Database Design And Implementation For E-Commerce

Sourabh Aggarwal (111601025), Nikhil Kumar Yadav (111601013)

March 27, 2019

Contents

1 Contribution

Since we are a two member team, each of us was involved constantly in each phase of the project, be it GUI, Procedures, Report etc. So, the division of task within ourselves isn't that straight forward to describe, just understand that we did everything together.

For detailed look at the different phases of our project, please refer github.

2 Introduction

In this project, our aim was to come up with a reasonably scalable database along with basic GUI for E-Commerce purpose.

We started by listing down various requirements presented in the next section. After that we proceeed on builing an ER Diagram to fullfil the same. After that it was time to implement all this in SQL. In due time, we managed to put various important features provided by almost all E-Commerce site in our project.

So the following report touches on each of these aspects in brief and sequential manner.

2.1 Requirements

Following is the list of requirements.

1. Idea of roles. Model should have role for each of Customer, Seller and Shipper. Each user is thus assigned its role.
2. Company maintains the details of it users like their name, address, phone number and email-id.
3. Each user has security credentials like their username and password.
4. It should be possible for each user to update their information including their password.
5. Customer requirements
   (a) Customers should be able to browse all products.

(b) They should be able to add those products in their cart. Thus each Customer should possess a cart into which they can add products for purchasal.

(c) Note that each product could be sold by different sellers, show customer should be able to select a product corresponding to the seller he or she wished to buy from.

(d) Once all the items are added in cart, Customer should then be able to purchase all those items in cart at once.

(e) For each of the product the Customer has bought, it should be possible for customer to pass the rating and review for both the product as well as seller.

6. Seller requirements

   (a) Each Seller possesses a rating which he or she should be able to see. This rating is given and updated by the users of role Customer who purchased the product from Seller.

   (b) Seller should list down the products he or she is willingly to sell. Each product should have:

      i. Product Name

      ii. Product Image

      iii. Price

      iv. Quantity which the seller possess of the product

      v. Pickup address

      vi. Description

      vii. Rating (Product rating is again updated by the users of role Customer)

   (c) It should be possible for seller to add or update his or her products.

   (d) It should be possible for seller to see his or her past sellings.

   (e) It should be possible for seller to see the products which he or she has listed.

   (f) It should be possible for seller to see their earnings in a specific duration.

(g) Seller should be able to browse Shippers so that he or she can decide and contact various Shippers.

(h) It should be possible for seller to see the latest purchasal's of his or her products which are pending to be shipped, thus seller should also have an interface to update customer with the sold product's shipment Tracking ID.

7. Shipper requirements

   (a) It should be possible for shipper to see his or her past shipments (using our interface) including source and destination of the product.

3 Entity Relation Diagram

Entity Relation Diagram can be represented in various notations, below shows the notation which we have followed.

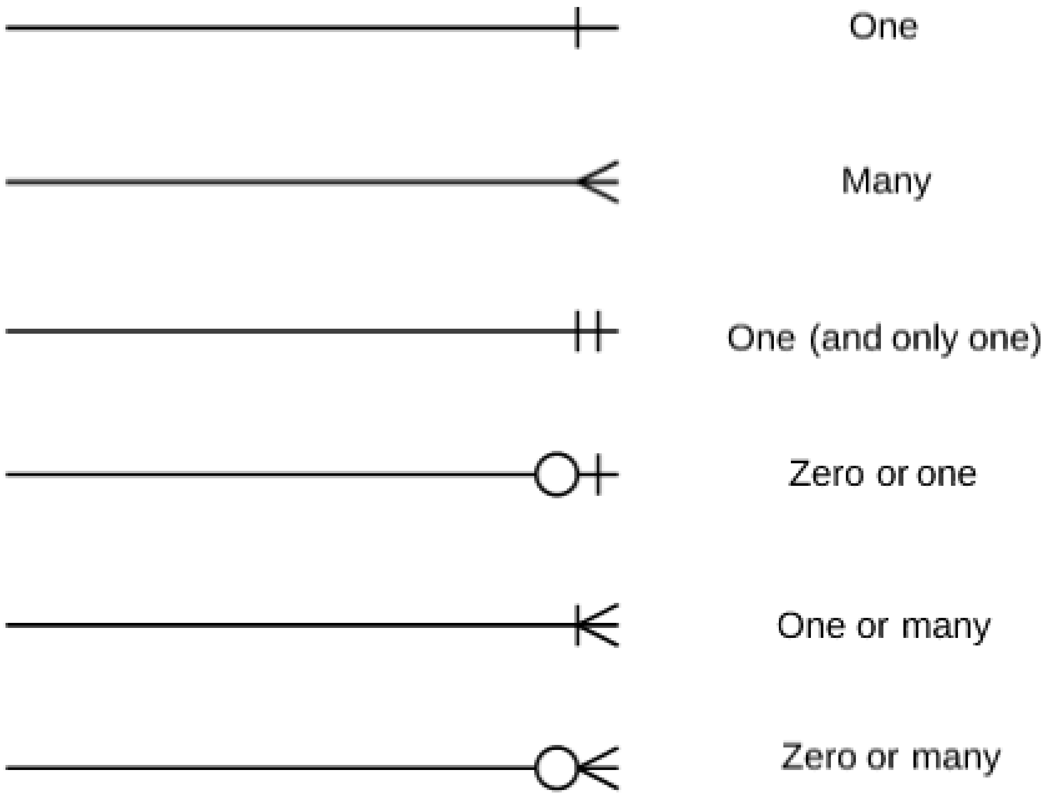


Figure 1: Entity Relation Diagram Notation

Following this notation, our ERD is represented below.

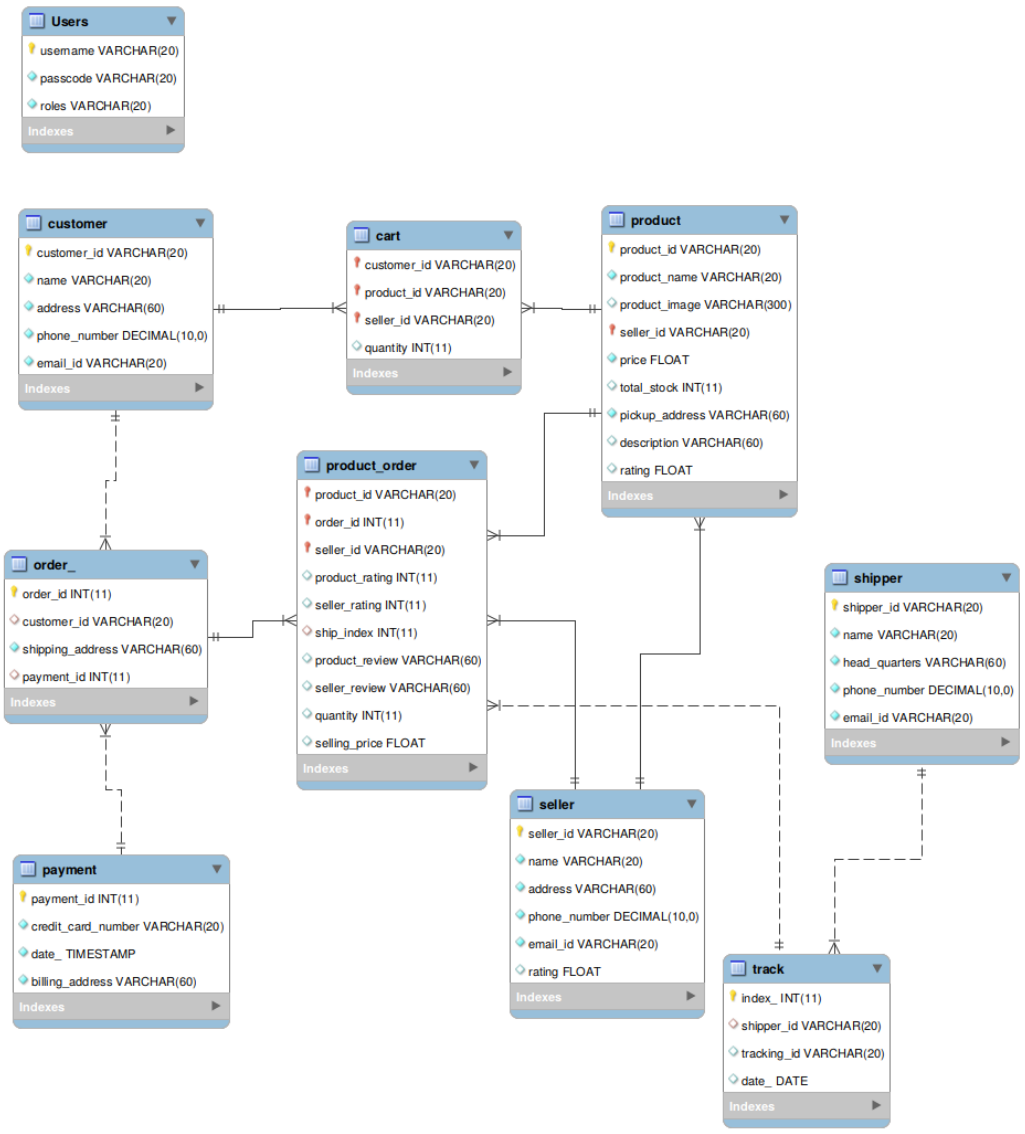


Figure 2: Entity Relationship Diagram for e-commerce

4 Database Schema And Normalization

In this section we describe various aspects of our schema and also mention that it is indeed normalized (BCNF).

Notation: A dependency $A \to B$ is called relevant if all other dependencies from A are of the form $A \to C$ where $C \subseteq B$.

- A table for basic details of customer.

  ```
  /*  Here: customer_id -> R is the only relevant
  dependency and hence it is in BCNF */
  create table customer (
    customer_id VARCHAR (20) primary key not null,
    name VARCHAR (20) not null,
    address VARCHAR (60) not null,
    phone_number DECIMAL (10) UNSIGNED not null,
    email_id VARCHAR (20) not null
  );
  ```

- A table for basic details of seller.

  ```
      /* Here: seller_id -> R is the only relevant
      dependency and hence it is in BCNF */
      /* Rating will be updated with the help of triggers.
      */
      create table seller (
        seller_id varchar (20) primary key not null,
        name varchar (20) not null,
        address varchar (60) not null,
        phone_number decimal (10) UNSIGNED NOT NULL,
        email_id VARCHAR (20) not null,
        rating float
      );
  ```

- A table for basic details of shipper.

  ```
      /*  Here: shipper_id -> R is the only relevant
      dependency and hence it is in BCNF  */
  create table shipper (
    shipper_id varchar (20) primary key not null,
    name varchar (20) not null,
    head_quarters varchar (60) not null,
    phone_number decimal (10) UNSIGNED not null,
    email_id VARCHAR (20) not null
  );
  ```

- Having described these basic tables, we can now describe table for products. Note that each product can be sold by different sellers in different price and quantity, thus, primary key is formed by both product_id and seller_id.

```
/*  Here: (product_id, seller_id) -> R is the only
relevant dependency and hence it is in BCNF  */
/* Rating will be updated with the help of triggers.
*/
create table product (
  product_id varchar (20) not null,
  product_name varchar (20) not null,
  seller_id varchar (20) not null,
  price float not NULL,
  total_stock int,
  pickup_address varchar (60) not null,
  description varchar (60),
  rating float,
  foreign key (seller_id) references seller
  (seller_id) on delete cascade,
  primary key (product_id, seller_id)
);
```

- When user makes a payment, we want to store payment details for which we have the following table.

```
/*   Here: payment_id -> R is the only relevant
dependency and hence it is in BCNF  */
create table payment (
  payment_id VARCHAR (20) primary key not null,
  credit_card_number VARCHAR (20) not null,
  date_ timestamp,
  billing_address varchar(60) not null
);
```

- User will have a front end feature to add items in cart. When the user is ready to buy, it will generate an *order_id* for all those products which he or she chose. Note that *order_id* will be generated *only* when user successfully does the payment.

```
/* Here: order_id -> R is the only relevant
dependency and hence it is in BCNF  */
create table order_ (
  order_id VARCHAR (20) primary key not null,
  customer_id VARCHAR (20),
  shipping_address varchar(60) not null,
  payment_id VARCHAR (20),
  foreign key (customer_id) references customer
  (customer_id) on delete set null,
  foreign key (payment_id) references payment
  (payment_id) on delete set null
);
```

- After generating the payment, we have to put the details of the bought items along with their order_id.

```
  /*  Here: (product_id, order_id, seller_id) -> R is
  the only relevant dependency and hence it is in BCNF
  */
  create table product_order (
    product_id varchar(20) not null,
    order_id varchar (20) not null,
    seller_id varchar (20),
    product_rating int check (product_rating in (NULL,
    1, 2, 3, 4, 5)),
    seller_rating int check (seller_rating in (NULL, 1,
    2, 3, 4, 5)),
    ship_index int,
    product_review varchar (60),
    seller_review varchar (60),
    quantity int,
    selling_price float,
    primary key (product_id, order_id, seller_id),
    foreign key (product_id) references product
    (product_id) on delete cascade,
    foreign key (order_id) references order_ (order_id)
    on delete cascade,
    foreign key (seller_id) references seller
    (seller_id) on delete cascade,
    foreign key (ship_index) references track (index_)
    on delete set null
  );
```

- Note that we used a foreign key in the above table which we haven't defined yet, which is *ship_index* . It is basically a unique identifier for each ordered product serving as an index of track table which we will use to track our items.

```
/*  Here: index_ -> R is the only relevant dependency
and hence it is in BCNF  */
create table track (
  index_ INT AUTO_INCREMENT primary key not null,
  shipper_id varchar (20),
  tracking_id varchar (20),
  foreign key (shipper_id) references shipper
  (shipper_id) on delete set null
);
```

- We also have an auxiliary table for keeping track of users with their old passwords as mysql.user encrypts the passwords and there is no way to get it back also this table is required for validation when the user tries to log in the system.

```
-- Clearly this table is in BCNF.
create table Users (
  username VARCHAR (20) primary key not null,
  passcode VARCHAR (20) not null,
  roles VARCHAR (20) not null
);
```

5 Roles, Triggers, Views

5.1 Views

```
  -- #########################################
-- ##########CUSTOMER VIEWS################
-- #########################################

-- This view will allow customer to view its details.
CREATE VIEW customer_add AS (SELECT *
                             FROM customer
                             WHERE CONCAT(customer_id,
                             "@localhost") IN (SELECT user()));

-- This view will allow customer to see the total cost of
his/her various orders
CREATE VIEW orderPrice AS (SELECT order_id, sum(selling_price *
quantity) as total_price
                           FROM (product_order)
                           GROUP BY order_id);

-- This view will tell the customer details corresponding to
his/her all order_id mentioning complete order details
(order_id, shipping_address, date_, total_price) except the
products in that order.
CREATE VIEW previousOrders AS (SELECT T1.order_id,
T1.shipping_address, T2.date_, T3.total_price
                               FROM (order_ as T1) NATURAL JOIN
                               (payment as T2) NATURAL JOIN
                               (orderPrice as T3)
                               WHERE CONCAT(T1.customer_id,
                               "@localhost") IN (SELECT
                               user()));

-- This view will allow customer to just see his various
order_id.
CREATE VIEW listOrders AS (SELECT order_id
                            FROM order_
                            WHERE CONCAT(customer_id,
                            "@localhost") IN (SELECT user()));
```

```
-- This view will give entry to the track table for each
(product_id, order_id) pair
CREATE VIEW trackID AS (SELECT order_id, product_id, ship_index
                        FROM product_order
                        WHERE order_id IN (SELECT * FROM
                        listOrders));

-- This view will augment the previous view with tracking_id
as well.
CREATE VIEW packageStatus AS (SELECT T1.order_id,
T1.product_id, T1.ship_index, T2.tracking_id
                              FROM (trackID as T1) JOIN (track
                              as T2) ON (T1.ship_index =
                              T2.index_));


-- ##########################################
-- ##########SELLER VIEWS#################
-- ##########################################

-- This view will allow seller to see his/her various
products.
CREATE VIEW sellerProducts AS (SELECT product_id, product_name,
price, total_stock, pickup_address, description
                                  FROM product
                                  WHERE CONCAT(seller_id,
                                  "@localhost") in (SELECT
                                  user()));

-- This view allow seller to see various orders which he or
she have sold (seller_id, product_id, quantity,
selling_price, date_)
CREATE VIEW sellerOrders AS (SELECT T1.seller_id,
T1.product_id, T1.quantity, T1.selling_price, T2.date_
                                FROM (product_order as T1)
                                natural join (payment as T2)
                                WHERE CONCAT(T1.seller_id,
                                "@localhost") in (SELECT
                                user()));
```

```
-- ########################################
-- ##########SHIPPER VIEWS################
-- ########################################

-- This view will allow shipper details (pickup_address,
shipping_address, tracking_id)
CREATE VIEW shipperTrack AS (SELECT index_, pickup_address AS
source, shipping_address AS destination, tracking_id
                              FROM (track JOIN product_order ON
                              index_ = ship_index) NATURAL JOIN
                              order_ NATURAL JOIN product
                              WHERE CONCAT(shipper_id,
                              "@localhost") = (SELECT user()));
```

5.2 Roles

Basically we have three roles.

- A role for database administrator.
- A role for customer.
- A role for supplier.
- A role for shipper.

And their details is best understood with the help of the following code:

```
  CREATE ROLE dbadmin;
CREATE ROLE customer;
CREATE ROLE seller;
CREATE ROLE shipper;

GRANT ALL PRIVILEGES ON AmaKart.* TO dbadmin;

-- Make sure that any view on which a role gets access on
should have the filter "SELECT user()"
GRANT ALL PRIVILEGES ON AmaKart.customer_add TO customer;
GRANT SELECT ON AmaKart.previousOrders TO customer;
GRANT SELECT ON AmaKart.listOrders TO customer;
GRANT SELECT ON AmaKart.packageStatus TO customer;

GRANT SELECT ON AmaKart.sellerProducts TO seller;
GRANT SELECT ON AmaKart.sellerOrders TO seller;

GRANT SELECT ON AmaKart.shipperTrack TO shipper;
```

5.3 Triggers

```
-- When a product is sold, we want to mention its
selling_price as later the seller can update the price
DELIMITER //
CREATE TRIGGER setPrice BEFORE INSERT on product_order
FOR EACH ROW BEGIN
  SET NEW.selling_price = (SELECT price FROM product WHERE
  product_id = NEW.product_id and seller_id = NEW.seller_id);
END//
DELIMITER ;

-- When a customer passes a rating for product we have to
update it in our product table
DELIMITER //
CREATE TRIGGER updateRatingProduct AFTER UPDATE on
product_order
FOR EACH ROW BEGIN
  IF NEW.product_rating != NULL THEN
    UPDATE product SET rating = (SELECT AVG(product_rating)
    FROM product_order WHERE product_id = NEW.product_id) WHERE
    product_id = NEW.product_id;
  END IF;
END//
DELIMITER ;


-- When a customer passes a rating for seller we have to
update it in our seller table
DELIMITER //
CREATE TRIGGER updateRatingSeller AFTER UPDATE on product_order
FOR EACH ROW BEGIN
  IF NEW.seller_rating != NULL THEN
    UPDATE seller SET rating = (SELECT AVG(seller_rating) FROM
    product_order WHERE seller_id = NEW.seller_id) WHERE
    seller_id = NEW.seller_id;
  END IF;
END//
DELIMITER ;

-- When a product is sold, we need to add an entry to our
track table for the same
DELIMITER //
CREATE TRIGGER addTrack BEFORE INSERT on product_order
FOR EACH ROW BEGIN
  INSERT INTO track () Values ();
  SET NEW.ship_index = (SELECT MAX (index_) FROM track);
END//
DELIMITER ;
```

6 Functions And Procedures

Below you can see details description and implementation of various Procedures and Functions

6.1 Customer Procedures

1. Procedure to see purchases between some duration, "seePurchasesByDate(IN startTime TIMESTAMP, IN endTime TIMESTAMP)" -¿ Will return complete details from order, payment, product and product_order table.
2. Procedure to see items in cart, "getProductsFromCart ()" -¿ which selects everything from view "showCart".
3. Procedure for customer to checkout items present in cart, "purchaseEverythingInCart(IN oid varchar(20))" -¿ which takes in order id and then for each product present in cart, will add the corresponding entry to product_order table.
4. Procedure for customer to remove product from cart, "removeProductCart(IN pid varchar(20), IN sid varchar(20))" -¿ which takes in that products product_id and seller_id and thus delete such product from cart (deletion is performed using view showCart).
5. Procedure for customer to update a product in cart (i.e. change quantity of the chosen product), "CREATE PROCEDURE updateProductCart(IN pid varchar(20), IN sid varchar(20), IN N INT)".
6. Procedure for customer to make an order, "makeorder(IN cnum varchar(20), IN badd varchar(20), IN cid varchar(20), IN sadd varchar(20))" which takes a credit card number, "cnum", billing address, "badd", customer ID, "cid" and a shipping address, "sadd" and first adds an entry into payment table by selecting date using "NOW()" function and then since payment table had payment_id which was set to automatic increment

```
-- Procedure for customer to makeorder
DELIMITER //
CREATE PROCEDURE makeorder(IN cnum varchar(20), IN badd
varchar(20), IN cid varchar(20), IN sadd varchar(20))
BEGIN
```

```
    DECLARE curr_time TIMESTAMP;
    DECLARE payid INT;
    DECLARE oid INT;
    set curr_time = NOW();
    INSERT INTO
    payment(credit_card_number,date_,billing_address) values
    (cnum,curr_time,badd);
    SELECT payment_id from payment where credit_card_number =
    cnum and date_ = curr_time and billing_address = badd
    order by payment_id desc into payid;
    INSERT INTO order_
    (customer_id,payment_id,shipping_address) VALUES
    (cid,payid,sadd);
    SELECT order_id from order_ where customer_id = cid and
    payment_id = payid and shipping_address = sadd order by
    order_id desc into oid;
    call purchaseEverthingInCart(oid);
END;
//
DELIMITER ;

-- Procedure to return the total earning of a seller
between supplied dates
DELIMITER //
CREATE PROCEDURE addProductToCart(IN cid varchar(20),IN pid
varchar(20),IN sid varchar(20),IN q int)
BEGIN
    IF ((SELECT count(*) from showCart where product_id = pid
    and seller_id = sid) = 1) THEN
      UPDATE showCart set quantity = q where product_id = pid
      and seller_id = sid;
    ELSE
      INSERT INTO showCart VALUES (cid,pid,sid,q);
    END IF;
END;
//
DELIMITER ;

-- Procedure to see latest N Purchases
DELIMITER //
CREATE PROCEDURE seeLatestNPurchases(IN N INT)
```

```
BEGIN
    select * from payment natural join order_ natural join
    product_order natural join product join track on
    (product_order.ship_index = track.index_) where
    CONCAT(order_.customer_id, "@localhost") IN (SELECT
    user()) ORDER BY payment.date_ DESC LIMIT N;
END;
//
DELIMITER ;

-- Procedure to see Purchases between dates
DELIMITER //
CREATE PROCEDURE seePurchasesByDate(IN startTime TIMESTAMP,
IN endTime TIMESTAMP)
BEGIN
    select * from payment natural join order_ natural join
    product_order natural join product join track on
    (product_order.ship_index = track.index_) where
    CONCAT(order_.customer_id, "@localhost") IN (SELECT
    user()) and payment.date_ BETWEEN startTime AND endTime;
END;
//
DELIMITER ;

-- Procedure to see products within price range
DELIMITER //
CREATE PROCEDURE queryProductsTim(IN productName varchar(20),
IN lowRange FLOAT, IN highRange FLOAT)
BEGIN
    select * from product where product_name like CONCAT('%',
    productName, '%') AND price BETWEEN lowRange AND
    highRange ORDER BY price ASC;
END;
//
DELIMITER ;

-- Procedure to see reviews of a product
DELIMITER //
CREATE PROCEDURE ProductReviews(IN pid varchar(20), IN sid
varchar(20))
BEGIN
```

```
    SELECT name, product_rating, product_review,
    seller_rating, seller_review FROM (product_order natural
    join order_ natural join customer natural join payment)
    WHERE product_id = pid AND seller_id = sid;
END;
//
DELIMITER ;

-- Procedure to add review for a product
DELIMITER //
CREATE PROCEDURE addReviewProduct(IN pid varchar(20), IN oid
varchar(20), IN sid varchar(20), IN rev varchar(60))
BEGIN
    UPDATE product_order SET product_review = rev WHERE
    product_id = pid and order_id = oid and seller_id = sid;
END;
//
DELIMITER ;

-- Procedure to add review for a seller
DELIMITER //
CREATE PROCEDURE addReviewSeller(IN pid varchar(20), IN oid
varchar(20), IN sid varchar(20), IN rev varchar(60))
BEGIN
    UPDATE product_order SET seller_review = rev WHERE
    product_id = pid and order_id = oid and seller_id = sid;
END;
//
DELIMITER ;

-- Procedure to see products sorted by rating
DELIMITER //
CREATE PROCEDURE queryProductsRat(IN productName varchar(20))
BEGIN
    select * from product where product_name like CONCAT('%',
    productName, '%') ORDER BY rating DESC;
END;
//
DELIMITER ;

-- Procedure to add rating for product
```

```
DELIMITER //
CREATE PROCEDURE addRatingProduct(IN pid varchar(20), IN oid
varchar(20), IN sid varchar(20), IN rating INT)
BEGIN
    IF (rating IN (1,2,3,4,5)) THEN
      UPDATE product_order SET product_rating =  rating WHERE
      product_id = pid and order_id = oid and seller_id =
      sid;
    END IF;
END;
//
DELIMITER ;

-- Procedure to add rating for seller
DELIMITER //
CREATE PROCEDURE addRatingSeller(IN pid varchar(20), IN oid
varchar(20), IN sid varchar(20), IN rating INT)
BEGIN
    IF (rating IN (1,2,3,4,5)) THEN
      UPDATE product_order SET seller_rating = rating WHERE
      product_id = pid and order_id = oid and seller_id =
      sid;
    END IF;
END;
//
DELIMITER ;


-- Procedure to update customer info
DELIMITER //
CREATE PROCEDURE custUpdateInfo(IN customer_id varchar(20),
IN passwordd VARCHAR(20), IN named varchar(20), IN addressd
VARCHAR(60), IN phone_number DECIMAL(10) UNSIGNED, IN
email_id VARCHAR(20))
BEGIN
    IF (CHAR_LENGTH(passwordd) > 0) THEN
      UPDATE Users SET Users.passcode = passwordd WHERE
      Users.username = customer_id;
    END IF;
    IF (CHAR_LENGTH(named) > 0) THEN
```

```
      UPDATE customer SET customer.name = named WHERE
      customer.customer_id = customer_id;
    END IF;
    IF (CHAR_LENGTH(addressd) > 0) THEN
      UPDATE customer SET customer.address = addressd WHERE
      customer.customer_id = customer_id;
    END IF;
    IF (phone_number <> 0) THEN
      UPDATE customer SET customer.phone_number =
      phone_number WHERE customer.customer_id = customer_id;
    END IF;
    IF (CHAR_LENGTH(email_id) > 0) THEN
      UPDATE customer SET customer.email_id = email_id WHERE
      customer.customer_id = customer_id;
    END IF;
END;
//
DELIMITER ;

-- ##########################################
-- ###########SELLER PROCEDURES#############
-- ##########################################


-- Procedure for seller to see sold but not shipped
products

DELIMITER //
CREATE PROCEDURE soldButNotShipped(IN seller_id varchar(20))
BEGIN
  select product_order.* from product_order join track on
  product_order.ship_index = track.index_ where shipper_id is
  NULL;
  -- select * from product_order where
  product_order.seller_id = seller_id and ship_index is
  NULL;
END;
//
DELIMITER ;

-- Procedure for seller to ship a sold product
```

```
DELIMITER //
CREATE PROCEDURE shipSoldProduct(IN gproduct_id varchar(20),
IN gorder_id varchar(20), IN gseller_id varchar(20), IN
gshipper_id varchar(20), IN gtracking_id varchar(20), IN
gdate DATE)
BEGIN
  set @c = (select ship_index from product_order where
  seller_id = gseller_id and product_id = gproduct_id and
  order_id = gorder_id);
  update track set shipper_id = gshipper_id, tracking_id =
  gtracking_id, date_ = (select NOW()) where index_ = @c;
END;
//
DELIMITER ;



-- Procedure for seller to see his rating
DELIMITER //
CREATE PROCEDURE getRating(IN seller_id varchar(20))
BEGIN
  select COALESCE(rating, 0) from seller where
  seller.seller_id = seller_id;
END;
//
DELIMITER ;


-- Procedure for seller to check whether there is already a
product with given seller_id and product_id
DELIMITER //
CREATE PROCEDURE sellerCheckExistProd(IN product_id
varchar(20), IN seller_id varchar(20))
BEGIN
  select * from product where product.product_id = product_id
  and product.seller_id = seller_id;
END;
//
DELIMITER ;

-- Procedure for seller to add new product
```

```
DELIMITER //
CREATE PROCEDURE addProduct(IN product_id varchar(20), IN
seller_id varchar(20), IN product_name varchar(20), IN
product_image varchar(300), IN price float, IN total_stock
int, IN pickup_address varchar(60), IN description
varchar(60))
BEGIN
  insert into product(product_id, product_name,
  product_image, seller_id, price, total_stock,
  pickup_address, description) values (product_id,
  product_name, product_image, seller_id, price, total_stock,
  pickup_address, description);
END;
//
DELIMITER ;


-- Procedure for seller to update his specific product
details.
DELIMITER //
CREATE PROCEDURE updateProductInfo(IN product_id varchar(20),
IN seller_id varchar(20), IN product_name varchar(20), IN
product_image varchar(300), IN price float, IN total_stock
int, IN pickup_address varchar(60), IN description
varchar(60))
BEGIN
  IF (CHAR_LENGTH(product_name) > 0) THEN
    UPDATE product SET product.product_name = product_name
    where product.product_id = product_id and
    product.seller_id = seller_id;
  END IF;
  IF (CHAR_LENGTH(product_image) > 0) THEN
    UPDATE product SET product.product_image = product_image
    where product.product_id = product_id and
    product.seller_id = seller_id;
  END IF;
  IF (price > 0.1) THEN
    UPDATE product SET product.price = price where
    product.product_id = product_id and product.seller_id =
    seller_id;
  END IF;
  IF (CHAR_LENGTH(pickup_address) > 0) THEN
    UPDATE product SET product.pickup_address =
    pickup_address where product.product_id = product_id and
    product.seller_id = seller_id;
  END IF;
  IF (CHAR_LENGTH(description) > 0) THEN
    UPDATE product SET product.description = description
    where product.product_id = product_id and
    product.seller_id = seller_id;
  END IF;
END;
//
DELIMITER ;


-- Procedure to update seller's info
DELIMITER //
CREATE PROCEDURE sellerUpdateInfo(IN seller_id varchar(20),
IN passwordd VARCHAR(20), IN named varchar(20), IN addressd
VARCHAR(60), IN phone_number DECIMAL(10) UNSIGNED, IN
email_id VARCHAR(20))
BEGIN
    IF (CHAR_LENGTH(passwordd) > 0) THEN
      UPDATE Users SET Users.passcode = passwordd WHERE
      Users.username = seller_id;
    END IF;
    IF (CHAR_LENGTH(named) > 0) THEN
      UPDATE seller SET seller.name = named WHERE
      seller.seller_id = seller_id;
    END IF;
    IF (CHAR_LENGTH(addressd) > 0) THEN
      UPDATE seller SET seller.address = addressd WHERE
      seller.seller_id = seller_id;
    END IF;
    IF (phone_number <> 0) THEN
      UPDATE seller SET seller.phone_number = phone_number
      WHERE seller.seller_id = seller_id;
    END IF;
    IF (CHAR_LENGTH(email_id) > 0) THEN
      UPDATE seller SET seller.email_id = email_id WHERE
      seller.seller_id = seller_id;
    END IF;
END;
//
DELIMITER ;

-- Procedure for seller to see his or her past sold
products within a specific time duration
DELIMITER //
CREATE PROCEDURE seeSellingsBetweenDuration(IN startTime
TIMESTAMP, IN endTime TIMESTAMP)
BEGIN
    select product_order.* from payment natural join order_
    natural join product_order where
    CONCAT(product_order.seller_id, "@localhost") IN (SELECT
    user()) AND payment.date_ BETWEEN startTime AND endTime;
END;
//
DELIMITER ;

-- Procedure to see latest N Sellings
DELIMITER //
CREATE PROCEDURE seeLatestNSellings(IN N INT)
BEGIN
    select product_order.* from payment natural join order_
    natural join product_order where
    CONCAT(product_order.seller_id, "@localhost") IN (SELECT
    user()) ORDER BY payment.date_ DESC LIMIT N;
END;
//
DELIMITER ;

-- Procedure to see similary products with increasing price
DELIMITER //
CREATE PROCEDURE selQuerySimProducts(IN productName
varchar(20))
BEGIN
    select * from product where product_name like CONCAT('%',
    productName, '%') AND CONCAT(seller_id, "@localhost") IN
    (SELECT user()) ORDER BY price ASC;
END;
//
```

```
DELIMITER ;

-- Procedure to see similar products sorted by rating
DELIMITER //
CREATE PROCEDURE selQueryProductsRat(IN productName
varchar(20))
BEGIN
    select * from product where product_name like CONCAT('%',
    productName, '%') AND CONCAT(seller_id, "@localhost") IN
    (SELECT user()) ORDER BY rating DESC;
END;
//
DELIMITER ;

-- ##########################################
-- ##########SHIPPER PROCEDURES############
-- ##########################################



-- Procedure to update shipper's info
DELIMITER //
CREATE PROCEDURE shipperUpdateInfo(IN shipper_id varchar(20),
IN passwordd VARCHAR(20), IN named varchar(20), IN addressd
VARCHAR(60), IN phone_number DECIMAL(10) UNSIGNED, IN
email_id VARCHAR(20))
BEGIN
    IF (CHAR_LENGTH(passwordd) > 0) THEN
      UPDATE Users SET Users.passcode = passwordd WHERE
      Users.username = shipper_id;
    END IF;
    IF (CHAR_LENGTH(named) > 0) THEN
      UPDATE shipper SET shipper.name = named WHERE
      shipper.shipper_id = shipper_id;
    END IF;
    IF (CHAR_LENGTH(addressd) > 0) THEN
      UPDATE shipper SET shipper.head_quarters = addressd
      WHERE shipper.shipper_id = shipper_id;
    END IF;
    IF (phone_number <> 0) THEN
```

```
      UPDATE shipper SET shipper.phone_number = phone_number
      WHERE shipper.shipper_id = shipper_id;
    END IF;
    IF (CHAR_LENGTH(email_id) > 0) THEN
      UPDATE shipper SET shipper.email_id = email_id WHERE
      shipper.shipper_id = shipper_id;
    END IF;
END;
//
DELIMITER ;

-- Procedure for shipper to see his or her past shipments
within a specific time duration
DELIMITER //
CREATE PROCEDURE seeShipmentsBetweenDuration(IN startTime
DATE, IN endTime DATE)
BEGIN
    select * from track where CONCAT(shipper_id,
    "@localhost") IN (SELECT user()) AND date_ BETWEEN
    startTime AND endTime;
END;
//
DELIMITER ;

-- Procedure to see latest N Shipments
DELIMITER //
CREATE PROCEDURE seeLatestNShipments(IN N INT)
BEGIN
    select * from track where CONCAT(shipper_id,
    "@localhost") IN (SELECT user()) ORDER BY date_ DESC
    LIMIT N;
END;
//
DELIMITER ;

-- ##########################################
-- ###############FUNCTIONS################
-- ##########################################

-- Function to return the total earning of a seller between
supplied dates
```

```
DELIMITER //
CREATE FUNCTION sellerStatsBetweenDate(startTime TIMESTAMP,
endTime TIMESTAMP)
RETURNS FLOAT DETERMINISTIC
BEGIN
    DECLARE temp FLOAT;
    SELECT SUM(quantity*selling_price) INTO temp FROM
    product_order natural join order_ natural join payment
    WHERE date_ BETWEEN startTime and endTime;
    RETURN temp;
END;
//
DELIMITER ;
```

7 Use Cases

Lets for this case assume that we a supplier. We first want to register and then add products to sell.

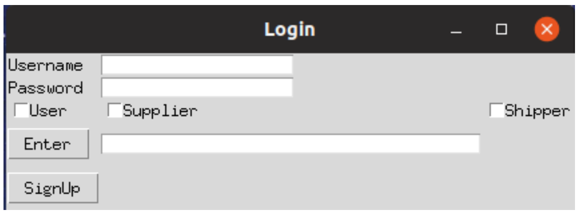


Figure 3: Login page.

So at the login page we will select SignUp. It will then open up the SignUp page.

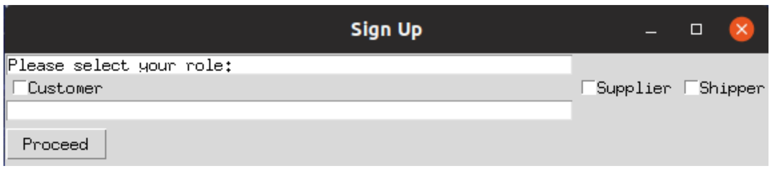
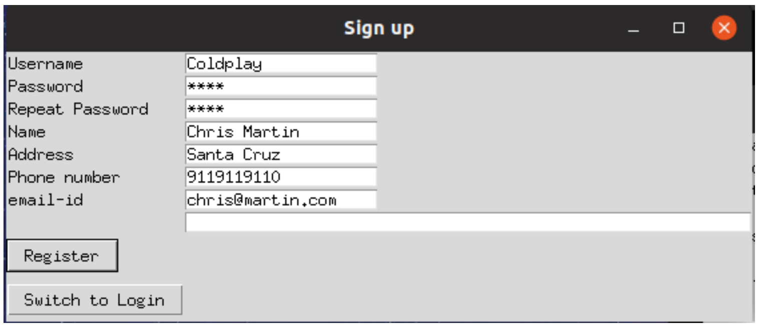

Please select your role:

Figure 4: SignUp page(Selecting role).

Since we are Supplier we will select supplier option and then proceed.

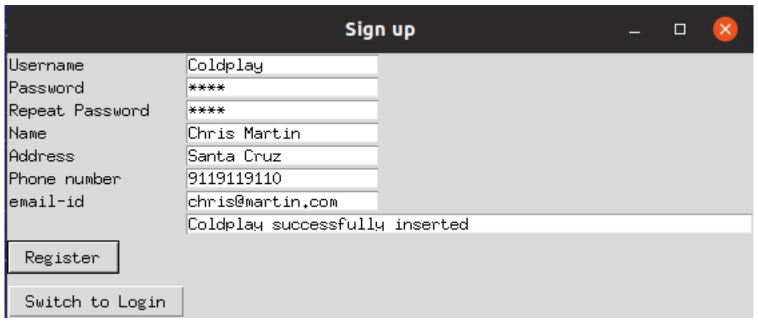


Sign up

Username	Coldplay
Password	****
Repeat Password	****
Name	Chris Martin
Address	Santa Cruz
Phone number	9119119110
email-id	chris@martin.com

Figure 5: SignUp page(Entering Details).

After entering the details press the Register button. It will display that the user is successfully created.



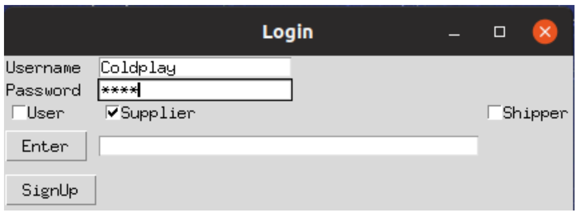
Sign up

Username	Coldplay
Password	****
Repeat Password	****
Name	Chris Martin
Address	Santa Cruz
Phone number	9119119110
email-id	chris@martin.com

Coldplay successfully inserted

Figure 6: SignUp page(Conformation).

After registering goto the login page by pressing the Switch to Login button.



Login

Username	Coldplay
Password	****

User [✓ Supplier] Shipper

Figure 7: Login page(Entering Details).

After login you will be taken to a welcome page which will have two options for you

- Either to add new products in the market.
- Or to change the quantites or price of existing products.

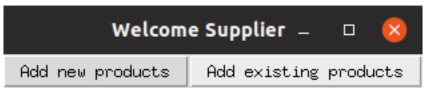
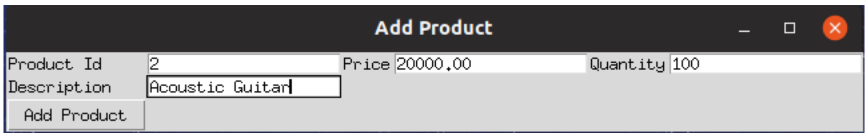


Figure 8: Welcome page for Supplier.

So lets add a new product. You will be taken to a page which will ask to fill the details of the product.



Product Id 2 Price 20000.00 Quantity 100
Description Acoustic Guitar

Figure 9: Add new product.

Now the product has been added successfully.

8 Progress And TODOS

8.1 GUI

- Signup is working perfectly (just that inplace of integer if string is given then its an issue, can write a function to check it though).
- Login is working perfectly.
-

9 Useful links

Complete project - Link
GUI source code - Link