Online Shopping System (EasyCart)

Grp No. 17

Pratik Pathak (2017255)
Rishav(2017182)
Gajal(2017150)
Suhail(2017201)
Ramachandran(2017084)

Application: An online shopping system portal

We are aiming to build the database system of a basic e-commerce product selling website.

Week 1

Stakeholders: System Admin, Customers, Sellers, Reviewers(Management)

The work of these stakeholders will be as follows:

- **System Admin**
  The primary work includes:
  1. Management of the database.
  2. Addition/Removal of products on the basis of reviews.
  3. Removal of inactive users and monitoring sales.
- **Customer**
  The primary work includes:
  1. Creating and updating their profile
  2. Buying products from the portal
  3. Reviewing their buy

- **Seller**
  The primary work includes:
  1. Creating and updating their profile
  2. Adding products on the portal
  3. Managing the prices for products and checking total products sold.

- **Reviewer**
  The primary work includes:
  1. Adding customer/ product review for the portal.
  2. Evaluating the items on the basis of reviews

Week 2

The questions pertaining to each of the stakeholders are:

- System Admin
  1. Checking the validity of the users who have signed in to the system by checking their information.
  2. What are the primary types of products provided by a new seller?
  3. Are there any products or sellers who have received a very low review?
  4. How many users have not purchased an item for a long time and how many products haven’t fetched a buyer for long?
  5. Check if the quantity of any product is zero

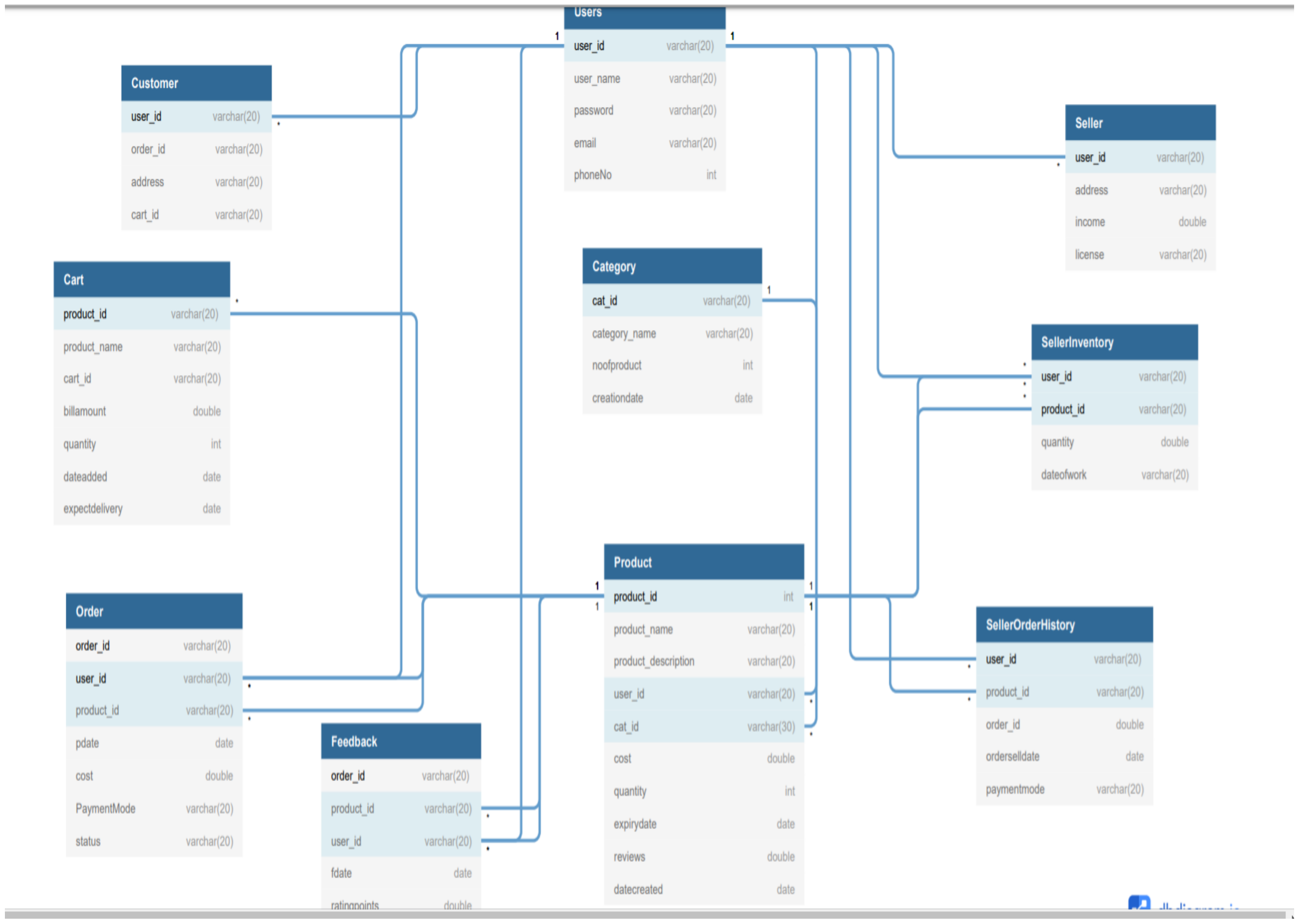
- Customer
  1. Given a product name, what all such products are available?
  2. Finding the details of a particular product
  3. If a product is selected, adding it to the cart so that there is no need to search it again
  4. Buying the products from the cart and then giving a review on the product
  5. Updating the details for the customer.

- Seller
  1. Creating a seller account on the portal.
  2. Adding products for the sellar on the portal.
  3. Checking the sales and feedback for any product by the quantities sold and the review received.
  4. Updating the prices and description for a particular product.
  5. Check whether the discount for a product can be given.

- Reviewer
  1. Check if the review score is above or less than 5 and label the review as ‘good’ or ‘bad’ accordingly.
  2. Add/Update the review score to the product.
  3. Add the review score of a product by customers on the portal.
  4. Check if the review score is not given for any product.

5. Check which products have the highest rating.

Schema Diagram



Week 3

The tables are as follows:

1.Product

Name	Data Type	Constraint
Prod_ID	Varchar(20)	Not null, primary key
Prod_Name	Char(20)	Not null
Description	Varchar(20)	
User_ID	Varchar(20)	Not null, foreign key
Category	Varchar(30)	Not null, foreign key
Cost	double	
Quantity	integer	
Rating	float	
Total Reviews	date	

2.User

Name	Data Type	Constraint
User_ID	Varchar(20)	Not null, primary key
User_Name	Varchar(20)	Not null
Password	Varchar(20)	Not null
Email	Varchar(20)	
Phone No.	Integer	

3. Seller

Name	Data Type	Constraint
Seller_ID	Varchar(20)	Not null, primary key

Seller_Name	Varchar(20)	Not null
Password	Varchar(20)	Not null
Email	Varchar(20)	
Phone No.	Integer	

4.Seller Inventory

Name	Data Type	Constraint
User_ID	Varchar(20)	Not null, foreign key
Prod_ID	Varchar(20)	Not null, primary key
Quantity	int	
Updated date	Varchar(20)	

5. Customer

Name	Data Type	Constraint
Customer_ID	Varchar(20)	Not null, primary key
Customer_Name	Varchar(20)	Not null
Password	Varchar(20)	Not null
Email	Varchar(20)	
Phone No.	Integer	

6.Cart

Name	Data Type	Constraint
Cart_ID	Varchar(20)	Not null, primary key
Prod_Name	Varchar(20)	
Prod_ID	Varchar(20)	
Customer_ID	Varchar(20)	
Bill amount	double	
Date Added	date	
Expected Delivery	date	

7.Orders

Name	Data Type	Constraint
Order_id	Varchar(20)	Not null, primary key
Prod_ID	Varchar(20)	Not null
User_ID	Varchar(20)	Not null, foreign key
Purchased Date	date	Not null
Cost	real	
Payment Mode	Varchar(20)	
Status	Varchar(20)	

8.Feedback Table

Name	Data Type	Constraint
Order_ID	Varchar(20)	Not null, primary key
Prod_ID	Varchar(20)	Not null, foreign key
User_ID	Varchar(20)	Not null, foreign key
Feedback Date	date	
RatingPoints	double	

9.Category Table

Name	Data Type	Constraint
Cat_ID	Varchar(20)	Not null, primary key
Category_Name	Varchar(20)	Not null

10.Seller Order History

Name	Data Type	Constraint
order_id	Varchar(20)	Primary key
User_ID	Varchar(20)	Not null, foreign key
Prod_ID	Varchar(20)	Not null, primary key
OrderSell Date	date	
Payment Mode	Varchar(20)	

Week 4

DATABASE and TABLE CREATION

```
create database EasyCart;
use EasyCart;

CREATE TABLE `Users` (
  `user_id` varchar(20) PRIMARY KEY,
  `user_name` varchar(20),
  `passcode` varchar(20),
  `email` varchar(20) UNIQUE,
  `phoneNo` varchar(20)
);


CREATE TABLE `Seller` (
  `Seller_id` varchar(20) PRIMARY KEY,
  `user_name` varchar(20),
  `passcode` varchar(20),
  `email` varchar(20) UNIQUE,
  `phoneNo` varchar(20)
);

CREATE TABLE `Product` (
  `product_id` varchar(20) PRIMARY KEY,
  `product_name` varchar(50),
  `product_description` varchar(100),
  `Seller_id` varchar(20),
  `Category` varchar(30),
  `cost` double,
  `quantity` int
);


CREATE TABLE `SellerInventory` (
  `user_id` varchar(20),
  `product_id` varchar(20),
  `quantity` double,
  `last updated` varchar(20),
  PRIMARY KEY (`user_id`, `product_id`)
);

CREATE TABLE `Customer` (
  `Customer_id` varchar(20) PRIMARY KEY,
  `user_name` varchar(20),
  `passcode` varchar(20),
  `email` varchar(20),
  `phoneNo` varchar(20)
);

CREATE TABLE `Cart` (
  `cart_id` varchar(20) PRIMARY KEY,
  `product_name` varchar(50),
  `product_id` varchar(20),
  `customer_id` varchar(20),
  `billamount` double,
  `dateadded` varchar(20),
  `expectdelivery` varchar(20)
);

CREATE TABLE `Orders` (
  `order_id` varchar(20) PRIMARY KEY,
  `user_id` varchar(20),
  `product_id` varchar(20),
  `order_date` date,
  `cost` double,
  `PaymentMode` varchar(20),
  `status` varchar(20)
);

CREATE TABLE `Feedback` (
  `order_id` varchar(20) PRIMARY KEY,
  `product_id` varchar(20),
  `user_id` varchar(20),
  `fdate` date,
  `ratingpoints` double
);


CREATE TABLE `Category` (
  `cat_id` varchar(20) PRIMARY KEY,
  `category_name` varchar(50)
);
```

```
CREATE TABLE `SellerOrderHistory` (
  `user_id` varchar(20),
  `product_id` varchar(20),
  `order_id` varchar(20) PRIMARY KEY,
  `ordersell date` date,
  `paymentmode` varchar(20)
);



ALTER TABLE `Product` ADD FOREIGN KEY (`user_id`) REFERENCES `Users` (`user_id`);

ALTER TABLE `Seller` ADD FOREIGN KEY (`user_id`) REFERENCES `Users` (`user_id`);

ALTER TABLE `Customer` ADD FOREIGN KEY (`user_id`) REFERENCES `Users` (`user_id`);

ALTER TABLE `SellerInventory` ADD FOREIGN KEY (`user_id`) REFERENCES `Users` (`user_id`);

ALTER TABLE `Orders` ADD FOREIGN KEY (`user_id`) REFERENCES `Users` (`user_id`);

ALTER TABLE `Feedback` ADD FOREIGN KEY (`user_id`) REFERENCES `Users` (`user_id`);

ALTER TABLE `SellerOrderHistory` ADD FOREIGN KEY (`user_id`) REFERENCES `Users`
(`user_id`);

ALTER TABLE `Product` ADD FOREIGN KEY (`cat_id`) REFERENCES `Category` (`cat_id`);
```

DATA INSERTION

```
INSERT INTO `Category` VALUES(1,"Automative and PowerSports");
INSERT INTO `Category` VALUES(2,"Baby Products");
INSERT INTO `Category` VALUES(3,"Beauty");
INSERT INTO `Category` VALUES(4,"Books");
INSERT INTO `Category` VALUES(5,"Camera and Photo");
INSERT INTO `Category` VALUES(6,"Consumer Electronics");
INSERT INTO `Category` VALUES(7,"Entertainment");
INSERT INTO `Category` VALUES(8,"Fashion and Clothing");
INSERT INTO `Category` VALUES(9,"Fine Art");
INSERT INTO `Category` VALUES(10,"Grocery and Food");
INSERT INTO `Category` VALUES(11,"Health and Personal Care");
INSERT INTO `Category` VALUES(12,"Industrial and Scientific");
INSERT INTO `Category` VALUES(13,"Music");
INSERT INTO `Category` VALUES(14,"Office Products");
INSERT INTO `Category` VALUES(15,"Outdoors");
INSERT INTO `Category` VALUES(16,"Software");
INSERT INTO `Category` VALUES(17,"Shoes");
INSERT INTO `Category` VALUES(18,"Sports");
INSERT INTO `Category` VALUES(19,"Others");


insert into Users values ( 'S1','Raman','123a','raman@mahoo.com','9711912198' ),
( 'C2','Shyam','123a','shyam@mahoo.com','9711912199' ),
( 'S3','Piyush','123b','piyush@mahoo.com','9711912188' ),
( 'C4','Mohit','123c','mohit@mahoo.com','9711002198' ),
( 'S5','Mayank','123d','mayank@mahoo.com','7811912198' ),
( 'C6','Madhav','123e','madhav@mahoo.com','9811912198' ),
( 'S7','Gujjar','123f','gujjar@mahoo.com','9711912177' ),
( 'C8','Nidhi','123g','nidhi@mahoo.com','9711912777' ),
( 'S9','Seema','123h','seema@mahoo.com','9751912198' ),
( 'C10','Sakshi','123i','sakshi@mahoo.com','9711852198' ),
( 'S11','Priyanshi','123j','priyanshi@mahoo.com','9711926198' ),
( 'C12','Atul','123k','atul@mahoo.com','9711912193' ),
( 'S13','Dhaman','123l','dhaman@mahoo.com','9711412198' ),
( 'C14','Rathore','123m','rathore@mahoo.com','9481912198' ),
( 'S15','Jay','123n','jay@mahoo.com','9711912218' ),
( 'C16','Michael','123o','michael@mahoo.com','9321912198' ),
( 'S17','Jack','123p','jack@mahoo.com','9711912116' ),
( 'C18','Jem','123q','jem@mahoo.com','9711912798' ),
( 'S19','William','123r','william@mahoo.com','9711912198' ),
( 'C20','Shona','123s','shona@mahoo.com','9451912198' );

insert into Seller values ( 'S1','Ghaziabad',35000,'456abc' ),
( 'S3','Jabalpur',45000,'457dbc' ),
( 'S5','Delhi',75000,'458aac' ),
( 'S7','Dhanbad',15000,'556abc' ),
( 'S9','Patna',45000,'457asc' ),
( 'S11','Loni',46700,'426wbc' ),
( 'S13','Arthala',65000,'456acc' ),
( 'S15','Noida',35800,'453abb' ),
( 'S17','Dholakpur',31000,'356dbc' ),
( 'S19','YamunaVihar',33000,'476aec' );

insert into Customer values ( 'C2', 'Ajanta','R2'),
( 'C4','LajpatNagar','R4' ),
( 'C6','Delhi','R6' ),
( 'C8','Johrat','R8' ),
( 'C10','MayurVihar','R10' ),
( 'C12','Sahibabad','R12' ),
( 'C14','LaxmiNagar','R14' ),
( 'C16','VivekVihar','R16' ),
( 'C18','Dilshad','R18' ),
( 'C20','Rohini','R20' );

insert into Category values ( 'A1','Electronics',3,'2010-02-01'),
('A2','Books',4,'2011-02-10'),
('A3','Movies',5,'2010-06-15'),
('A4','Clothing',5,'2011-03-01'),
('A5','Home decoration',3,'2010-12-08'),
('A6','Jewellery',4,'2010-01-09');

insert into Product values ( 'P1','Nokia 8','Charcoal 64GB 6GB RAM','S5','A1',15000,5,'2030-03-
01',4.3,'2010-05-03'),
('P3','Skull Candy Riff','Gray Built-In Microphone','S3','A1',1000,10,'2035-10-12',6.2,'2012-01-02'),
('P5','ASUS F570','Black AMD Quad Core 8GB RAM','S1','A1',25000,25,'2040-10-11',8.6,'2013-03-03'),
('P7','A Man Search for Meaning','Author Viktor Frankl','S9','A2',1000,20,'2039-09-01',7.8,'2015-04-01'),
('P9','A Search in Secret India','Author Paul Brunton','S3','A2',1500,15,'2035-10-02',8.2,'2019-10-19'),
('P11','The Light of Asia','Author Edwin Arnold','S13','A2',500,20,'2032-05-09',7.3,'2018-04-01'),
('P13','The Searchers','Director John Ford','S15','A3',750,20,'2048-03-09',6.4,'2018-11-14'),
('P15','Raging Bull','Director Martin Scorsese','S19','A3',800,15,'2050-01-10',9.5,'2018-10-25'),
('P17','The Deer Hunter','Director Michael Cimino','S17','A3',900,40,'2038-08-15',8.6,'2017-07-23'),
('P19','Citizen Kane','Director Orson Welles','S15','A3',500,30,'2049-02-03',7.7,'2014-10-28'),
('P21','Ran','Director Akira Kurusowa','S1','A3',600,30,'2035-04-18',6.8,'2010-09-15'),
('P23','GRITSTONES','Unisex Cotton Tie Dye Full Sleeves Shirt ','S3','A4',750,20,'2035-09-12',5.9,'2012-
12-23'),
('P25','Dennis Lingo','Men Stretchable Green Button Down Slim Fit Casual Shirt','S5','A4',700,20,'2031-
07-10',4.1,'2015-10-18'),
('P27','Ben Martin','Men Denim Regular Fit Jeans Blue','S7','A4',850,25,'2030-06-06',3.2,'2016-03-10'),
('P29','KRAVE','Women Striped Green Regular Fit Shirt','S13','A4',900,10,'2031-05-14',5.3,'2017-02-11'),
('P31','Elle','Women Slim Pants Red','S11','A4',800,15,'2030-10-10',6.4,'2013-09-13'),
('P33','RM Fengshui Marble Turtle','Decoration Gifting','S9','A4',1500,10,'2031-11-11',6.7,'2020-10-14'),
('P35','Decals Design Flowers Branch','Wall Sticker','S7','A4',2000,10,'2032-12-12',7.4,'2019-09-09'),
('P37','Furniture Cafe Zigzag Corner Wall Mount Shelf','Wall Decoration','S5','A4',1600,15,'2033-01-
01',8.3,'2018-10-10'),
('P39','The Idiot','Author Fyodor Dostoevsky','S3','A2',1000,20,'2034-02-02',7.2,'2017-04-05'),
('P41','Youbella','Bohemian Multicolor Metal Earrings For Women','S17','A5',12000,10,'2035-03-
03',8.1,'2010-03-02'),
('P43','Shining Diva Fashion','Oxidized Silver Ring for Women','S11','A5',25000,5,'2036-04-
04',4.2,'2011-02-02'),
('P45','Sukkhi','Gold Plated Jewellery Set for Women','S1','A5',22000,10,'2037-05-05',5.4,'2014-08-12'),
('P47','YouBella Jewellery Set','Silver Crystal Rhinestone Choker Necklace for
Women','S3','A5',25000,10,'2038-06-06',6.5,'2019-05-06');

insert into Cart values ('P1','Nokia 8','R2',45000,3,'2020-01-01','2020-01-08'),
('P3','Skull Candy Riff','R4',4000,4,'2020-01-10','2020-01-17'),
('P15','Raging Bull','R14',4800,6,'2020-02-11','2020-02-18'),
('P9','A Search in Secret India','R6',15000,10,'2019-03-12','2019-03-19'),
('P7','A Man Search for Meaning','R10',12000,12,'2020-04-13','2020-04-21'),
-- ('P13','The Searchers','R14',6000,8,'2020-01-15','2020-01-20'),
('P11','The Light of Asia','R8',2500,5,'2020-02-16','2020-02-21'),
('P19','Citizen Kane','R18',5000,10,'2020-03-17','2020-03-24'),
('P21','Ran','R20',1200,2,'2020-02-18','2020-02-25'),
('P33','RM Fengshui Marble Turtle','R12',15000,10,'2020-03-21','2020-03-28'),
('P5','ASUS F570','R16',50000,2,'2020-03-22','2020-03-29');

insert into Orders values ('O19','C20','P1','2020-01-01',45000,'COD','DELIVERED'),
('O28','C20','P9','2019-03-12',15000,'PAYTM','PENDING'),
('O83','C12','P11','2020-02-16',2500,'COD','DELIVERED'),
('O13','C14','P13','2020-01-15',6000,'NET BANKING','PROCESSING'),
('O3','C4','P3','2020-01-10',4000,'NET BANKING','DELIVERED'),
('O17','C18','P7','2020-04-13',12000,'PAYTM','PROCESSING'),
('O5','C6','P33','2020-03-21',15000,'COD','PENDING'),
('O7','C8','P21','2020-02-18',1200,'PAYTM','SHIPPED'),
('O45','C4','P5','2020-03-22',50000,'PAYTM','DELIVERED'),
('O15','C16','P3','2020-01-14',5000,'COD','PENDING'),
('O1','C2','P15','2020-02-11',4800,'NET BANKING','PENDING'),
('O9','C10','P19','2020-03-17',5000,'PAYTM','DELIVERED'),
('O11','C12','P7','2020-04-30',1000,'COD','PENDING');

insert into SellerInventory values ('S1','P1',4,'2020-01-03'),
('S3','P1',4,'2020-01-18'),
('S5','P13',14,'2019-09-13'),
('S7','P15',3,'2020-02-03'),
('S3','P7',2,'2019-09-20'),
('S7','P19',14,'2020-02-01'),
('S5','P33',42,'2020-01-17'),
('S17','P7',43,'2020-01-27'),
('S19','P33',54,'2019-10-21'),
('S15','P1',74,'2019-07-13'),
('S1','P3',40,'2020-01-09');

insert into feedback values ('O19','P1','C20','2020-01-01',4.5),
('O28','P9','C20','2019-03-12',2.9),
('O83','P11','C12','2020-02-16',4.7),
('O13','P13','C14','2020-01-15',2.9),
('O3','P3','C4','2020-01-10',3.6),
('O17','P7','C18','2020-04-13',4.7),
('O5','P33','C6','2020-03-21',1.5),
('O7','P21','C8','2020-02-18',1.2),
('O45','P5','C4','2020-03-22',4.0),
('O15','P3','C16','2020-01-14',3.5),
('O1','P15','C2','2020-02-11',3.8),
('O9','P19','C10','2020-03-17',4.1),
('O11','P7','C12','2020-04-30',3.7);

insert into sellerorderhistory values ('S1','P1','O19','2020-01-03','PAYTM'),
('S3','P9','O28','2020-01-18','NETBANKING'),
('S5','P11','O83','2019-09-13','COD'),
('S7','P13','O13','2020-02-03','PAYTM'),
('S3','P3','O3','2019-09-20','UPI'),
('S7','P7','O17','2020-02-01','COD'),
('S5','P33','O5','2020-01-17','PAYTM'),
('S17','P8','O21','2020-01-27','COD'),
('S19','P5','O45','2019-10-21','NETBANKING'),
('S15','P2','O1','2019-07-13','COD'),
('S13','P3','O15','2020-01-09','NETBANKING');
```

Week 6

-Identifying the common queries shared among different stakeholders

The final list after the discussions were:

-Syntax for indexing

```
CREATE [UNIQUE|FULLTEXT|SPATIAL] INDEX index_name
USING [BTREE | HASH | RTREE]
 ON table_name (column_name [(length)] [ASC | DESC],…)
```

- order_id and status are the attributes chosen from the Order Table so that the Customer can quickly check the status of delivery

-For example, for tracking the order one must have quick access to order table so creating a unique Btree index on order table can be useful

```
CREATE UNIQUE INDEX idx_orderid
USING BTREE
ON orders(order_id);
```

-For the product table, quick access is also required so that the customers do not need to wait
-prod_id is the attribute chosen from the Product Table so that the Customer and Seller can retrieve data such as price and product name about the product and the System Admin can add/delete products based on reviews quickly from the database

```
CREATE INDEX idx_proid
USING HASH
ON product(product_id);

CREATE INDEX idx_productname
ON product(product_name);
```

-Hash indexing is used in the above example

-As it is obvious most of the query will revolve around the user id and username so the below indexing is implemented in the user table

```
CREATE UNIQUE INDEX idx_userid
ON Users(user_id);

CREATE INDEX idx_username
ON Users(user_name);
```

- Identifying the various SQL query revolving around each of the stakeholders involved primarily including

The following are the queries involving various relational algebraic operations supporting the application features that involve database access and manipulation:

Projection and Selection

Find the customerid of the customer whose email is abc@gmail.com and passcode is abc

1 - SELECT Customer_id from Customer WHERE Email = 'abc@gmail.com' AND passcode = 'abc'

REL ALG

$$\pi_{\text{customer\_id}} (\sigma_{\text{email='abc@gmail.com'} \wedge \text{passcode='abc'}} (\text{Customer}))$$

Union, Projection and Selection

Find the reviews where the cost is either 2000 or 5000

2 - (SELECT Reviews from Product where Cost=2000) union
 (SELECT Reviews from Product where Cost=5000)

REL ALG

$$\pi_{\text{Product\_id}} (\sigma_{\text{Cost=2000}}(\text{Product})) \cup \pi_{\text{Product\_id}} (\sigma_{\text{Cost=5000}} (\text{Product}))$$

Set Difference, Projection and Selection

Find the categoryid of those categories whose product quantity is lesser than 50 but not lesser than 0

3 - (SELECT Cat_ID from Category where Product_quantity<50) except
 (SELECT Cat_ID from Category where Product_quantity<0)

REL ALG

$$\pi_{\text{Cat\_ID}} (\sigma_{\text{Product\_quantity<50}}(\text{Category})) - \pi_{\text{Cat\_ID}} (\sigma_{\text{Product\_quantity<0}} (\text{Category}))$$

Cartesian Product, Rename, Selection and Projection

Find the maximum review of a product

4 - SELECT Prod_ID, MAX(review) as rev from Product

REL ALG

$$\pi_{\text{review}} (\text{Product}) - \pi_{\text{Product.review}} (\sigma_{\text{Product.review < d.review}} (\text{Product} \times \rho_{d} (\text{review})))$$

Projection

Find all the content from users

5 - SELECT * from users

REL ALG

π (users)

Full Outer Join

Select all details of customers and orders together irrespective of whether the customer has placed an order

6 - SELECT * FROM Customer FULL OUTER JOIN Order ON Customer.User_ID=Order.User_ID

REL ALG

$\text{Customer} ⟗_{\text{Customer.User\_ID=Order.User\_ID}} \text{Order}$

Intersection

Find productids where both the review is 8 as well as the cost is 500

7 - (SELECT Prod_ID from Product where Reviews=8) intersect
    (SELECT Prod_ID from Product where Cost=500)

REL ALG

$\pi_{\text{Product\_id}} (\sigma_{\text{Reviews=8}} (\text{Product})) \cap \pi_{\text{Product\_id}} (\sigma_{\text{Cost=500}}(\text{Product}))$

Selection, Projection and Cartesian Product

Find the userids of customer and seller whose address is the same

8 - SELECT Customer. User_ID, Seller. User_ID from Customer, Seller where Customer. Address=Seller.Address

REL ALG

$\pi_{\text{Customer. User\_ID, Seller. User\_ID}} (\sigma_{\text{Customer. Address=Seller.Address}} (\text{Customer x Seller})$

Projection and Selection

Find all the content from Product where the sellerid is S012345

9 - SELECT * FROM Product WHERE seller_id =’S012345’

REL ALG

π ($\sigma_{\text{seller\_id='S012345'}}$ (Product))

Projection
Find all order ids in orders
10 - SELECT Order_ID from Orders

REL ALG

$\pi_{\text{Order\_ID}}$ (Product)

Some of the other queries used in the project were:

```
-- To verify the login credentials of a user
-- and hence show the details of the users
select user_id,user_name,phoneNo,email
from users
where user_id='S01' and passcode=sha2('123a''S01',256);

-- To show the customer order history ordered by date for a given customer
select Orders.user_id,Orders.order_id,Orders.product_id,product.seller_id,Orders.cost,Orders.order_date
from Orders ,product
where Orders.user_id='C12' and Orders.product_id=product.product_id order by(Orders.order_date) desc;

-- To find the details of the sellers who have successfully sold any product
select seller_id,user_name,email,phoneNo
from seller where exists(select user_id from sellerorderhistory
                                        where sellerorderhistory.user_id=seller_id );

-- To find all the products that are sold by a given seller
select product.product_id,product.product_name,product.product_description,product.cost
from sellerorderhistory,product
where sellerorderhistory.user_id='S03' and product.seller_id=sellerorderhistory.user_id
and product.product_id=sellerorderhistory.product_id;

-- It may happen that a seller has not sold any product
-- so where exists need to be used

select product.product_id,product.product_name,product.product_description
from sellerorderhistory,product
where sellerorderhistory.user_id='S17' and product.seller_id=sellerorderhistory.user_id
and product.product_id=sellerorderhistory.product_id
and sellerorderhistory.user_id in
(select seller_id
 from seller where exists(select user_id from sellerorderhistory where user_id=seller_id));

-- To identify the category which is facing shortage of product supplies
-- The threshold can be specified by the admin
-- here it is 3

select cat_id,category_name
from category
where prod_count_cat(category_name)<3;

-- To point out to the greatest order in terms of received by a given seller
-- if it has successfuly cracked any transaction

select user_id,max(product.cost)
from sellerorderhistory,product
where user_id='S03' and sellerorderhistory.product_id=product.product_id;

-- To find out the seller who has helped to reach the order targets in terms of cost
-- The threshold decided by the admin here it is 1000

select seller_id,user_name,email,phoneNo
from seller
where seller_max_order(seller_id)>1000 ;



-- To find the sellers which
-- are flagged for poor reviews by the customers

select s.selller_id,s.user_name,s.email,s.phoneNo
from seller as s,flagsellers
where flags!=0 and s.seller_id=flagsellers.seller_id;
```

<u>Week 7</u>

<u>Accessing SQL from programming language</u>

Two Approaches

- ○ Dynamic SQL
- ○ Embedded SQL

● Dynamic SQL

    ○ At runtime
    ○ SQL queries as character strings, submit to the database, and then retrieve the response
- Embedded SQL
    ○ SQL queries are identified at the compile time using a pre-processor
    ○ Pre-processor submits the queries to the database for precompilation and Optimization
    ○ Replace the SQL queries with the optimized code

We will be using python as the host language for embedded sql

The following are the embedded SQL queries and advanced aggregation functions that support the application features:

The statement cursor.execute(query,values) is used to execute the embedded queries

1 - COUNT

```
def count_above_7():

    cursor=db.cursor()

    query="SELECT COUNT(reviews) from Product WHERE reviews>7"

    try:
        cursor.execute(query)
        result=cursor.fetchall()
        print(result)

    except Error as error:
        print("Failed to count reviews {}".format(error))
        return -1
```

2 - AVG

```
def avg_review_product(Prod_Name):

    cursor=db.cursor()

    values(Prod_Name)

    query="SELECT AVG(reviews) from Product where Prod_Name=%s"
```

```
	try:
		cursor.execute(query,values)
		result=cursor.fetchall()
		print(result)
	except Error as error:
		print("Failed to find avg of product{}".format(error))
		return -1
```

3 - SUM

```
def sum_quantity_cart(Cart_ID):

	cursor=db.cursor()

	values(Cart_ID)

	query="SELECT SUM(Quantity) FROM Cart WHERE Cart_ID=%s"

	try:
		cursor.execute(query,values)
		result=cursor.fetchall()
		print(result)

	except Error as error:
		print("Failed to find sum of quantity in cart{}".format(error))
		return -1
```

4 - SELECT

```
def Show_Product(Prod_name, startprice, endprice):

	cursor=db.cursor()

	values(Prod_name,startprice,endprice)
	query="SELECT * FROM Product WHERE Product_name=%s AND Cost >= %s AND Cost <=
%s"

	try:
		cursor.execute(query)
		record=cursor.fetchall()
		db.commit()
```

```
    except Error as error:
        print("Failed to find products {}".format(error))
        return -1

    for i in record:
        print(i)

    return 0
```

5 - INSERT

```
def Add_to_Orders(i):

    cursor=db.cursor()

    Order_ID=generate_Order_ID()+1
    User_ID=i[3]
    Product_ID=i[2]
    Order_Date=i[5]
    Cost=i[4]
    PaymentStatus="Cash on Delivery"
    Status="Not Delivered"

    values=(Order_ID,User_ID,Product_ID,Order_Date,Cost,PaymentStatus,Status)
    query="INSERT INTO Orders (order_id, user_id, product_id, order_date, cost, PaymentMode,
status) VALUES (%s,%s,%s,%s,%s,%s,%s)"
    try:
        cursor.execute(query,values)
        db.commit()
    except Error as error:
        print("Failed to add to orders {}".format(error))
```

6 - UPDATE

In Seller.py

```
def update_product(sel_id):

    cursor=db.cursor()
    values=(sel_id)
    query="SELECT * FROM Product"
```

```
try:
        cursor.execute(query)
        result=cursor.fetchall()
        for i in result:
                print(i)

except Error as error:
        print("Failed to calculate income {}".format(error))

print(" ")

print("From these products enter the product_id of the product you want to delete.")

prod_id=int(input())

dict1={}
print("Select one of the following attributes:")

dict1[1]="product_name"
dict1[2]="product_description"
dict1[3]="cost"
dict1[4]="quantity"

print("1. Product Name")
print("2. Product Description")
print("3. Cost")
print("4. Quantity")

attribute=int(input())

if(attribute==1 or attribute==2):
        new_value=str(input())
elif(attribute==3):
        new_value=float(input())
else:
        new_value=int(input())

attribute=dict1[attribute]

cursor=db.cursor()
values=(new_value,prod_id)

query="UPDATE Product SET attribute = %s WHERE product_id = %s"
```

```
    try:
        cursor.execute(query,values)
        db.commit()
    except Error as error:
        print("Failed to update product from Product table {}".format(error))
```

7 - DELETE

```
def Checkout_Cart(Customer_id):

    cursor=db.cursor()
    values=(Customer_id)
    query="SELECT * FROM Cart WHERE Customer_id=%s"
    try:
        cursor.execute(query,values)
        result=cursor.fetchall()
        for i in result:
            Add_to_Orders(i)
            print(i)

    except Error as error:
        print("Failed to Checkout {}".format(error))


    query="DELETE FROM Cart where Customer_id=%s"
    try:
        cursor.execute(query,values)

    except Error as error:
        print("Failed to Checkout {}".format(error))
```

8 - MAX

```
def max_cost_product():

    cursor=db.cursor()

    query="SELECT MAX (Cost) FROM Product"

    try:
        cursor.execute(query)
        result=cursor.fetchall()
        print(result)

    except Error as error:
        print("Failed to find max of cost in product{}".format(error))
        return -1
```

9 - MIN

```
def min_cost_product():

    cursor=db.cursor()

    query="SELECT MIN (Cost) FROM Product"

    try:
        cursor.execute(query)
        result=cursor.fetchall()
        print(result)

    except Error as error:
        print("Failed to find min of cost in product{}".format(error))
        return -1
```

Function and procedure

● Procedures and functions allow "business logic" to be stored in the database and executed from SQL statements.

- ○ how many courses a student can take in a given semester,
- ○ the minimum number of courses a full-time instructor must teach in a year,
- ○ the maximum number of majors a student can be enrolled in,

Function used in the project are listed below:

```
----------------------------------------------------------------------------------------------------
-- Function to count the varieties of product available(distinguished on the basis of product_id)
-- under a  given category name as input.

-- select cat_id,category_name from category where prod_count_cat(category_name)<3;

DELIMITER //
CREATE FUNCTION `prod_count_cat`(category varchar(50)) RETURNS int(11)
```

```
    READS SQL DATA
    DETERMINISTIC
begin
declare prod_count integer;
select count(*) into prod_count from product
where product.category = category;
return prod_count;
end
//
DELIMITER ;

------------------------------------------------------------------------------------------------------------
-Triggers used in the project are listed below:
------------------------------------------------------------------------------------------------------------
DELIMITER $$
create trigger generalize_user
after insert
on users
for each row begin
 if new.user_id like 'S%' then
            insert into seller values ( new.user_id,new.user_name,new.passcode,new.email,new.phoneNo);
  insert into flagSellers values ( new.user_id,0);
 else
            insert into customer values (
new.user_id,new.user_name,new.passcode,new.email,new.phoneNo);
 end if;
 end $$
DELIMITER ;
------------------------------------------------------------------------------------------
-- whenever a new product is added into the product table it automatically gets filled to the
-- seller inventory table

DELIMITER $$
create trigger seller_inventory
after insert
on product
for each row begin
SET @v1 := (select Seller_id from product where product_id=new.product_id);
            insert into SellerInventory values ( @v1,new.product_id,new.quantity,CURDATE());
end $$
DELIMITER ;
------------------------------------------------------------------------------------------
-- special priority and discounts for the product listed in covid emergency category
```

```
DELIMITER $$
create trigger corona_help
before insert
on orders
for each row begin
SET @v1 := (select category from product where product_id=new.product_id);
if @v1 like '%Covid%' then
            set new.cost=new.cost-0.5*new.cost;
    set new.status=concat(new.status,'_On Priority');
end if;
end $$
DELIMITER ;

------------------------------------------------------------------------------------------------
-- whenever an order is delivered successfully it is added
-- to the sellerhistory table

DELIMITER $$
create trigger fill_seller_history
before insert
on orders
for each row begin
SET @v1 := (select Seller_id from product where product_id=new.product_id);
-- if @v1 like 'T1' then
if new.status like 'DELI%' then
            insert into sellerorderhistory values (
@v1,new.product_id,new.order_id,new.order_date,new.paymentmode);
end if;
end $$
DELIMITER ;


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

-- triggers to flagSellers depending upon the rating they have received

DELIMITER $$
create trigger feedback_response
after insert
on feedback
for each row begin
SET @v1 := (select Seller_id from product where product_id=new.product_id);
-- if @v1 like 'T1' then
if new.ratingpoints <1.5 then
```

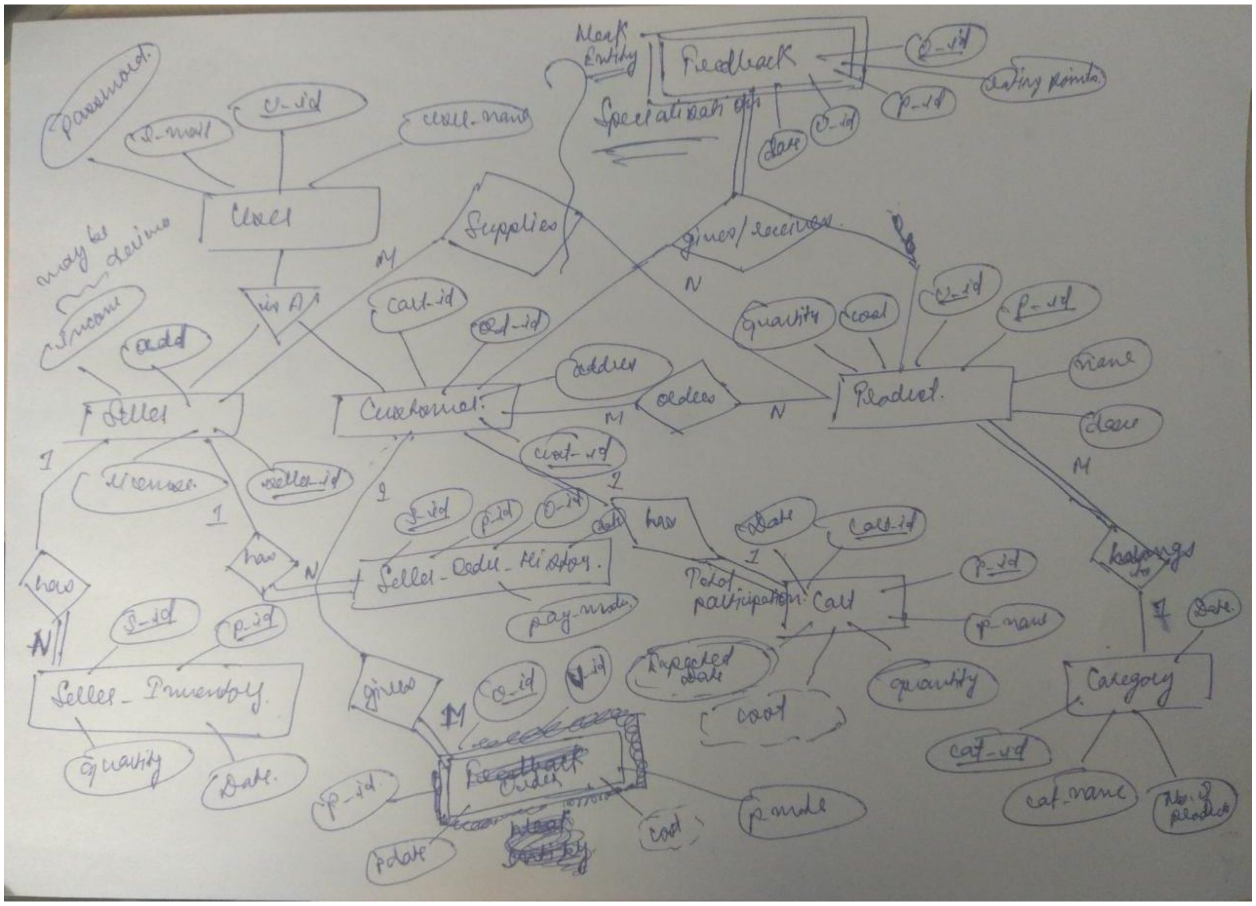
```
        update flagSellers set flags=flags+1 where seller_id=@v1 ;
end if;
end $$
DELIMITER ;
```

Week 8

Draw the E-R model for your application and update your w-in-p Document.

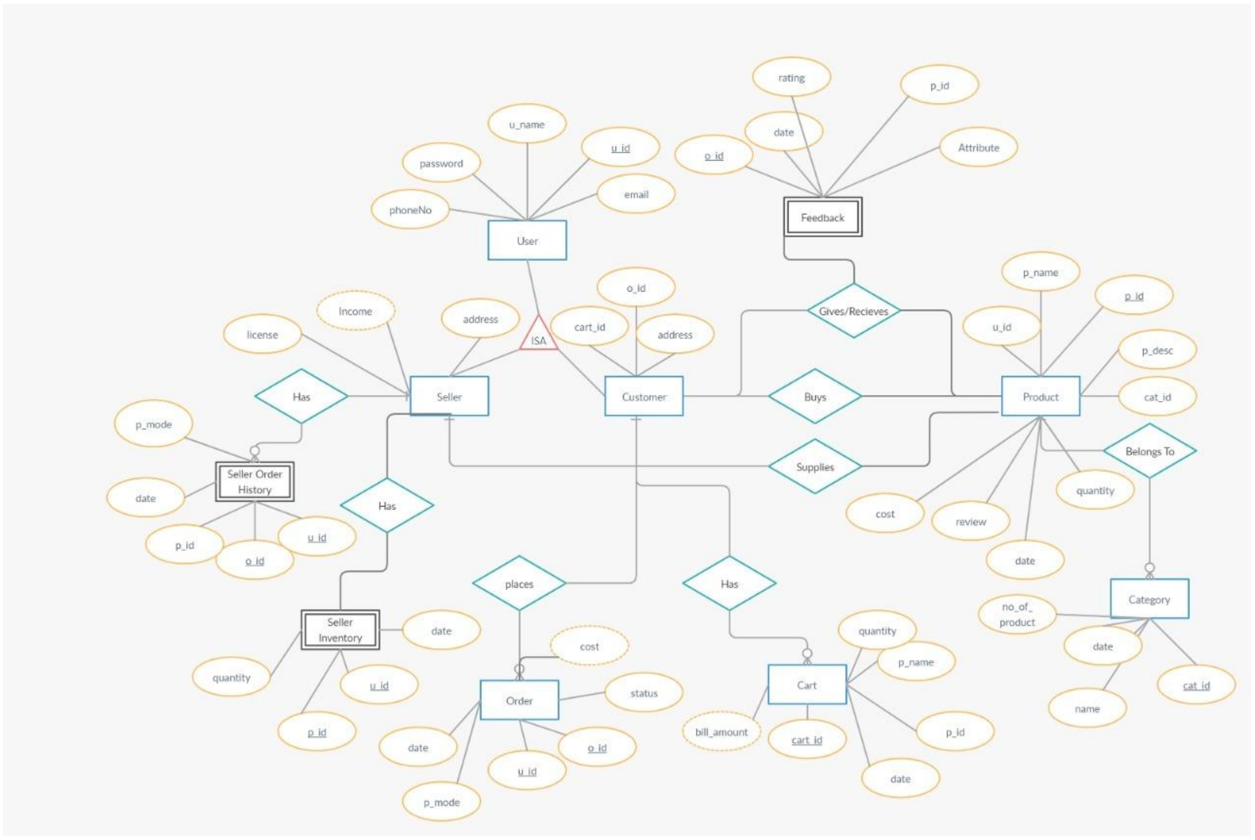
- Identifying the entity and relationship involved in the database
-Identifying weak entities and relationships.

-Drafting a rough E-R model shown below



-looking out for binary and ternary relationships and other advanced concepts of specialization and generalization

-ER model designed using creately for better visualization is shown below:



Week 9

Continue your previous weeks work.

-Regular weekly implementation of the project.
-As instructed in the slides

Week 10

Some of the innovative ideas introduced in this project are listed below:

1.salt hashing algorithm for storing passwords (Database Security)

-SHA-256 Cryptographic Hash Algorithm.
-A cryptographic hash (sometimes called 'digest') is a kind of 'signature' for a text or a data file. -SHA-256 generates an almost-unique 256-bit (32-byte) signature for a text.

-storing user password as plain text is a very naive approach
-so we are using salt hashing algorithm where the password is salted with the userid

```
insert into Users values ( 'S01','Raman',sha2('123a"S01',256),'raman@mahoo.com','9711912198' ),(
'C02','Shyam',sha2('123a"C02',256),'shyam@mahoo.com','9711912199' ),
```

```
'C02', 'Shyam', '4ce5cff2264b7272c131bf26d9e803175e6388110f1a05387ae82de7e5ab743a',
'shyam@mahoo.com', '9711912199'
```

'123a"C02'-------- '4ce5cff2264b7272c131bf26d9e803175e6388110f1a05387ae82de7e5ab743a'

2.Corona Sales

Providing an automatic 50% discount on the products under the category of ('T1','Covid Emergencies')

3.On Priority Orders

Providing the highest priority to the products ordered under the COVID Emergencies categories.

The above features are implemented with the help of triggers.

```
delimiter $$
create trigger corona_help
before insert
on orders
for each row begin
```

```
SET @v1 := (select cat_id from product where product_id=new.product_id);
if @v1 like 'T1' then
            set new.cost=new.cost-0.5*new.cost;
    set new.status=concat(new.status,'_On Priority');
end if;
end$$
delimiter ;
```

4.Rating System

- It helps the customer to rate the orders on a scale from 1 to 5
- It also helps to flag the sellers who have received less than a threshold
- Sellers beyond a certain no. of flag can also be removed from the database.

5.B Trees and Hash indexing

-The number of products ordered on an eCommerce website can be extensive so multilevel indexing popularly known as B Trees indexing is used.

```
CREATE UNIQUE INDEX idx_orderid
USING BTREE
ON orders(order_id);
```

-Similarly for product table hash indexing is used

```
CREATE INDEX idx_proid
USING HASH
ON product(product_id);
```

6.Triggers

-To keep the database updated and consistent the triggers
Are added at all the places wherever necessary.

7.App in development

-An effort to provide an interactive user interface connected to the database.

The key innovative feature and Implementation

A front end GUI has been implemented to support the application and database that has been created.