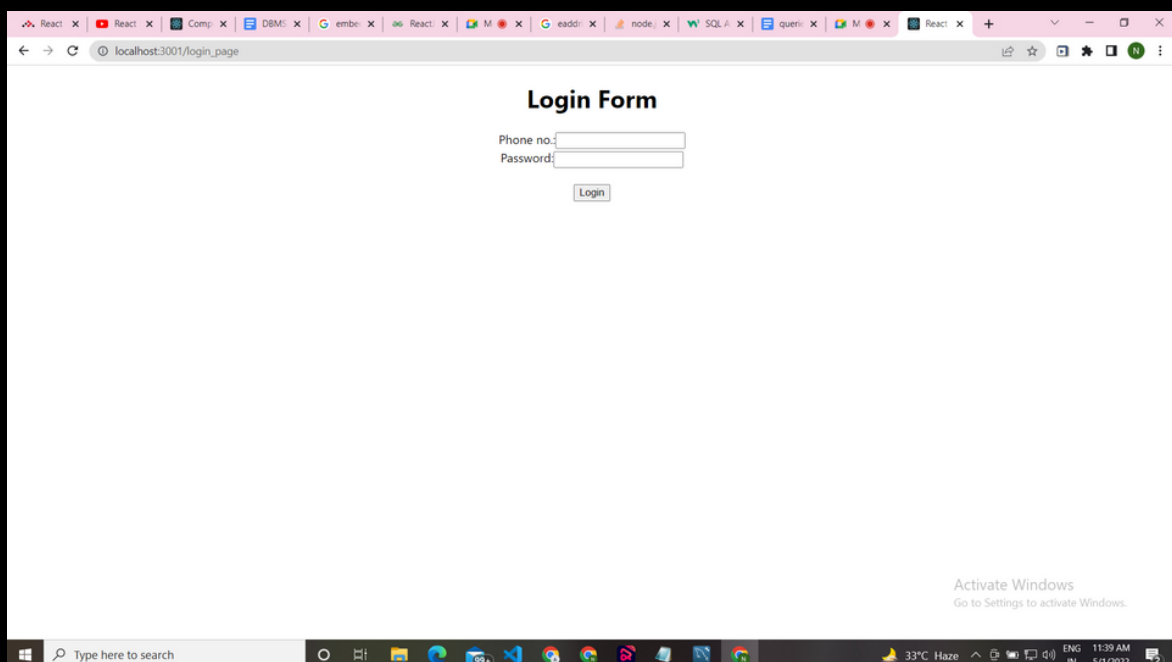
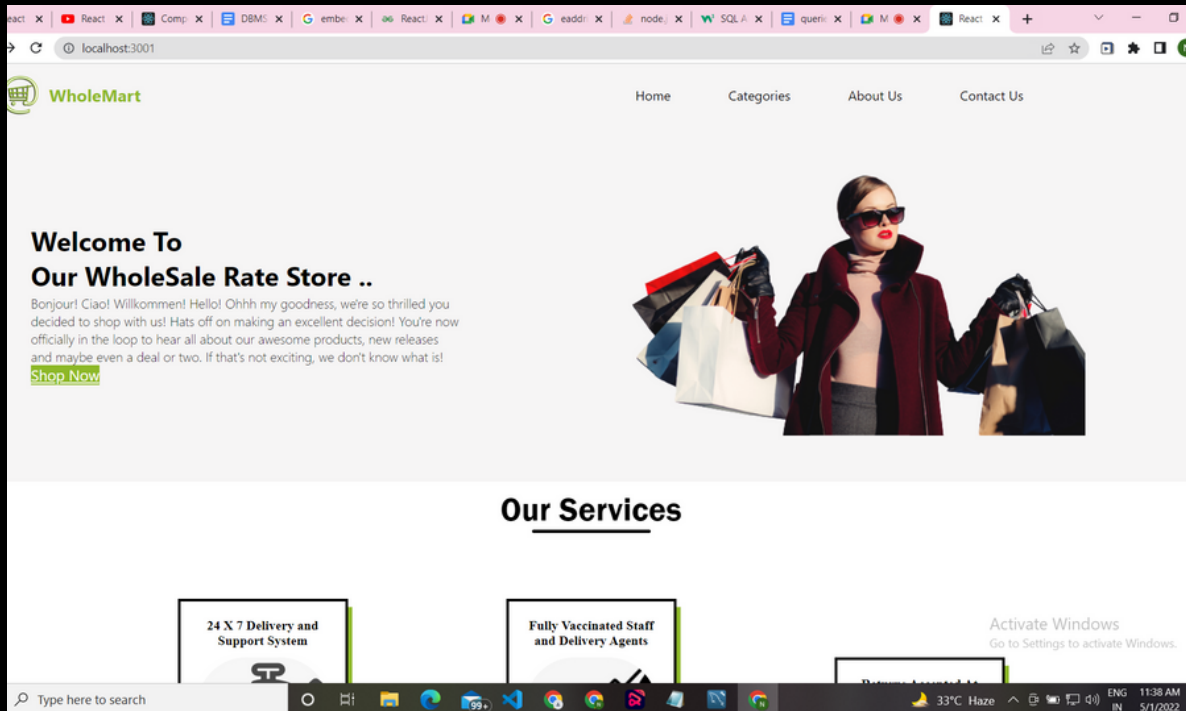
This trigger works as whenever the quantity of any item in the product table updated to 0 or less than 0 , it will fire itself to make that item quantity equal to 40 in the stock.

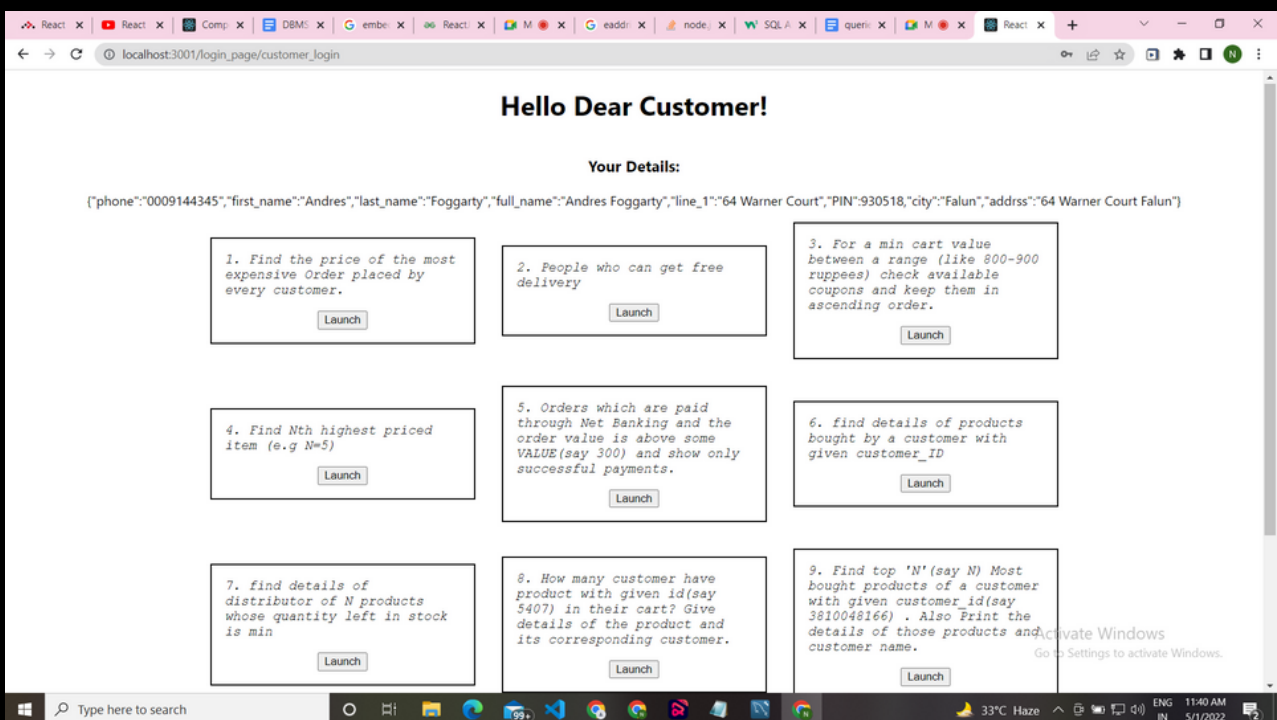
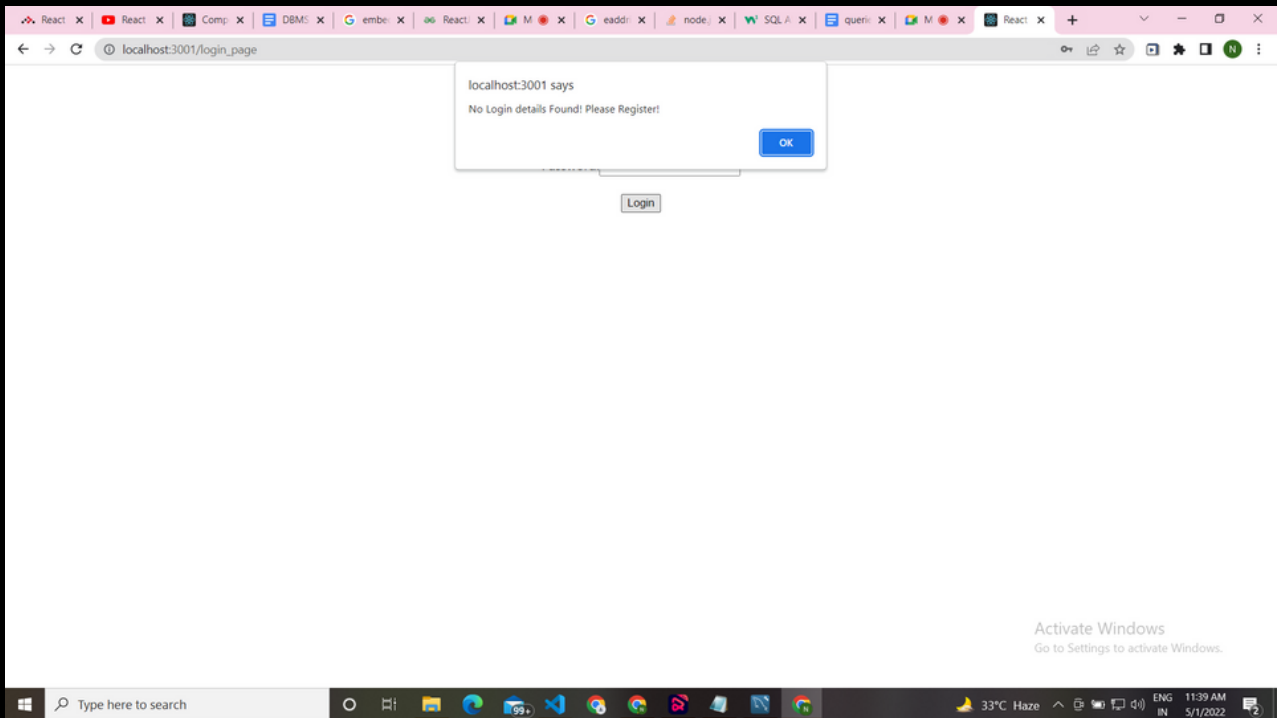
```
delimiter |
create trigger delivery before insert on orders
for each row
begin
    if new.price > 750 then
        set new.delivery_charges = 0 ;
    end if ;
end ;
|
delimiter
```

Description of the trigger:-

This trigger works as whenever customer's item price is more than 750 , they don't have to give any delivery charges for their order.

SCREENSHOTS OF USER INTERFACE (UI)





React x React x Comp x DBM x embe x React x M x eaddr x node x SQL x queri x M x React x

localhost:3001/login_page/customer_login/q9

Welcome to query Handler!

Find top 'N'(say N) Most bought products of a customer with given customer_id(say 3810048166) . Also Print the details of those products and customer name.

SQL QUERY : select customer.full_name,s.p_name,s.p_desc,s.quantity from customer,(select product.p_name,product.p_desc,customer_ID,cart.quantity from product JOIN cart on cart.product_ID=product.product_ID WHERE cart.customer_ID=3810048166 order by cart.quantity desc limit 5)as s where customer.phone=s.customer_ID;

Enter VALUE of N !! (BY Default VALUE=5) :5
Enter VALUE of customer_ID !! (BY Default VALUE=3810048166) :3810048166
Launch

full_name	product_name	product_desc	quantity
Christophorus Bourdice	USS-6XLPantalon	lorem ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum	12
Christophorus Bourdice	Chapeaude	lorem ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum	12
Christophorus Bourdice	Ensemblede	lorem ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum	8
Christophorus Bourdice	ChaussettesathlÃ©tiquesrespirantesen	lorem ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum	5
Christophorus Bourdice	Casquettesdebaseball	lorem ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum lorem ipsum	5

Activate Windows
Go to Settings to activate Windows.

Type here to search

33°C Haze 11:40 AM 5/1/2022

React x React x Comp x DBM x embe x React x Goog x eaddr x node x SQL x queri x M x React x

localhost:3001/login_page/employee_login

Hello Employee!

Your Details:

["phone":"0016965991","e_name":"Jacques","salary":30498,"Joining_date":"2020-06-23T18:30:00.000Z","work_from":"08:00:00","work_to":"14:00:00","working_hours":60000,"gender":"Female","department":"Advertisement","age":23]

Activate Windows
Go to Settings to activate Windows.

Type here to search

33°C Haze 11:43 AM 5/1/2022

MEMBER CONTRIBUTION

This project is the outcome of the hard work done equally by the members of Group 57. However , below are the tasks that was assigned to Group 57 members specifically.

By Ankit Chaurasia :-

- Data population
- Grants
- Views
- Updation from midsem review

By Aayush Kumar :-

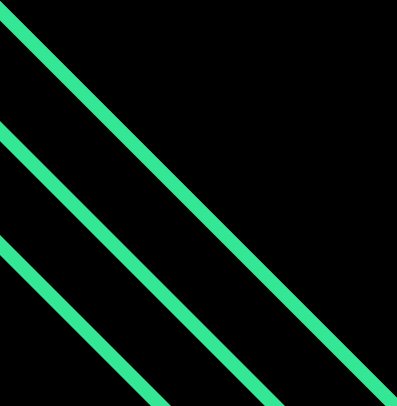
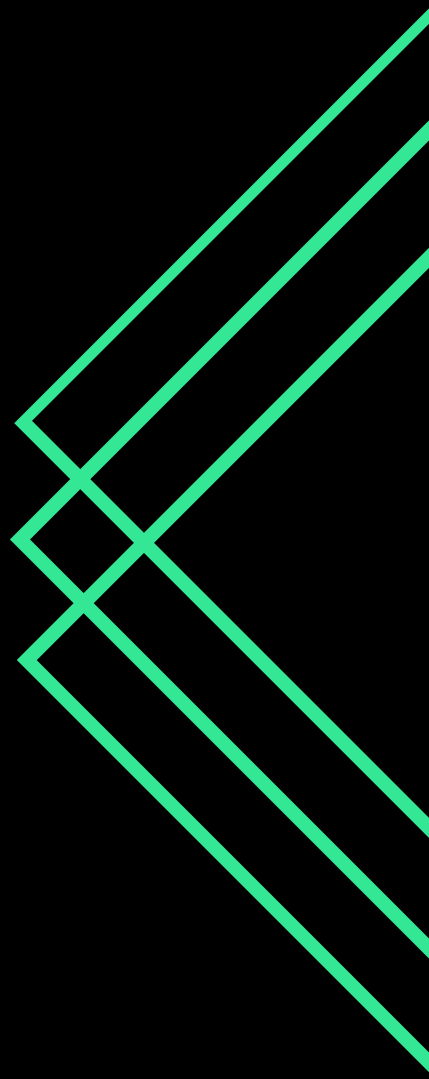
- Triggers
- Indexing
- Report Making and Document Updation
- Refactoring population scripts



By Nittin Yadav :-

- User Interface (Front End)
- Back End
- Connection of Back End with Front End
- Embedded SQL Queries

By Dishant Yadav :-

- Scope of Project
 - SQL Queries
 - Query Optimization
 - Stake Holders and Problem Statement
- 
- 

LINK OF GITHUB REPOSITORY

just_for_you



THANK YOU