DBMS Project
End-Sem Presentation

Jan - Apr 2022

Furno™

C H E T A N | C H A R V I | J O G I T H | O J U S
2020046 | 2020045 | 2020072 | 2020094
G R O U P - 55

Contents

Link to GitHub Repository

https://github.com/OjusSinghal/furno

Recap of mid-sem submission

Mid-Sem presentation link

ER Diagram - Mid Sem

Entities - Mid Sem:

- buyer
- seller
- productListing
- brand
- review
- cartItem
- cart
- order
- paymentMethod
- complaints
- productSale
- promoCode

Relational Schema - Mid Sem

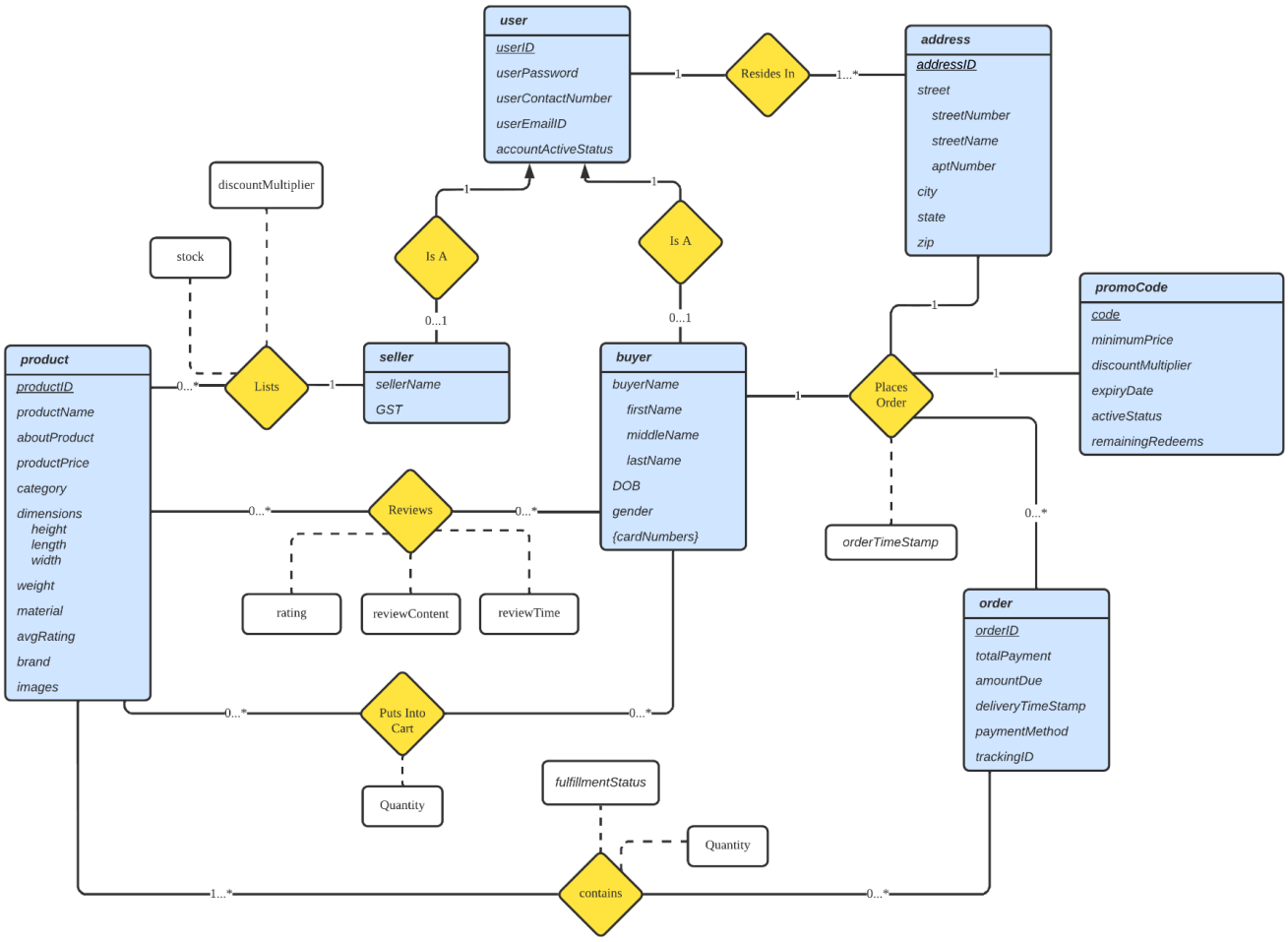
- buyer(buyerID, firstName, middleName, lastName, DOB, age, gender, contactNumber, emailID, password, streetNumber, streetName,aptNumber, city, state, zip)
- seller(sellerID, sellerName, sellerContactNumber, sellerEmailID, GST, streetNumber, streetName,aptNumber, city, state, zip)
- productListing(productID, sellerID, productName, description, category, MRP, discount, currentPrice, stock)
- brand(brandID, brandName)
- review(buyerID, productID, sellerID, rating, comment, date)
- cartItem(cartItemID, quantity, totalCost)
- cart(cartID, buyerID, cartTotal)
- order(orderID, totalPayment, orderTimeStamp, fulfillmentStatus, deliveryTimeStamp, trackingID)

Description of changes incorporated based on mid-sem review

Changes in the ER

1. Removed foreign keys from the ER diagram
2. Linked relations with entities more accurately like reviews and product
3. Removed the inaccurately mentioned weak entity cartItem
4. Rectified the inaccurately cited ternary relationship
5. Corrected some multiplicities
6. Removed some entities causing redundancy in data storage

Link to the new ER Diagram



Changes in the Relational Schema

1. Modified the relational schema based on the updated ER diagram
2. Corrected primary keys of product
3. Optimized the storage of addresses by making it an entity

Entities
Foreign Keys
Primary Keys

buyer - (buyerID, firstName, middleName, lastName, buyerEmailID, buyerPassword, buyerContactNumber, DOB, gender, accountActiveStatus)

paymentCards - (cardNumber, *buyerID*)

seller - (sellerID, sellerName, sellerEmailID, sellerPassword, sellerContactNumber, GST, accountActiveStatus)

buyerResidesIn - (addressID, *buyerID*, streetNumber, streetName, aptNumber, city, userState, zip)

sellerResidesIn - (addressID, *sellerID*, streetNumber, streetName, aptNumber, city, userState, zip)

promoCode - (codeName, minimumPrice, discountMultiplier, expiryDate, activeStatus, remainingRedeems)

product - (productID, *sellerID*, productName, aboutProduct, productPrice, category, height, length, width, productWeight, material, brand, avgRating, discountMultiplier, stock, images)

putsIntoCart - (*buyerID*, *productID*, quantity, buyingPrice)

Orders - (orderID, *buyerID*, *codeName*, totalPayment, orderTime, amountDue, deliveryTime, trackingID, *addressID*, cardNumber)

containsProduct - (*orderID*, *productID*, quantity, fulfillmentStatus)

reviews - (*productID*, *buyerID*, review, rating, timestamp)

Other Changes

1. Created new complex queries. Previously the queries were simple.
2. Correctly populated all relations. Previously some data were missing.
3. Added useful attributes to relationships rather than making them part of the entity.
4. Reduced the number of entities and tables, vastly simplifying the database design without compromising on efficiency and redundancy.
5. Slightly updated the scope of the project to align it with the progress after mid-sem

Views

```
-- Buyer profile for the buyer to see for themselves

create view buyerProfile as select firstName, middleName, lastName,
buyerEmailID as EmailID, buyerContactNumber as contactNumber, DOB,
gender,count(orderID) as totalOrders, count(codeName) as
numberOfPromoCodesRedeemed,round(avg(timestampdiff(day, orderTime,
deliveryTime))) as avgDeliveryTime from buyer natural join orders where
deliveryTime is not null group by buyerID;

-- Seller profile for the seller to see for themselves

create view sellerProfile as select sellerName as name, sellerEmailID as
emailID, sellerContactNumber as contactNumber, GST, sum(quantity) as
totalItemsSold,sum(quantity * productPrice) as totalRevenueGenerated,
round(avg(timestampdiff(day, orderTime, deliveryTime))) as avgDeliveryTime
from seller natural join orders natural join containsProduct natural join
product group by sellerID;

-- product profile for the admin to see statistics about a product

create view productView as select productName as Name, aboutProduct as
about, productPrice as MRP, category, avgRating as rating,
round(sum(quantity * productPrice * discountMultiplier)) as
totalRevenueGenerated,sum(quantity) as totalItemsSold, count(distinct
buyerID) as numberOfBuyersWhoBaughtThis from seller natural join orders
natural join containsProduct natural join product group by productID;

-- what the buyer views their cart as

create view cart as select productName as name, productPrice as MRP,
quantity, brand, avgRating as rating, round(100 - 100 *
discountMultiplier) as discountPercentage, productPrice *
discountMultiplier * quantity as totalDiscountedPrice, sellerName from
putsIntoCart natural join product natural join seller;
```

```sql
-- buyer about a product while browsing

create view productBrowsing as select productName as Name, sellerName,
aboutProduct as about, productPrice as MRP, category, dimensionsHeight,
dimensionsLength, dimensionsWidth, productWeight as weight, material,
brand, avgRating as rating, round(discountMultiplier * productPrice) as
sellingPrice from product natural join seller;

-- buyer about their order

create view orderDetails as (select orderID, totalPayment, date(orderTime)
as orderDate, case when amountDue > 0 then "unpaid" else "paid" end as
paymentStatus, deliveryTime, trackingId, paymentMethod as paidUsing,
street, city, userState, zip from orders natural join buyerResidesIn);
```

Grants designed for different roles

Roles Identified for employees at Furno™:

1. Super admin - can do *anything*
2. Admins - Control the information of specific databases
    a. SellerSideAdmin
    b. BuyerSideAdmin
    c. ProductSideAdmin
3. Developers - They need to show information on the website
    a. SellerSideDeveloper
    b. BuyerSideDeveloper
    c. ProductSideDeveloper

```
-- super admin, can do everything
CREATE USER 'superAdmin'@'localhost' IDENTIFIED BY
'superAdminPassword123!';

-- static privileges
GRANT SELECT, INSERT, UPDATE, DELETE, CREATE, DROP, RELOAD, SHUTDOWN,
PROCESS, FILE, REFERENCES, INDEX, ALTER, SHOW DATABASES, CREATE TEMPORARY
TABLES, LOCK TABLES, EXECUTE, REPLICATION SLAVE, REPLICATION CLIENT,
CREATE VIEW, SHOW VIEW, CREATE ROUTINE, ALTER ROUTINE, CREATE USER, EVENT,
TRIGGER, CREATE TABLESPACE, CREATE ROLE,  DROP ROLE ON *.* TO
`superAdmin`@`localhost` WITH GRANT OPTION;


-- dynamic privileges (everything the "root" user has)
GRANT APPLICATION_PASSWORD_ADMIN, AUDIT_ABORT_EXEMPT, AUDIT_ADMIN,
AUTHENTICATION_POLICY_ADMIN, BACKUP_ADMIN, BINLOG_ADMIN,
BINLOG_ENCRYPTION_ADMIN, CLONE_ADMIN, CONNECTION_ADMIN,
ENCRYPTION_KEY_ADMIN, FLUSH_OPTIMIZER_COSTS, FLUSH_STATUS, FLUSH_TABLES,
```

```
FLUSH_USER_RESOURCES, GROUP_REPLICATION_ADMIN, INNODB_REDO_LOG_ARCHIVE,
INNODB_REDO_LOG_ENABLE, PASSWORDLESS_USER_ADMIN,
PERSIST_RO_VARIABLES_ADMIN, REPLICATION_APPLIER, REPLICATION_SLAVE_ADMIN,
RESOURCE_GROUP_ADMIN, RESOURCE_GROUP_USER, ROLE_ADMIN,
SERVICE_CONNECTION_ADMIN, SESSION_VARIABLES_ADMIN, SET_USER_ID,
SHOW_ROUTINE, SYSTEM_USER,  SYSTEM_VARIABLES_ADMIN,
TABLE_ENCRYPTION_ADMIN, XA_RECOVER_ADMIN ON *.* TO
`superAdmin`@`localhost` WITH GRANT OPTION;

-- needed to enable an external user to connect as and have the privileges
of another user
GRANT PROXY ON ``@`` TO `superAdmin`@`localhost` WITH GRANT OPTION;

-- sellerSideAdmin
-- maintains the databases concerning the seller
CREATE USER 'sellerSideAdmin'@'localhost' IDENTIFIED BY
'sellerSideAdminPassword123!';

grant SELECT, INSERT, UPDATE, DELETE, CREATE on seller to
'sellerSideAdmin'@'localhost';
grant SELECT, INSERT, UPDATE, DELETE, CREATE on sellerResidesIn to
'sellerSideAdmin'@'localhost';
grant SELECT, INSERT, UPDATE, DELETE, CREATE on orders to
'sellerSideAdmin'@'localhost';
grant SELECT, INSERT, UPDATE, DELETE, CREATE on containsProduct to
'sellerSideAdmin'@'localhost';



-- buyerSideAdmin
-- maintains the databases concerning the buyer
CREATE USER 'buyerSideAdmin'@'localhost' IDENTIFIED BY
'buyerSideAdminPassword123!';

grant SELECT, INSERT, UPDATE, DELETE, CREATE on buyer to
'buyerSideAdmin'@'localhost';
grant SELECT, INSERT, UPDATE, DELETE, CREATE on buyerResidesIn to
'buyerSideAdmin'@'localhost';
grant SELECT, INSERT, UPDATE, DELETE, CREATE on orders to
'buyerSideAdmin'@'localhost';
```

```sql
grant SELECT, INSERT, UPDATE, DELETE, CREATE on containsProduct to
'buyerSideAdmin'@'localhost';
grant SELECT, INSERT, UPDATE, DELETE, CREATE on putsIntoCart to
'buyerSideAdmin'@'localhost';
grant SELECT, INSERT, UPDATE, DELETE, CREATE on reviews to
'buyerSideAdmin'@'localhost';
grant SELECT, INSERT, UPDATE, DELETE, CREATE on paymentCards to
'buyerSideAdmin'@'localhost';


-- productSideAdmin
-- maintains the databases concerning the product
CREATE USER 'productSideAdmin'@'localhost' IDENTIFIED BY
'productSideAdminPassword123!';

grant SELECT, INSERT, UPDATE, DELETE, CREATE on product to
'productSideAdmin'@'localhost';
grant SELECT, INSERT, UPDATE, DELETE, CREATE on orders to
'productSideAdmin'@'localhost';
grant SELECT, INSERT, UPDATE, DELETE, CREATE on containsProduct to
'productSideAdmin'@'localhost';
grant SELECT, INSERT, UPDATE, DELETE, CREATE on promoCode to
'productSideAdmin'@'localhost';
grant SELECT, INSERT, UPDATE, DELETE, CREATE on reviews to
'productSideAdmin'@'localhost';


-- web developer for product page
create user 'productPageDeveloper'@'localhost' IDENTIFIED BY
'productPageDeveloperPassword123!';
grant select on productBrowsing to 'productPageDeveloper'@'localhost';
grant select on promoCode to 'productPageDeveloper'@'localhost';
grant select on reviews to 'productPageDeveloper'@'localhost';


-- web developer for buyer profile page
create user 'buyerProfileDeveloper'@'localhost' identified by
'buyerProfileDeveloperPassword123!';
grant select on buyerProfile to 'buyerProfileDeveloper'@'localhost';
grant select on orderDetails to 'buyerProfileDeveloper'@'localhost';
```

```sql
grant select on cart to 'buyerProfileDeveloper'@'localhost';
grant select on buyerResidesIn to 'buyerProfileDeveloper'@'localhost';


-- web developer for seller profile page
create user 'sellerProfileDeveloper'@'localhost' identified by
'sellerProfileDeveloperPassword123!';
grant select on sellerProfile to 'sellerProfileDeveloper'@'localhost';
grant select on product to 'sellerProfileDeveloper'@'localhost';
grant select on sellerResidesIn to 'sellerProfileDeveloper'@'localhost';
```

Main SQL Queries

1

```
-- Top 10 Sellers of all time in terms of total money earned displayed in
the order of their earnings(descending order)

SELECT sellerID, SUM(productPrice*quantity) as total
FROM containsproduct, product
WHERE containsproduct.productID = product.productID
GROUP BY sellerID
ORDER BY total DESC LIMIT 10;
```

2

```
-- Percentage of people who placed an order in the last month against
total registered buyers

SELECT CONCAT(ROUND(( COUNT(distinct orders.buyerID)/COUNT(distinct
buyer.buyerID) * 100 ),2),'%') AS percentage
FROM orders,buyer
WHERE (MONTH(CURDATE()) - MONTH(orders.deliveryTime)) <= 1 AND
(MONTH(CURDATE()) - MONTH(orders.deliveryTime)) >=0;
```

3

```
-- The most expensive items in each of the category along with the name of
that product and the seller

SELECT DISTINCT category, max(productPrice) AS MaximumPrice, productName,
sellerName
FROM product, seller
WHERE product.sellerID = seller.sellerID
GROUP BY category
ORDER BY MaximumPrice;
```

4

```
-- Sale, 20 percent off on beds which have more than 5 pieces left in the
stock.

UPDATE product
SET discountMultiplier = 0.2
WHERE stock >5 AND category = "beds";
```

5

```
-- Advanced search bar where you can search for a product by its name,
description, brand or even category. We prioritised the name of the
product over everything else and then sorted in ascending order of the
category. e.g. chair

SELECT productName, category, aboutProduct, brand
FROM product
WHERE productName LIKE '%chair%' OR aboutProduct LIKE '%chair%' OR brand
LIKE '%chair%' OR category LIKE '%chair%'
order by productName = '%chair%', category;
```

6

```

-- View the total amount of the items and their quantities present in the
cart after applying the discount for a particular buyer eg. B009219197

SELECT (SUM(productPrice*(1-discountMultiplier)*quantity)) AS total
FROM product, putsIntoCart
WHERE putsIntoCart.productID = product.productID AND putsIntoCart.buyerID
= 'B009219197';
```

7

```
-- Edit the product description of the products which are out of stock

UPDATE product
SET aboutProduct = CONCAT("OUT OF STOCK ", aboutProduct)
WHERE stock <= 0;
```

8

```
-- Delete from orders which are more than 10 years old and account is
status is deactive

DELETE FROM orders
WHERE (YEAR(CURDATE()) - YEAR(orders.ordertime))>=10
AND buyerID IN
(SELECT buyerID FROM buyer WHERE accountActiveStatus = false);
```

9

```
-- Display the status of the placed orders along with an appropriate
message for a particular buyer eg, a buyer with buyerID="B016077550".

select orderID,
(case
   when current_date()-deliveryTime > 0 then concat("Order will be
delivered soon, your tracking ID is:",trackingID)
   when current_date()-deliveryTime < 0 then concat("The order was
delivered to you on ", deliveryTime)
   else "The order will be delivered today"
end) as deliveryStatus, totalPayment
from orders
where buyerID="B016077550";
```

10

```
-- To find the names of products a buyer has ordered from a particular
seller eg. buyerID = "B016077550" and "S073588105"

select distinct productName
from product
where sellerID ="S073588105" and productID = any
(select productID
from buyer, containsProduct, orders
where buyer.buyerID = "B016077550" and buyer.buyerID = orders.buyerID and
orders.orderID = containsProduct.orderID);
```

Embedded SQL Queries

1

```
-- Find buyer City
f"select city from buyerresidesin where buyerid='{account['buyerID']}'"
```

2

```
-- Find popular products
f"select * from buyerresidesin NATURAL JOIN orders NATURAL JOIN
containsProduct NATURAL JOIN product where city = '{city['city']}' and
product.avgRating>3.5 order by product.avgRating;"
```

3

```
-- Login Queries
f"select * from buyer where buyerEmailID='{email}' and
buyerPassword='{password}'"
f"select * from seller where sellerEmailID='{email}' and
sellerPassword='{password}'"
```

4

```
-- Registration Queries
f"insert into buyer values ('{buyerid}', '{firstName}', '{middleName}',
'{lastName}', '{email}', '{password}', '{contactNumber}', '{dob}',
'{gender}',1)"
f"insert into seller values ('{sellerID}', '{sellerName}',
'{userEmailID}', '{userPassword}', '{userContactNumber}', {gst}, 1)"
```

5

```
-- -- Update Password
f"update buyer set buyerPassword='{newPass}' where
buyerID='{session['id']}'"
f"update seller set sellerPassword='{newPass}' where
sellerID='{session['id']}'"
```

6

```
-- Reviews
f"select *,firstName from reviews,buyer where
productID='{product['productID']}' and reviews.buyerID=buyer.buyerID"
```

7

```
-- Searching Product
f"select *,sellerName from product,seller where productName like
'%{keyword}%' and product.sellerID=seller.sellerID"
```

Steps taken for query optimization

1

1. INNER JOIN instead of NATURAL JOIN
2. Only sellerID and sum called, all other information can be accessed by sellerID itself

```
ALTER TABLE `buyerresidesin` ADD INDEX `buyerresidesin_idx_city` (`city`);
ALTER TABLE `product` ADD INDEX `product_idx_avgrating` (`avgRating`);
```

2

1. Only percentage used rather than all other unnecessary information
2. No join used on orders and buyer as we only need to find information about only number of buyers who have placed an order and the number of total buyers, the other information can be access through the individual tables itself.

```
ALTER TABLE orders ADD COLUMN `month_deliverytime` TINYINT GENERATED ALWAYS
AS (MONTH(deliveryTime)) VIRTUAL;
```

3

1. No NATURAL JOIN used
2. Only required elements are selected
3. Group by used before order by

```
ALTER TABLE `product` ADD INDEX `product_idx_sellerid_category` (`sellerID`,`category`);
```

4

```
ALTER TABLE `product` ADD INDEX `product_idx_category_stock` (`category`,`stock`);
```

5

1. Only required columns selected
2. In order clause more priority is given to productName specifically “chair” than category

ALTER TABLE `product` ADD INDEX `product_idx_category` (`category`);

6

1. Only selecting and calculating the cart total of a particular buyer
2. No NATURAL JOIN is used

ALTER TABLE `product` ADD INDEX `product_idx_category_stock` (`category`,`stock`);

7

1. Instead of retyping the whole description concat function is used to concatenate “out of stock” with the existing description.

ALTER TABLE `product` ADD INDEX `product_idx_stock` (`stock`);

8

1. Instead of manually inputting date, curdate() function is used so that no need to keep updating the date over and over again.
2. Nested query instead of join

ALTER TABLE `buyer` ADD INDEX `buyer_idx_accountactivestatus` (`accountActiveStatus`);

9

1. Instead of multiple nested query, we’ve used CASE WHEN syntax to print the appropriate information along with a small message
2. Instead of manually inputting date, curdate() function is used so that no need to keep updating the date over and over again.

ALTER TABLE `product` ADD INDEX `product_idx_category_stock` (`category`,`stock`);

10

1. Nested query instead of unnecessary joins
2. The order of nesting matters here because in this order we are using no join in the parent and three joins in the child query, but if the order was reversed we would have used four joins in the parent and no join in the child query, hence preventing an extra join for better runtime.

```
ALTER TABLE `orders` ADD INDEX `orders_idx_buyerid_orderid` (`buyerID`,`orderID`);
```

Indices created:

```
--buyerResidesIn--
ALTER TABLE `buyerresidesin` ADD INDEX `buyerresidesin_idx_city` (`city`);

--product--
ALTER TABLE `product` ADD INDEX `product_idx_category_stock`
(`category`,`stock`);

--buyer-
ALTER TABLE `buyer` ADD INDEX `buyer_idx_accountactivestatus`
(`accountActiveStatus`);

--orders--
ALTER TABLE `orders` ADD INDEX `orders_idx_buyerid_orderid` (`buyerID`);

--buyer--
ALTER TABLE `buyer` ADD INDEX `buyer_idx_buyerid_orderid` (`buyerID`);

--orders--
ALTER TABLE `containsProduct` ADD INDEX
`containsProduct_idx_buyerid_orderid` (`orderID`);
```

Triggers

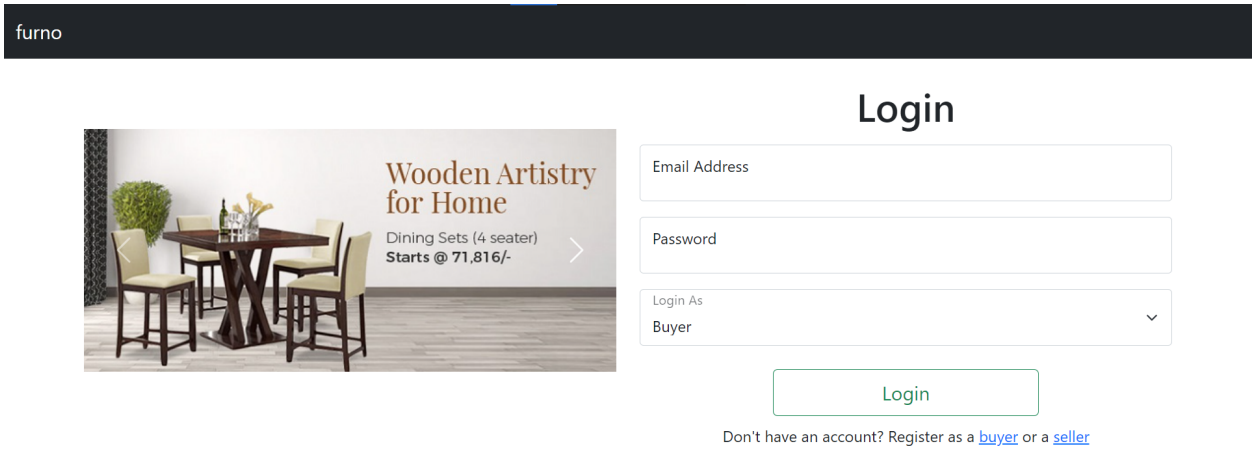
```
CREATE TRIGGER reviews_ai AFTER INSERT ON reviews
FOR EACH ROW
   UPDATE product SET avgRating = (select AVG(rating) FROM reviews WHERE
productID = NEW.productID);
```

```

delimiter ##
CREATE TRIGGER cart_bi BEFORE INSERT ON putsIntoCart
FOR EACH ROW
BEGIN
update putsIntoCart
set quantity = (case
   when NEW.productID IN (SELECT productID FROM putsIntoCart WHERE buyerID
= NEW.buyerID) THEN
       quantity = quantity+ NEW.quantity
   ELSE
       quantity = NEW.quantity
   END )
where productID = NEW.productID AND buyerID = NEW.buyerID;
END ##
delimiter ;

```

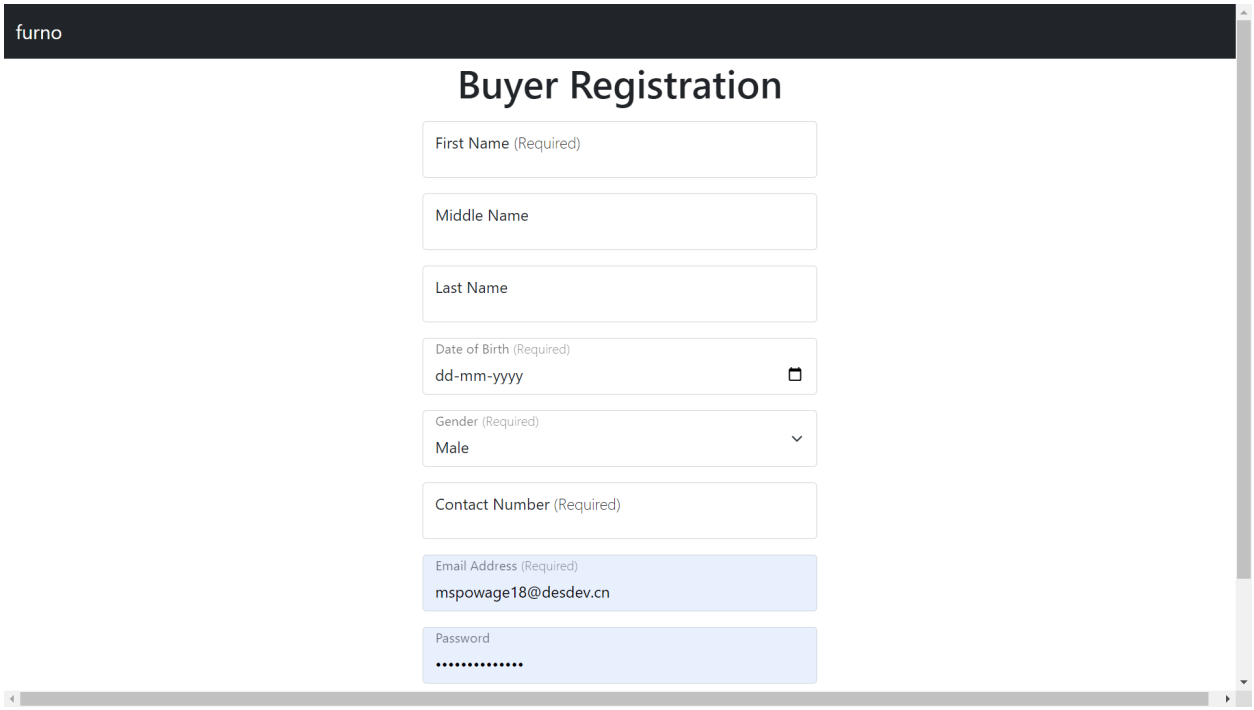
```
DELIMITER $$
CREATE TRIGGER orders_ai AFTER INSERT ON orders
FOR EACH ROW
BEGIN
UPDATE promoCode SET remainingRedeems = remainingRedeems -1 WHERE
promoCode.codeName = codeName AND remainingRedeems >= 1;
UPDATE promoCode SET activeStatus = 0 WHERE remainingRedeems = 0 OR
CURDATE() > expiryDate;
END$$
DELIMITER ;
```

Screenshots of UI

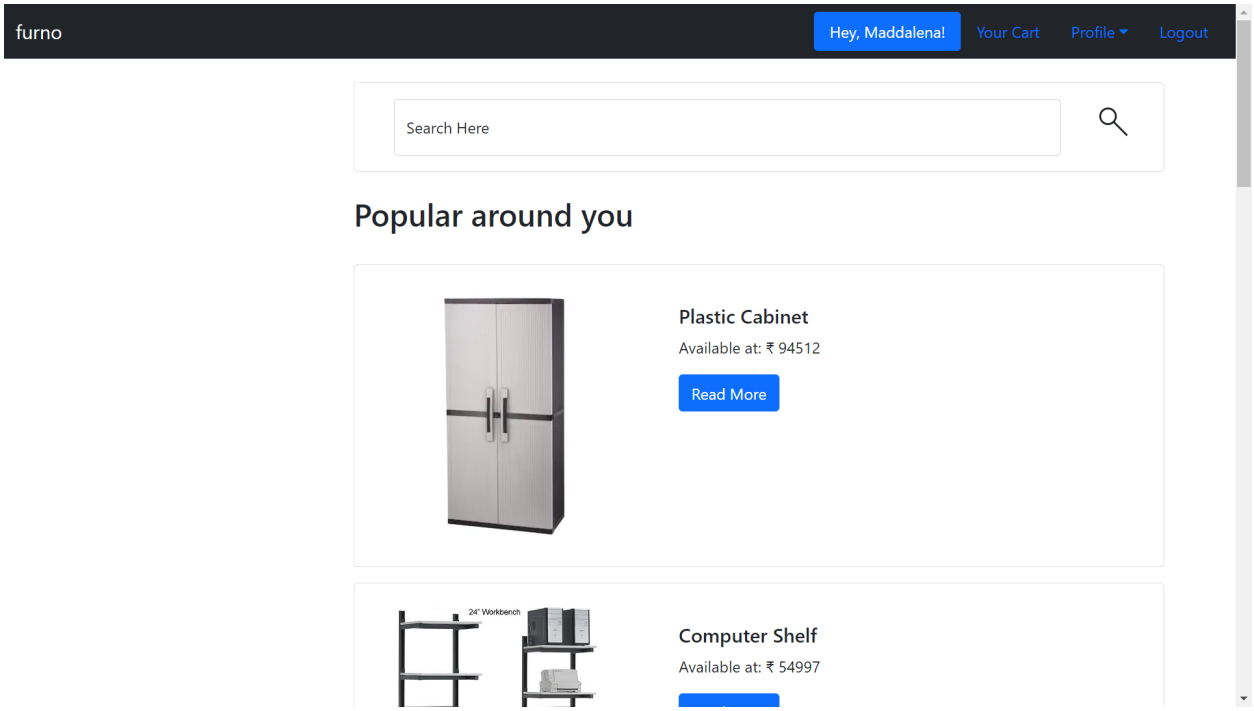
Login



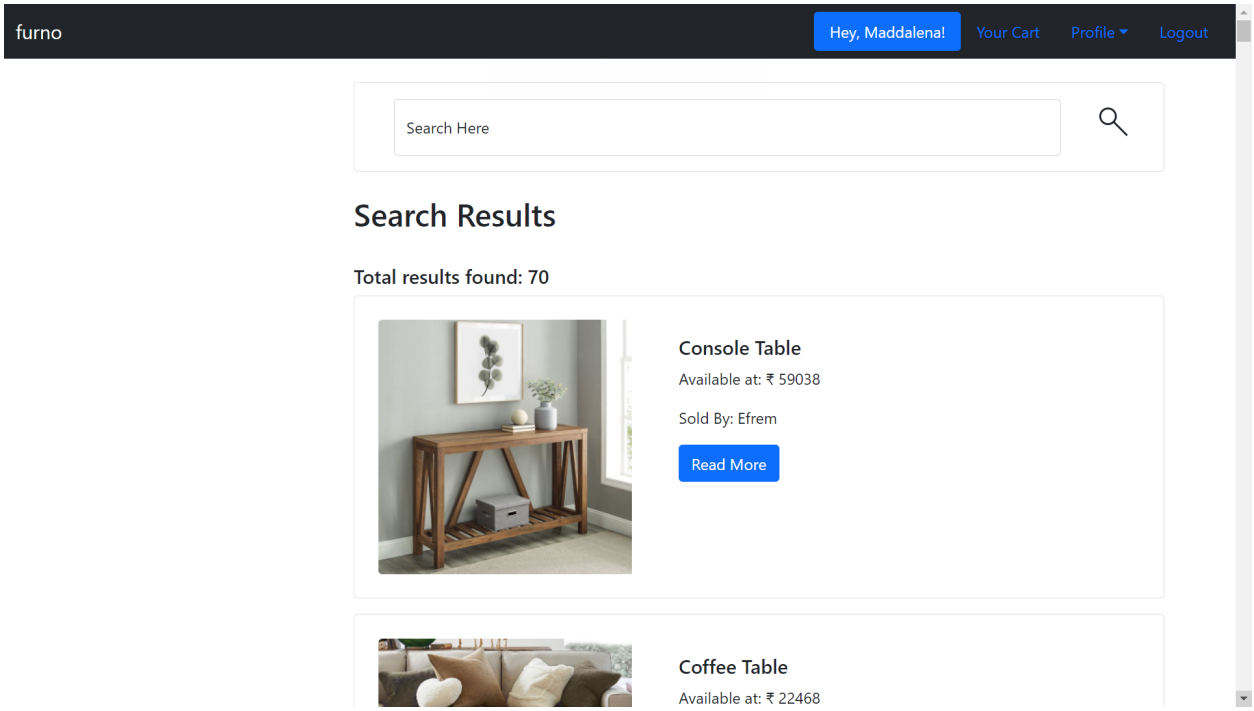
Registration



Buyer Home Page



Search Results



Product Information

furno

Hey, Maddalena! Your Cart Profile Logout

Sofa Spa

Price
67156

Dimensions
18.61 x 70.27 x 70.61

Weight
1459.31

Material
Fibers

Brand
Yakitri

Average Rating
4.72

Stock
91

non quam nec dui luctus rutrum nulla tellus in sagittis dui vel

View Profile

furno

Hey, Maddalena! Your Cart Profile Logout

Profile

First Name
Maddalena

Middle Name
Oakeby

Last Name
Spowage

Email ID
mspowage18@desdev.cn

Contact Number
4369571732

Date of Birth
2000-05-10

Gender
M

Address-1

Street: Forest
City: Miami
State: Florida
Zip: 170713

Address-2

Street: Melody
City: Reno
State: Nevada
Zip: 112966

Address-3

Street: Russell
City: San
State: California
Zip: 123570

Saved Cards

1
4612643745855435

2
5110432700280417

Added functionality / USP

1. A smooth and dynamic Web Application was created using python frameworks such as Flask. A lot of work went into designing and developing the application's front-end.
2. Multiple roles were identified considering Furno as a company, and these roles were granted privileges to specialized views rather than the original relations.
3. The application’s functionality was designed keeping in mind the real use case, and some added functionalities were as follows:
    a. Buyers and sellers can have multiple addresses, and place orders at multiple locations.
    b. Buyers can apply promo codes on their orders to get discounts.
    c. Buyers can review and rate products and sellers can see statistics about their products

Member Contribution

Contribution common to all:

1. Ideated for the following:
    a. ER Diagram
    b. Relational Schema
    c. Triggers
    d. Views
    e. Grants
    f. Queries
    g. Website

Specific Contribution:

Chetan:

1. Designed and created the front-end and back-end of the web application
2. Implemented the schema on MySQL with constraints

Charvi Jindal:

1. Designed and implemented the SQL queries, indexes, embedded queries, and triggers.
2. Optimized the SQL queries

Jogith S Chandran:

1. Designed and implemented the SQL queries, embedded queries, and triggers.
2. Optimized the SQL queries

Ojus Singhal:

1. Implemented the schema on MySQL with constraints, and populated the database.
2. Designed and implemented the ER Diagram, views, and grants

Note: A good portion of the work was done jointly on meets.