Database Design And Implementation For E-Commerce

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

February 23, 2019

Contents

1 Contribution

As of now, we did everything together.

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. Company maintains the details of stock like their id, name, quantity, rating etc.
2. Company maintains the details of users like their id, name, address, phone number, ewallet.
3. Only users which have purchased the product can leave rating and review to product, they can also give rating to the seller. Thus users should also be able to see their past puchases.
4. Users can add balance to their ewallet.
5. Company maintains the details of its suppliers like their id, name, address, phone number and rating. Each supplier has at least some stock for some item. (Suppliers can add new product (stock) and mention its quantity which he/she has.)
6. When users browses for a product, suppliers will be listed based on the quantity user wants.

3 Entity Relation Diagram

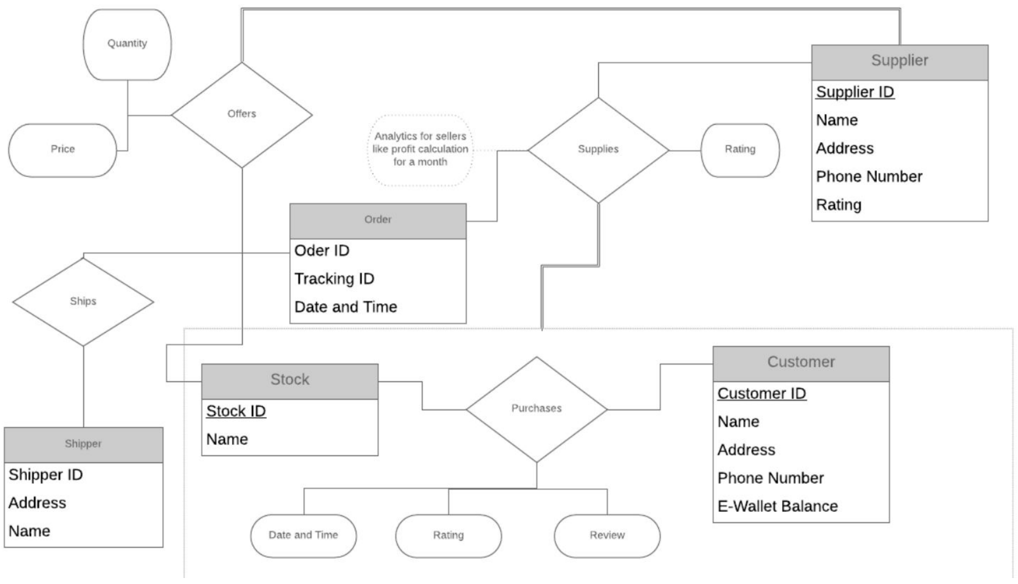


Figure 1: Entity Relationship Diagram for e-commerce

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;

-- I am not sure about these two please check
GRANT INSERT ON AmaKart.order_ TO customer;
GRANT INSERT ON AmaKart.payment 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 Use Cases

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

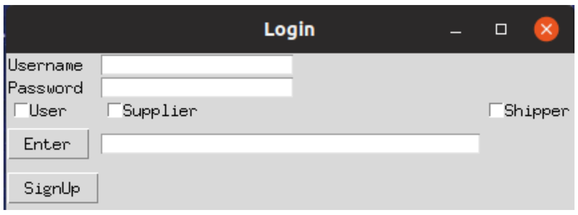


Figure 2: Login page.

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

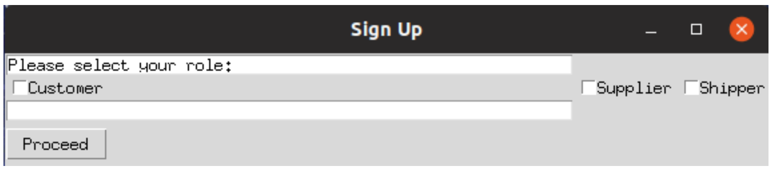
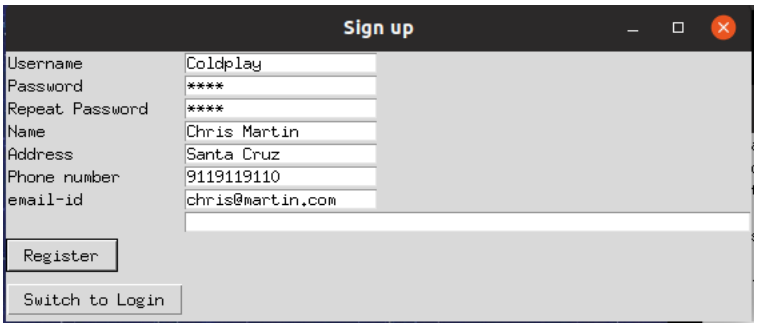


Figure 3: SignUp page(Selecting role).

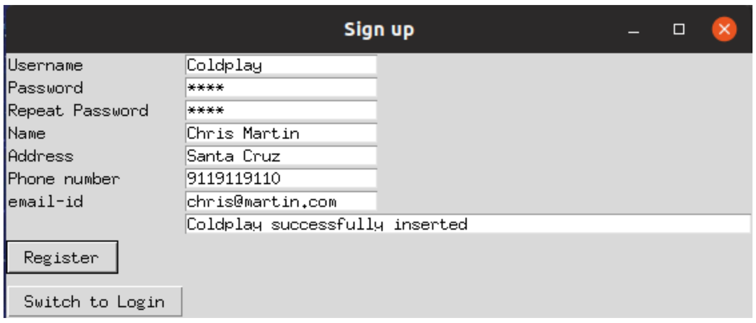
Since we are Supplier we will select supplier option and then proceed.



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

Figure 4: SignUp page(Entering Details).

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

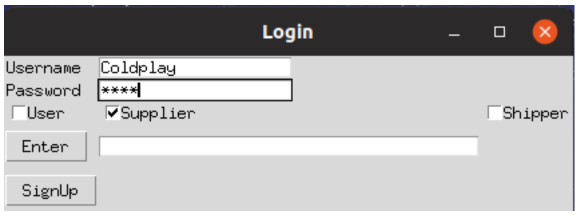


Sign up

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

Figure 5: SignUp page(Conformation).

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



Login

Username	Coldplay
Password	****

☐ User ☑ Supplier ☐ Shipper

Figure 6: 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.

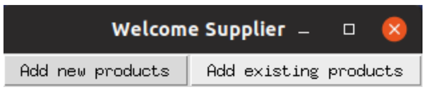
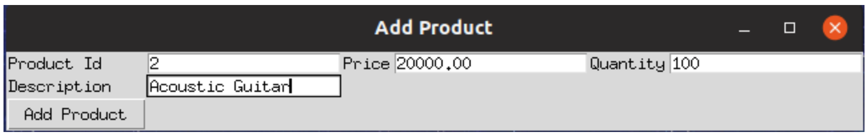


Figure 7: Welcome page for Supplier.

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



Add Product

Product Id 2 Price 20000.00 Quantity 100
Description Acoustic Guitar

Figure 8: Add new product.

Now the product has been added successfully.

7 Useful links

Complete project - Link
GUI source code - Link

8 Logs

For Developers Use only

- $11^{th} Feb 2019$ **-**
  1. Table Schema Modified.
  2. Basic Application Interface Completed.
  3. Added Use Cases in Report.
  4. And yes we are calling it AmaKart.