Eric Johnson School of Engineering and Computer Science

CS6360- Database Design

Final Project Report - Spring 2023

EaseBuy

Milind Choudhary (MXC210096),
Dipenkumar Gohil (DXG220003),
Dhruv Mittal (DXM220015),
Shrut Shah (SDS210014),
Nishtha Shukla (NXS220018)

supervised by
Dr. Jalal Omer
05/01/2023

Contents

1 Introduction

The problem statement for the project involves developing a real time application for an E-Commerce. The platform will allow the customers to register and purchase products. Additionally, the administrator will also be allowed by the platform to have access to all the relations. The application will be scalable, portable and easily maintainable. We intend to develop an online, real time, web based application for the E-Commerce platform. In the subsequent paragraphs and chapters, we will present comprehensive analysis and information about the design of the solution we propose.

The E-Commerce platform has a sign-up and login window and which can be used by the customers and administrators to provide their personal information and get a unique user id. After the user id of a customer has been generated, the platform allows the user to purchase products, manage their orders, check their personalized cart and track the orders. The platform will allow the customer to navigate to different categories of the products as listed on the website. Furthermore, the platform will place the order once the payment transaction of the customer has been verified. The user can also be registered as an administrator, who can overview all the products that has been purchased by the different customers, all the customer details and the payment verification. The administrator will also have the access to change the relations if need arises to incorporate future needs. This project is highly relevant since e-commerce heavily relies on databases for tracking transactions, managing product databases, and enabling the marketing team to track traffic, acquire potential customers, and maintain existing ones.

Section 2 provides an outline about the system requirements which are defined with help of context diagram of the database system, functional requirements, non-functional requirements and interface requirements. The following section will discuss the conceptual design of the database which includes the ER model of the database, and some business rules of the ER model. These rules are the constraints that should be followed by the platform. After this, Section 4 will talk about the logical database schema which was inferred from the ER Diagram in Section 3. The logical schema is constructed using SQL statements along with appropriate referential constraints. Section 5 describes the functional dependencies (FD) in the schema and these FDs were used as a reference for the database normalization.

Finally, Section 9 will highlight the future work that can be done and it will also contain the conclusion which our team arrived upon.

2 System Requirements

2.1 Context Diagram

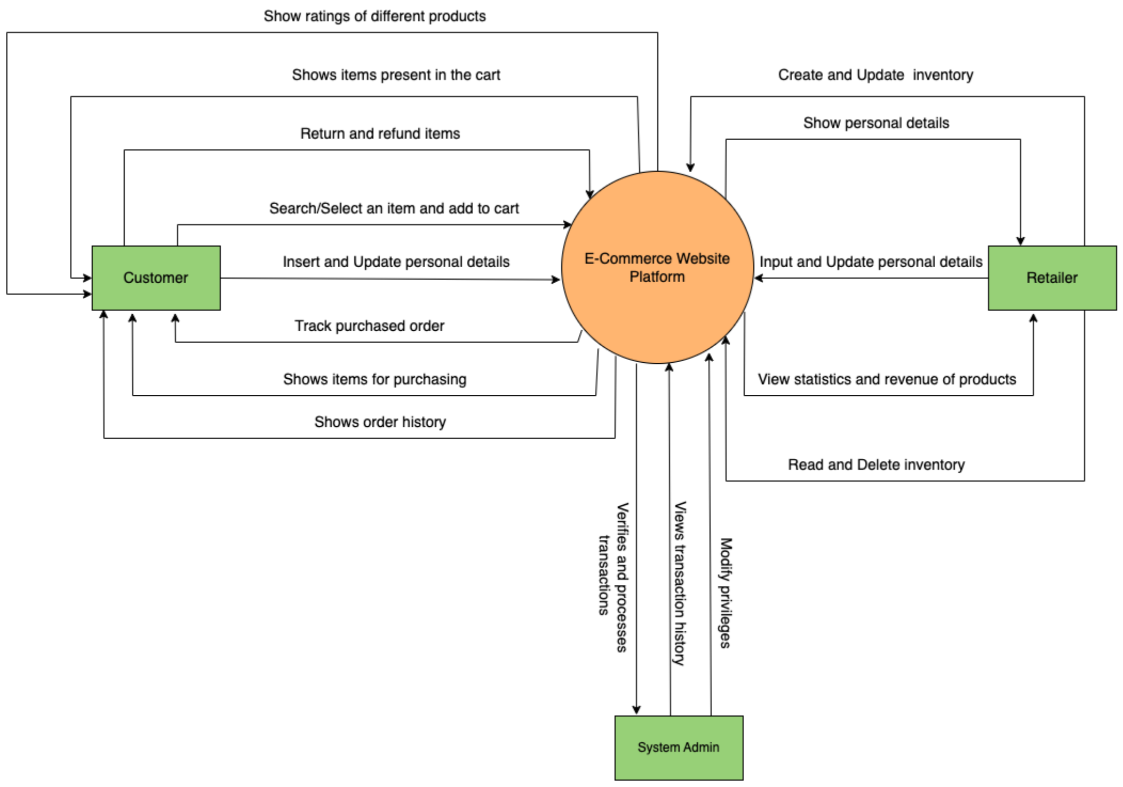

Show ratings of different products
Shows items present in the cart
Return and refund items
Search/Select an item and add to cart
Insert and Update personal details
Track purchased order
Shows items for purchasing
Shows order history
Customer
E-Commerce Website Platform
Create and Update inventory
Show personal details
Input and Update personal details
Retailer
View statistics and revenue of products
Read and Delete inventory
Verifies and processes transactions
Views transaction history
Modify privileges
System Admin

Figure 1: Context Diagram for EaseBuy

2.2 Interface Requirements

1. Customer Interface

   a) Login page will redirect to the customer interface after proper authentication.

   b) Customer Interface will contain the product categories, product details, profile and cart.

   c) This interface interacts with the payments and authorization modules respectively.

   d) Customers will also be able to provide product reviews and ratings.

2. Admin Interface

   a) Login page will redirect to the admin interface after proper authentication.

   b) Admin will have complete control over the E-Commerce platform.

   c) Admin can add different functionalities to the platform for better user experience.

d) Admin will have control over the database of the products, user details, all the product database.

e) This interface interacts with the payments and authorization modules respectively. Along with this, it validates if the payment is done accurately.

f) Admin interface can view the products purchased by the customer.

2.3 Functional and Non-Functional Requirements

2.3.1 Login Functional Requirements

- The system will allow the user to log in.
- The system will verify the username and password.
- The system will not allow the user to log in with an invalid username or password.
- The system will be able to remember usernames and passwords.
- The system will allow users to create accounts.
- The system will enable users to log out of their account.
- The system will allow the user to reset password.

2.3.2 Browsing Functional Requirements

- View products and their details.
- View the different categories of products.
- Review products and add items to the cart.
- After logging in the customer should be able to view special deals.
- View their purchase history.
- View feedback, reviews and ratings given by previous customers.

2.3.3 Customer Functional Requirements

- Insert and update the personal details.
- Selects a product for purchasing.

- Make payments after purchasing a product.
- The system will allow customers to review the products.
- Cancel orders.

2.3.4 Administrator Functional Requirements

- The system will allow administrators to view every product.
- The system will allow administrators to add, update, delete and modify the records of products.
- The system will allow administrators to add, update, delete and modify the details of the customer.
- The system will allow administrators to add, update, delete and modify columns and tables.
- The system will allow administrators to grant permissions to users.
- Administrators will monitor the efficiency of operations and the system.
- Administrators will make a security backup of the records.
- Administrators will acquire and monitor software and hardware resources.
- The system will not allow administrators to view passwords of the users.
- The system will allow administrators to process and verify the payment transactions.

2.3.5 Non-Functional Requirements

1. **Usability**
    - How quickly users can complete their task on a single page.
    - The speed at which people complete activities in the store.
    - The speed at which people complete activities in the store.
    - Frequency and duration of user errors.
2. **Security**
    - Account authorization: A customer can view only his/her orders and cannot have access to other customers orders.

- The user has to create a strong password (which contains numbers and letters) for their account.
- The user will have to choose at least one security question to help verify the user's identity during a login attempt.
- The website should ensure secure transactions for the customer.

3. **Performance**
   - The website should load fast irrespective of the traffic.
4. **Maintainability**
   - It is necessary to update and maintain the database regularly with respect to the load on the platform.
   - As updates are done and the website grows, the ecommerce platform should remove all the back end complexities and be able to make changes to the system in future.
5. **Scalability**
   - The website's performance won't be impacted as it develops and adds functionality.
   - To handle additional transactions on the website, it needs to be possible to add extra memory space, servers, or disc space
6. **Compatibility**
   - The application should be compatible with multiple platforms and hardware.
7. **Data Integrity**
   - Manipulation, retrieval, storing of data by the application should be accurate and consistent.
8. **Compliances**
   - The E-Commerce website should follow the Payment Card Industry Data Security Standard (PCI DSS) when handling credit card data.
   - The E-Commerce website should be GDPR compliant.
   - The E-Commerce website should be compliant to the shipping compliance.

3 Conceptual Design of Database

3.1 Entity-Relationship Model

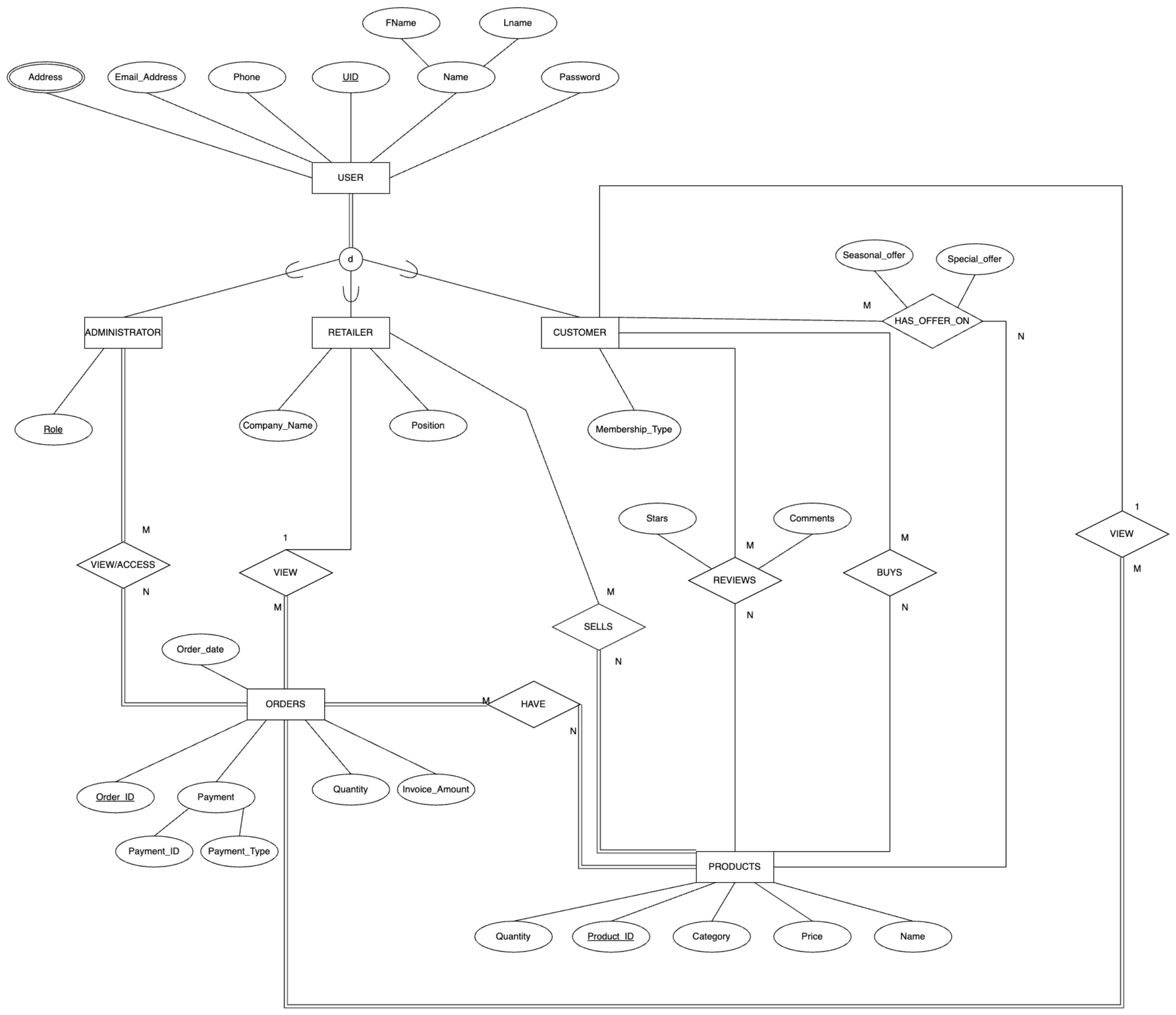


Figure 2: ER Diagram for EaseBuy

3.2 Business Rules and Integrity Constraints

- Neither the admin nor the customer can have the same email.
- A user's email may not be null.
- An administrator must have a valid role (cannot be null).
- An order must contain one or more products.
- Customers can only buy available products.

- A payment may not be more than the cost of the order.
- Customers can only give one review per product.
- One order can only have one payment type.
- A user can only either be admin or customer.
- A customer can only purchase a maximum of 7 items of a particular product in a single order.

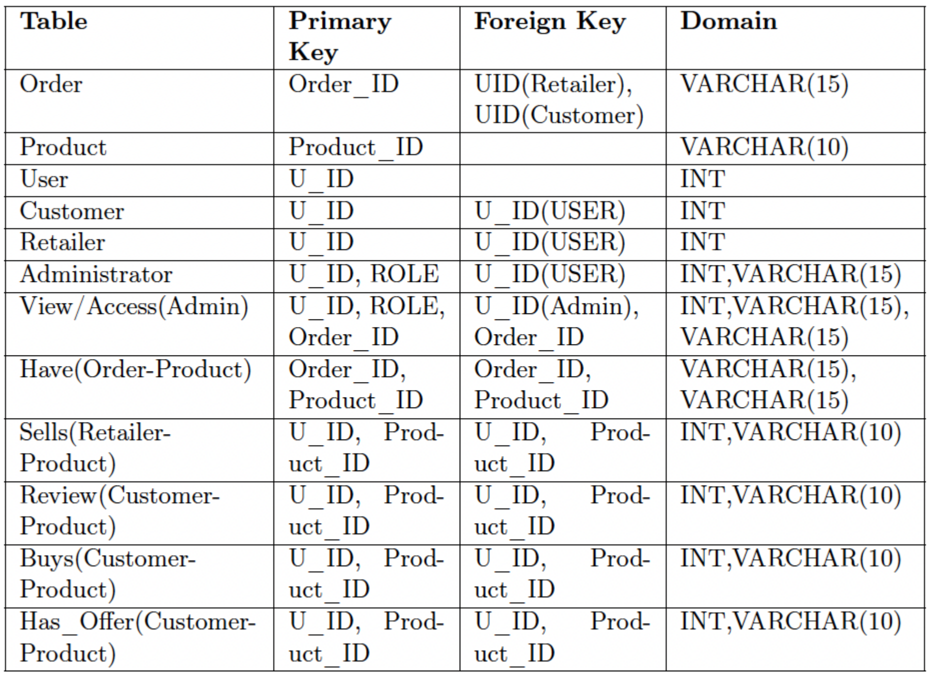


Table	**Primary Key**	**Foreign Key**	**Domain**
Order	Order_ID	UID(Retailer), UID(Customer)	VARCHAR(15)
Product	Product_ID		VARCHAR(10)
User	U_ID		INT
Customer	U_ID	U_ID(USER)	INT
Retailer	U_ID	U_ID(USER)	INT
Administrator	U_ID, ROLE	U_ID(USER)	INT,VARCHAR(15)
View/Access(Admin)	U_ID, ROLE, Order_ID	U_ID(Admin), Order_ID	INT,VARCHAR(15), VARCHAR(15)
Have(Order-Product)	Order_ID, Product_ID	Order_ID, Product_ID	VARCHAR(15), VARCHAR(15)
Sells(Retailer-Product)	U_ID, Product_ID	U_ID, Product_ID	INT,VARCHAR(10)
Review(Customer-Product)	U_ID, Product_ID	U_ID, Product_ID	INT,VARCHAR(10)
Buys(Customer-Product)	U_ID, Product_ID	U_ID, Product_ID	INT,VARCHAR(10)
Has_Offer(Customer-Product)	U_ID, Product_ID	U_ID, Product_ID	INT,VARCHAR(10)

Figure 3: Integrity Constraints for EaseBuy

4 Logical Database Schema

4.1 Relational Database Schema

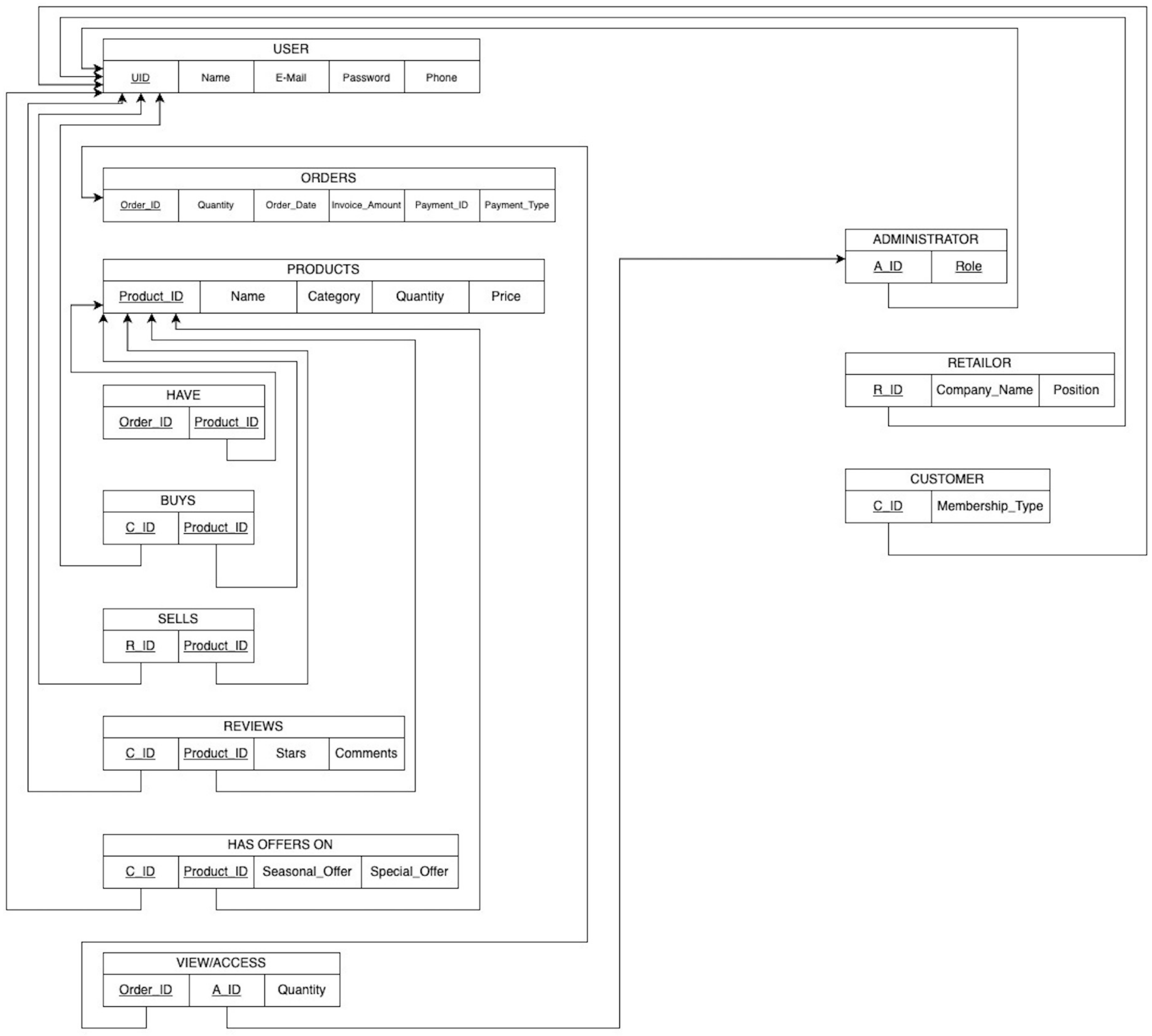


Figure 4: Relational Database Schema for EaseBuy

4.2 DB Construction SQL Statements

1. **Orders**

```
    create table ORDERS (
    Order_ID varchar(15) not null,
    Quantity int not null default 1,
    Order_Date date not null,
    Invoice_Amount int not null,
    R_ID varchar(15) not null,
    C_ID varchar(15) not null,
    Payment_ID varchar(15) not null unique,
    constraint ORDERS_PK primary key (Order_ID)
);
```

2. **Payment**

```
create table Payment (
    Payment_ID varchar(15) not null,
    Payment_Type varchar(5) not null,
    constraint Payment_PK primary key (Payment_ID),
    constraint Payment_ORDERS_FK foreign key (Payment_ID) references ORDERS (Payment_ID)
);
```

3. **Users**

```
create table USERS (
U_ID varchar(15) not null,
    First_Name varchar(15) not null,
    Last_Name varchar(15) not null,
    Email varchar(20) not null unique,
    Phone varchar(10) not null,
    constraint USERS_PK primary key (U_ID)
);
```

4. **Administrator**

```
create table Administrator (
    A_ID varchar(15) not null,
    Role varchar(50) not null,
    constraint ADMIN_PK primary key (A_ID, Role),
    constraint ADMIN_USERS_FK foreign key (A_ID) references USERS (U_ID)
);
```

5. **Customer**

```
create table Customer (
    C_ID varchar(15) not null,
    Membership_Type varchar(10) not null,
    constraint CST_PK primary key (C_ID),
    constraint CST_USERS_FK foreign key (C_ID) references USERS (U_ID)
);
```

6. **User_Address**

```
create table User_Address (
    U_ID varchar(15) not null,
    Address varchar(50) not null,
    constraint USRADDR_PK primary key (U_ID, Address),
    constraint USRADDR_USERS_FK foreign key (U_ID) references USERS (U_ID)
);
```

7. **User_Email**

```
create table User_Email (
    Email varchar(20) not null,
    Password varchar(20) not null,
    constraint USREML_PK primary key (Email),
    constraint USREML_USERS_FK foreign key (Email) references USERS (Email)
);
```

8. **Products**

```
create table PRODUCTS (
Product_ID varchar(15) not null,
    Name varchar(15) not null unique,
    Quantity int not null,
    Price int not null,
    constraint PRODUCTS_PK primary key (Product_ID)
);
```

9. **Prod_Name**

```
create table Prod_Name (
    Name varchar(20) not null,
    Category varchar(20) not null,
    constraint PRODNM_PK primary key (Name),
    constraint PRODNM_PRODUCTS_FK foreign key (Name) references PRODUCTS (Name));
```

10. **Reviews**

```
create table REVIEWS (
C_ID varchar(15) not null,
Product_ID varchar(15) not null,
    Comments varchar(100) not null,
    constraint REVIEWS_PK primary key (C_ID, Product_ID)
);
```

11. **Prod_Star**

```
create table Prod_Stars (
    Product_ID varchar(15) not null,
    Stars varchar(1) not null,
    constraint PRODSTR_PK primary key (Product_ID)
    #constraint PRODSTR_REVIEWS_FK foreign key (Product_ID) references PRODUCTS (Product_
);
```

12. **Offers**

```
create table OFFERS (
C_ID varchar(15) not null,
Product_ID varchar(15) not null,
    Special_Offer varchar(20) not null,
    constraint OFFERS_PK primary key (C_ID, Product_ID)
);
```

13. **Seasonal_Offers**

```
create table Seasonal_Offers (
    Product_ID varchar(15) not null,
    Seasonal_Offer varchar(20) not null,
    constraint SESONOFR_PK primary key (Product_ID)
    #constraint PRODSTR_REVIEWS_FK foreign key (Product_ID) references PRODUCTS (Product_
);
```

14. **Ord_have_Prod**

```
create table ORD_have_PROD (
Order_ID varchar(15) not null,
    Product_ID varchar(15) not null,
    constraint ORDhavePROD_PK primary key (Order_ID, Product_ID),
    constraint ORDhavePROD_ORD_FK foreign key (Order_ID) references ORDERS (Order_ID),
    constraint ORDhavePROD_PROD_FK foreign key (Product_ID) references PRODUCTS (Product_
);
```

15. **Cust_buys_Prod**

```
create table CST_buys_PROD (
C_ID varchar(15) not null,
    Product_ID varchar(15) not null,
    constraint CSTbuyPROD_PK primary key (C_ID, Product_ID),
    constraint CSTbuyPROD_CST_FK foreign key (C_ID) references CUSTOMER (C_ID),
    constraint CSTbuyPROD_PROD_FK foreign key (Product_ID) references PRODUCTS (Product_
);
```

16. **AdminviewOrder**

```
create table ADMINviewORDER (
A_ID varchar(15) not null,
    Order_ID varchar(15) not null,
    constraint ADMINviewORD_PK primary key (A_ID, Order_ID),
    constraint ADMINviewORD_ADMIN_FK foreign key (A_ID) references ADMINISTRATOR (A_ID),
    constraint ADMINviewORD_ORDER_FK foreign key (Order_ID) references ORDERS (Order_ID)
);
```

4.3 Expected Data Volumes

We calculate the expected data volumes by taking 1byte for a character and then multiply it with future scaling.

1. **Admin:** 15 + 15 = 30bytes/record + 2bytes(column) = 32bytes/record.
   Initially we have 4x32 = 128 and in future we expect it to be 10x32=320bytes.

2. **CustBuysProd:** 15 + 15 = 30bytes/record + 2bytes(column) = 32bytes/record.
   Initially we have 5x2x32 = 320 and in future we expect it to be 10x4x32=1280bytes. This is assuming average customer buys 2 product and in future they will buy 4 products.

3. **Customer:** 15 + 10 = 25bytes/record + 2bytes(column) = 27bytes/record.
   Initially we have 27x5 = 135 and in future we expect it to be 10x27 = 270bytes.

4. **Offers:** 15 + 15 + 20 = 50bytes/record + 3bytes(column) = 53bytes/record.
   Initially we have 5x2x53 = 530bytes/record and in future we expect it to be 5x10x53 =

2650bytes.

5. **Orders:** 15 + 15 + 15 + 15 = 60bytes/record + 7bytes(column) = 67bytes/record.

   Initially we have 6x67 = 402bytes/record and in future we expect it to be 10x67 = 770bytes.

6. **Order_have_prod:** 15 + 15 = 30bytes/record + 2bytes(column) = 32bytes/record.

   Initially we have 1x10x32 = 320bytes/record and in future we expect it to be 2x30x32 = 1920bytes.

7. **Payment:** 15 + 5 = 20bytes/record + 2bytes(column) = 22bytes/record.

   Initially we have 6x22 = 132bytes/record and in future we expect it to be 10x22 = 220bytes.

8. **Prod_Name:** 20 + 20 = 40bytes/record + 2bytes(column) = 42bytes/record.

   Initially we have 9x42 = 378bytes/record and in future we expect it to be 20x42 = 840bytes.

9. **Prod_star:** 15 + 1 = 16bytes/record + 2bytes(column) = 18bytes/record.

   Initially we have 9x18 = 162bytes/record and in future we expect it to be 20x18 = 360bytes.

10. **Products:** 15 + 15 = 30bytes/record + 4bytes(column) = 34bytes/record.

    Initially we have 9x34 = 306bytes/record and in future we expect it to be 20x34 = 680bytes.

11. **Reviews:** 15 + 15 + 100 = 130bytes/record + 3bytes(column) = 133bytes/record.

    Initially we have 5x2x133 = 1330bytes/record and in future we expect it to be 10x4x133 = 5320bytes.

12. **Rtl_Sell_Prod:** 15 + 15 = 30bytes/record + 2bytes(column) = 32bytes/record.

    Initially we have 5x2x32 = 320bytes/record and in future we expect it to be 10x4x32 = 1280bytes.

13. **Seasonal_Offer:** 15 + 20 = 35bytes/record + 2bytes(column) = 37bytes/record.

    Initially we have 9x1x37 = 333bytes/record and in future we expect it to be 20x2x27 = 1480bytes.

14. **User_Address:** 15 + 50 = 65bytes/record + 2bytes(column) = 67bytes/record.

    Initially we have 14x67 = 938bytes/record and in future we expect it to be 30x67 = 2010bytes.

15. **User_Email:** 20 + 20 = 40bytes/record + 2bytes(column) = 42bytes/record.

    Initially we have 14x42 = 588bytes/record and in future we expect it to be 30x42 =

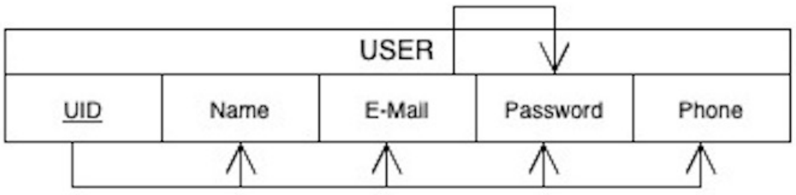
1260bytes.

16. **Users:** 15 + 15 + 15 + 20 + 10 = 75bytes/record + 5bytes(column) = 80bytes/record. Initially we have = 14x80 = 1120bytes/record and in future we expect it to be 30x80 = 2400bytes.

    The total estimated data volume is: 23640 bytes

5 Functional Dependencies and Data Normalization

1. **Normalizing User Info:**



The user table already satisfies 1NF (no multivalued attributes) and 2NF (no partial dependency) conditions. 3NF not satisfied as there is a transitive dependency.

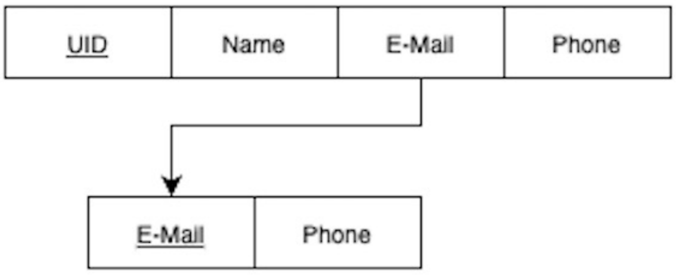
User (UID, Fname, Lname, Email, Password, Phone)

FDs:

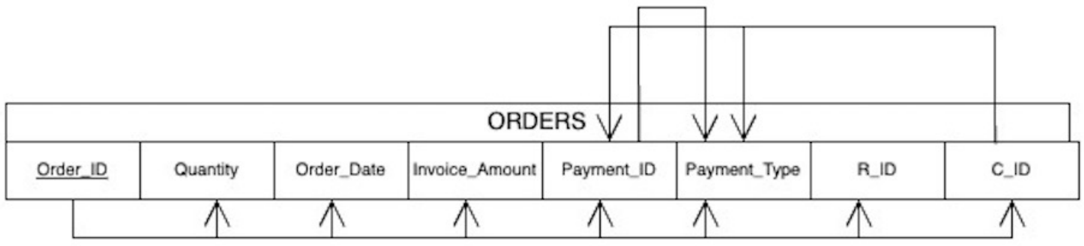
F1 – UID-> Name, Email, Password, Phone

F2 – Email-> Password

Now in 3NF



2. **Normalizing Orders:**



The order table already satisfies 1NF (no multivalued attributes) and 2NF (no partial dependency) conditions. 3NF not satisfied as there is a transitive dependency.

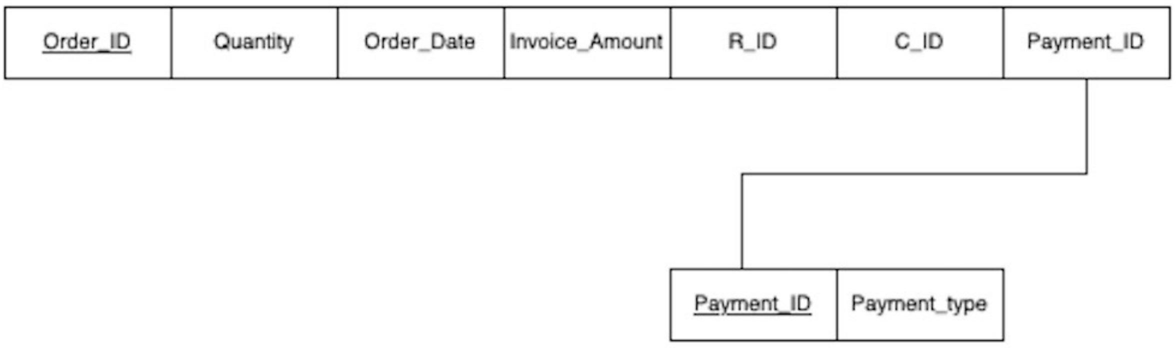
Orders (Order ID, Quantity, order_date, invoice_amount, paymentID, payment_type, R_ID, C_ID)

FDs:

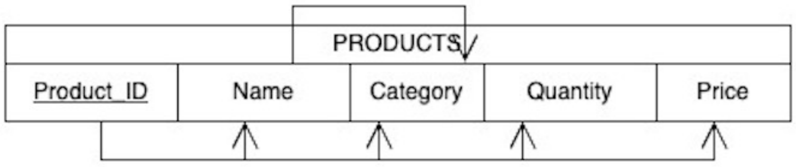
F1 – OrderID-> quantity, order_date, invoice_amount, payment_id, payment_type, R_ID, C_ID

F2 – Payment_ID->Payment_type

Now in 3NF



3. **Normalizing Products:**



The products table is already satisfies 1NF (no multivalued attributes) and 2NF (no partial dependency) conditions. 3NF not satisfied as there is a transitive dependency.

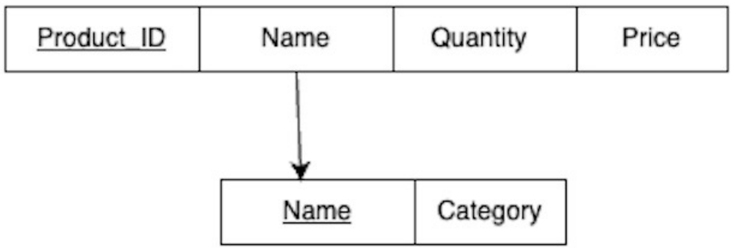
Products (Product_ID, Name, Category, Qunatity, Price)

FDs:

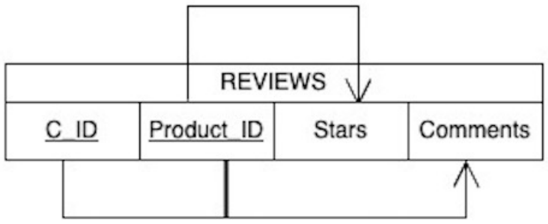
F1 – Product_ID-> Name, Category, Quantity, Price

F2 – Name->Category

Now in 3NF



4. **Normalizing Reviews:**



The reviews table is already satisfies 1NF (no multivalued attributes) and 2NF (no partial dependency) conditions. 3NF not satisfied as there is a transitive dependency.

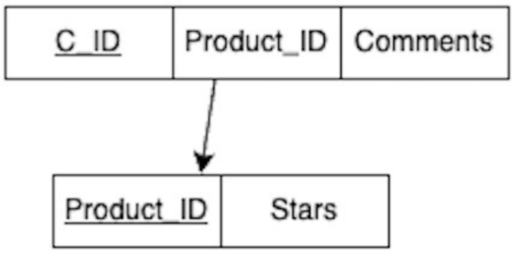
Reviews (C_ID, Product_ID, Stars, Comments)

FDs:

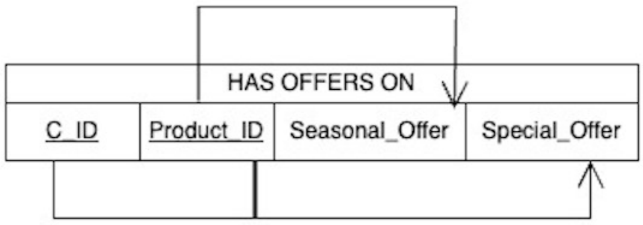
F1 – C_ID, Product_ID-> Comments

F2 – Product_ID->Stars

Now in 2NF and 3NF



5. **Normalizing Has Offers:**



The products table is already satisfies 1NF (no multivalued attributes).2NF and 3NF not satisfied as there is a partial dependency and transitive dependency respectively.

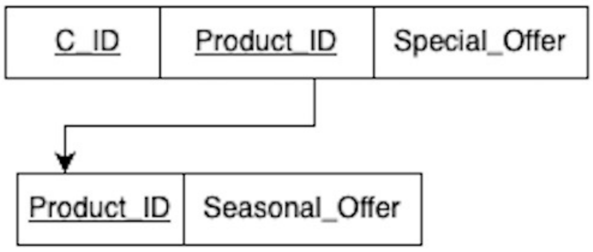
HasOffersOn (C_ID, Product_ID, Seasonal_offer, Special_offer)

FDs:

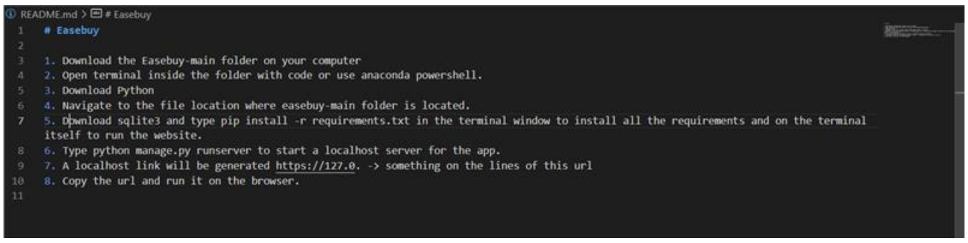
F1 – C_ID, Product_ID-> Special_offer

F2 – Product_ID->Seasonal_offer

Now in 2NF and 3NF

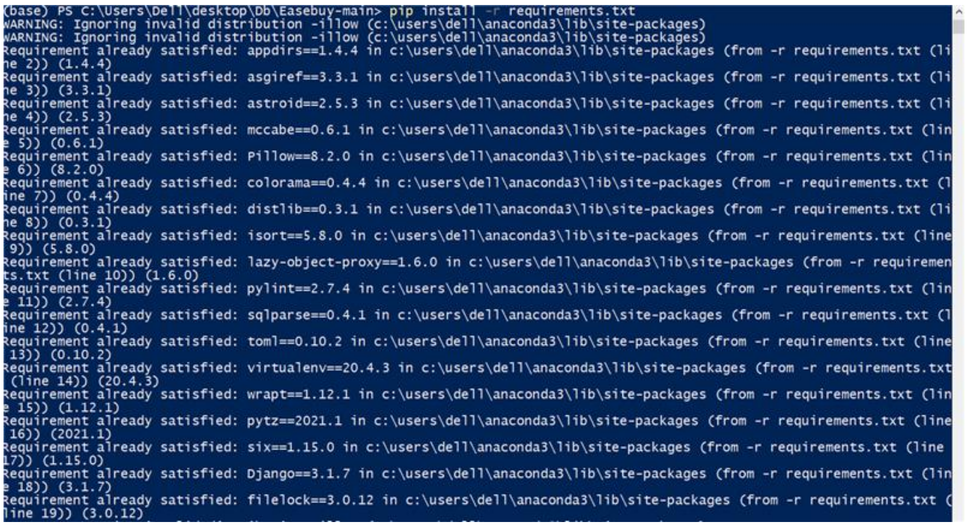


6 Database System



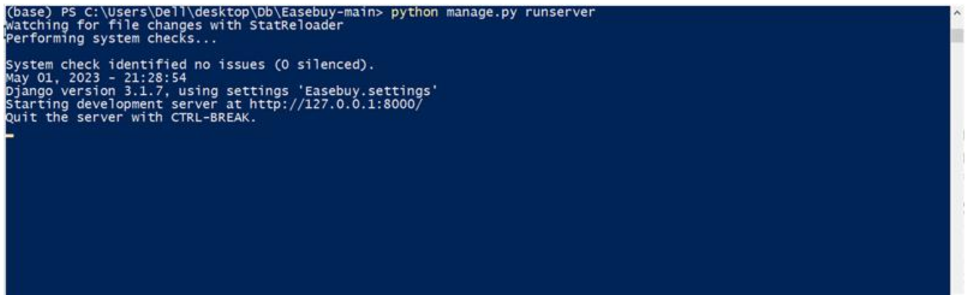
```
README.md > # Easebuy
# Easebuy

1. Download the Easebuy-main folder on your computer
2. Open terminal inside the folder with code or use anaconda powershell.
3. Download Python
4. Navigate to the file location where easebuy-main folder is located.
5. Download sqlite3 and type pip install -r requirements.txt in the terminal window to install all the requirements and on the terminal
itself to run the website.
6. Type python manage.py runserver to start a localhost server for the app.
7. A localhost link will be generated https://127.0. -> something on the lines of this url
8. Copy the url and run it on the browser.
```

Figure 5: How to invoke and install the System



```
(base) PS C:\Users\Dell\desktop\Db\Easebuy-main> pip install -r requirements.txt
WARNING: Ignoring invalid distribution -illow (c:\users\dell\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -illow (c:\users\dell\anaconda3\lib\site-packages)
Requirement already satisfied: appdirs==1.4.4 in c:\users\dell\anaconda3\lib\site-packages (from -r requirements.txt (li
ne 2)) (1.4.4)
Requirement already satisfied: asgiref==3.3.1 in c:\users\dell\anaconda3\lib\site-packages (from -r requirements.txt (li
ne 3)) (3.3.1)
Requirement already satisfied: astroid==2.5.3 in c:\users\dell\anaconda3\lib\site-packages (from -r requirements.txt (li
ne 4)) (2.5.3)
Requirement already satisfied: mccabe==0.6.1 in c:\users\dell\anaconda3\lib\site-packages (from -r requirements.txt (lin
e 5)) (0.6.1)
Requirement already satisfied: Pillow==8.2.0 in c:\users\dell\anaconda3\lib\site-packages (from -r requirements.txt (lin
e 6)) (8.2.0)
Requirement already satisfied: colorama==0.4.4 in c:\users\dell\anaconda3\lib\site-packages (from -r requirements.txt (l
ine 7)) (0.4.4)
Requirement already satisfied: distlib==0.3.1 in c:\users\dell\anaconda3\lib\site-packages (from -r requirements.txt (li
ne 8)) (0.3.1)
Requirement already satisfied: isort==5.8.0 in c:\users\dell\anaconda3\lib\site-packages (from -r requirements.txt (line
 9)) (5.8.0)
Requirement already satisfied: lazy-object-proxy==1.6.0 in c:\users\dell\anaconda3\lib\site-packages (from -r requiremen
ts.txt (line 10)) (1.6.0)
Requirement already satisfied: pylint==2.7.4 in c:\users\dell\anaconda3\lib\site-packages (from -r requirements.txt (lin
e 11)) (2.7.4)
Requirement already satisfied: sqlparse==0.4.1 in c:\users\dell\anaconda3\lib\site-packages (from -r requirements.txt (l
ine 12)) (0.4.1)
Requirement already satisfied: toml==0.10.2 in c:\users\dell\anaconda3\lib\site-packages (from -r requirements.txt (line
 13)) (0.10.2)
Requirement already satisfied: virtualenv==20.4.3 in c:\users\dell\anaconda3\lib\site-packages (from -r requirements.txt
 (line 14)) (20.4.3)
Requirement already satisfied: wrapt==1.12.1 in c:\users\dell\anaconda3\lib\site-packages (from -r requirements.txt (lin
e 15)) (1.12.1)
Requirement already satisfied: pytz==2021.1 in c:\users\dell\anaconda3\lib\site-packages (from -r requirements.txt (line
 16)) (2021.1)
Requirement already satisfied: six==1.15.0 in c:\users\dell\anaconda3\lib\site-packages (from -r requirements.txt (line
17)) (1.15.0)
Requirement already satisfied: Django==3.1.7 in c:\users\dell\anaconda3\lib\site-packages (from -r requirements.txt (lin
e 18)) (3.1.7)
Requirement already satisfied: filelock==3.0.12 in c:\users\dell\anaconda3\lib\site-packages (from -r requirements.txt (
line 19)) (3.0.12)
```

Figure 6: Installing Python Dependencies (Screen Dump)



```
(base) PS C:\Users\Dell\desktop\Db\Easebuy-main> python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
May 01, 2023 - 21:28:54
Django version 3.1.7, using settings 'Easebuy.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

Figure 7: Running Server (Screen Dump)

7 Additional Views and Queries

7.1 Views

1. The view **MaskData_CUSTOMERS** will show only a subset of fields from the table store_customer.

```
CREATE VIEW MaskData_CUSTOMERS AS
SELECT id, first_name, last_name, email, phone
FROM store_customer;
```

2. The view **MaskOrder_ORDERS** will show only a subset of fields from the table store_order to protect customer privacy.

```
CREATE VIEW MaskOrder_ORDERS AS
SELECT id, date, quantity, price, customer_id, product_id, status
FROM store_order;
```

3. The view **MaskProducts_PRODUCTS** will show only a subset of fields from the table store_products.

```
CREATE VIEW MaskProducts_PRODUCTS AS
SELECT id, name, price, image
FROM store_products;
```

7.2 Additional SQL Statements (Triggers)

1. The trigger **checkQuantity_Products** will prevent from inserting items with total count more than 15,000.

```
delimiter //
CREATE TRIGGER checkQuantity_Products BEFORE INSERT ON PRODUCTS
FOR EACH ROW
IF NEW.Quantity > 15000 THEN
SIGNAL SQLSTATE '50001' SET MESSAGE_TEXT = 'Quantity must be less than 15000.';
END IF;//
delimiter;
```

2. The trigger **checkQuantity_Orders** will prevent from purchasing items with total count more than 60.

```
delimiter //
CREATE TRIGGER checkQuantity_Orders BEFORE INSERT ON ORDERS
FOR EACH ROW
IF NEW.Quantity > 60 THEN
SIGNAL SQLSTATE '50001' SET MESSAGE_TEXT = 'Quantity must be less than 60.';
END IF;//
delimiter ;
```

3. The trigger **checkStars_ProdStars** will restrict the user to put stars in the range of 1 to 5 stars.

```
delimiter //
CREATE TRIGGER checkStars_ProdStars BEFORE INSERT ON Prod_Stars
FOR EACH ROW
IF NEW.Stars > 5 or NEW.Stars < 0 THEN
SIGNAL SQLSTATE '50001' SET MESSAGE_TEXT = 'Stars must be within 0 and 5.';
END IF;//
delimiter ;
```

8 User Application Interface

8.1 Signup/Login

We have a "store_customer" table that has attributes "id", "phone", "email", "password", "last_name", "first_name".
Upon Signup a new record is inserted in the table using the INSERT statement.
Upon login the function fetches the details of the customer based on the email using select statement and then compares the entered password with the fetched password.

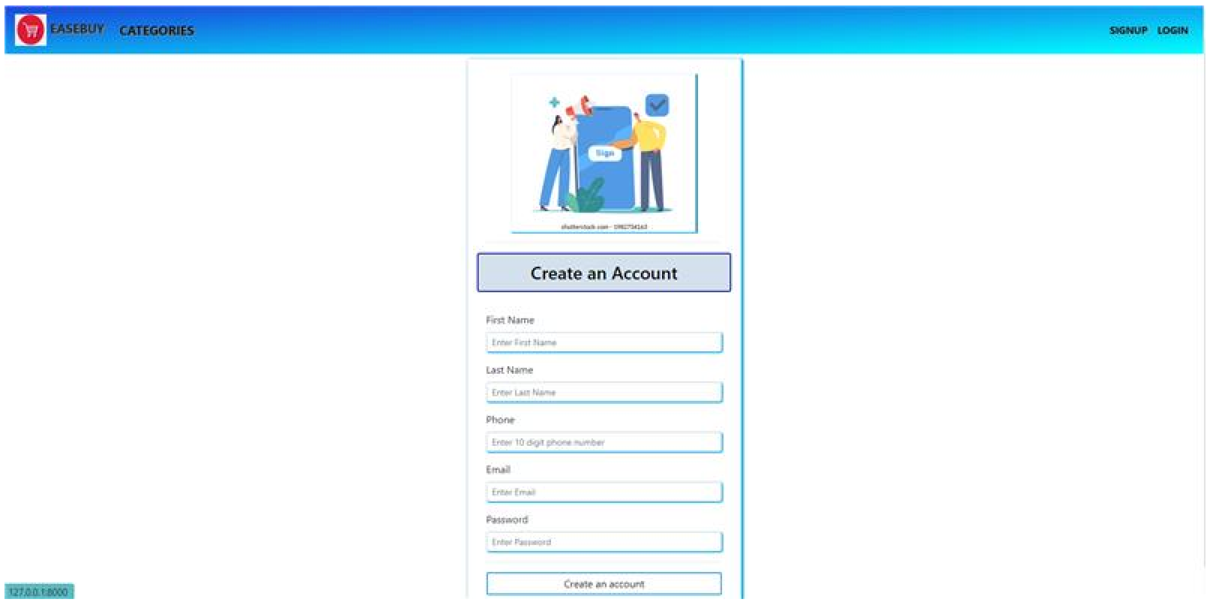


Figure 8: Signup Page for EaseBuy

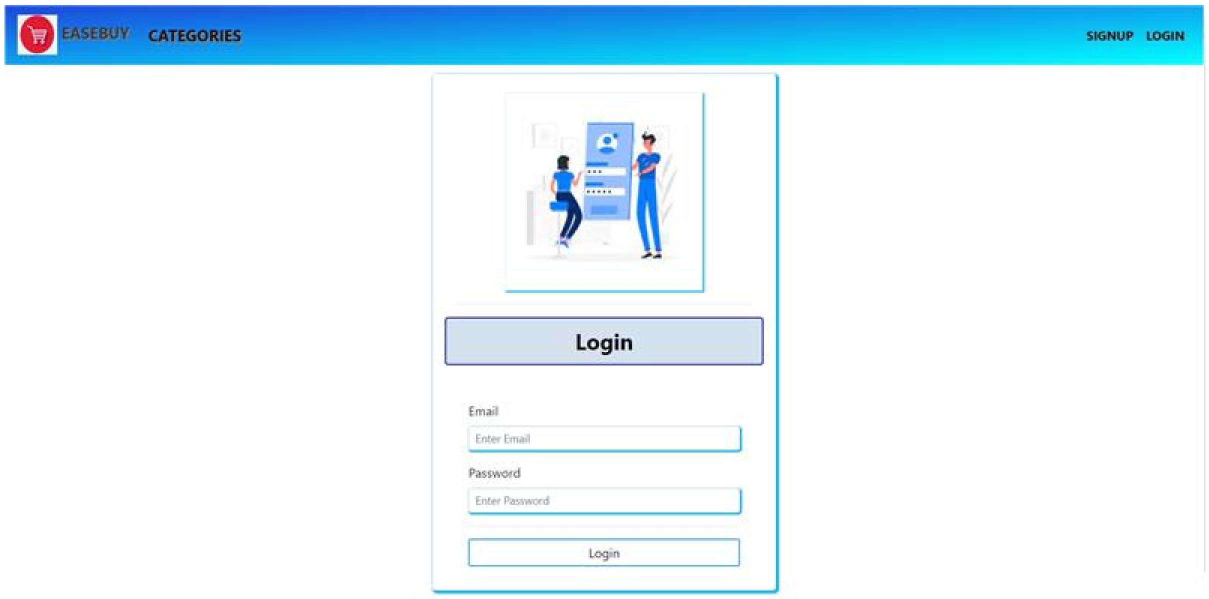


Figure 9: Login Page for EaseBuy

8.2 Homepage

This function uses select query to fetch all products and their details from the "store_products" table.

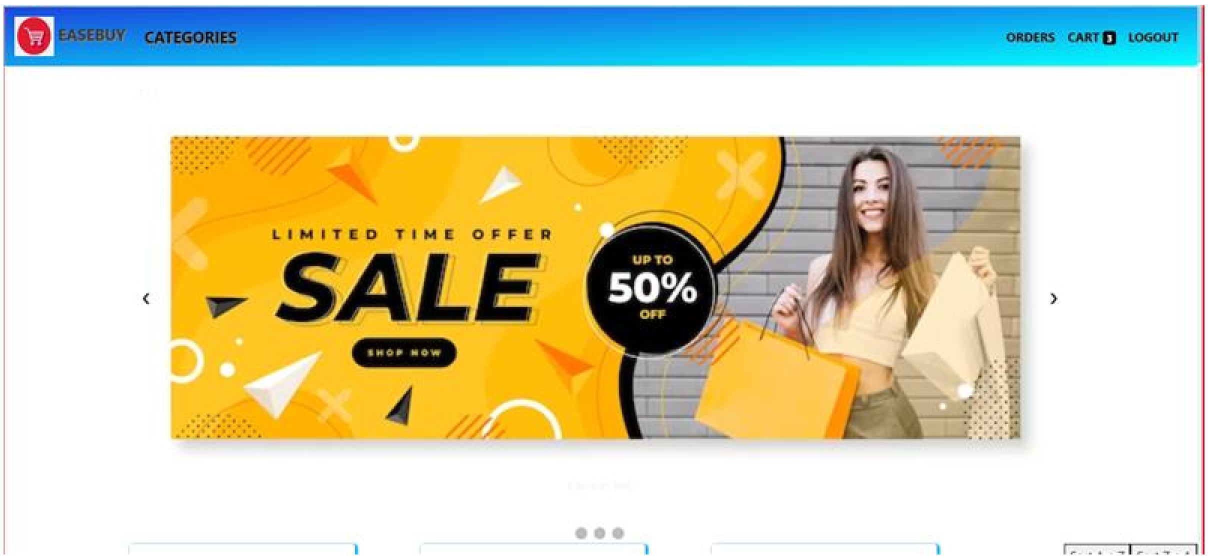


Figure 10: Home Page for EaseBuy

8.3 Categories

For each category we have created a view that will contain products of that particular category. Hence, Upon selecting any one of the categories, it will fetch and only display all the products of that category.

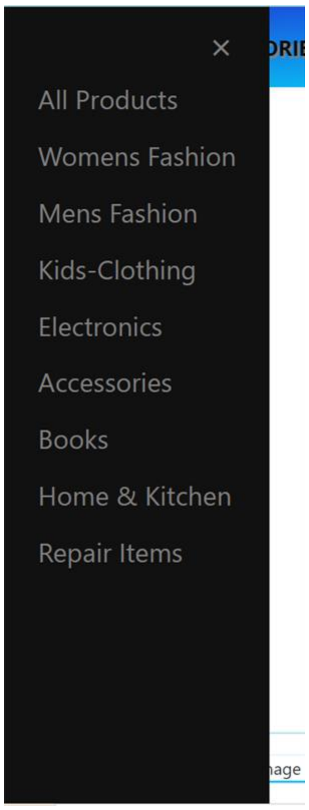
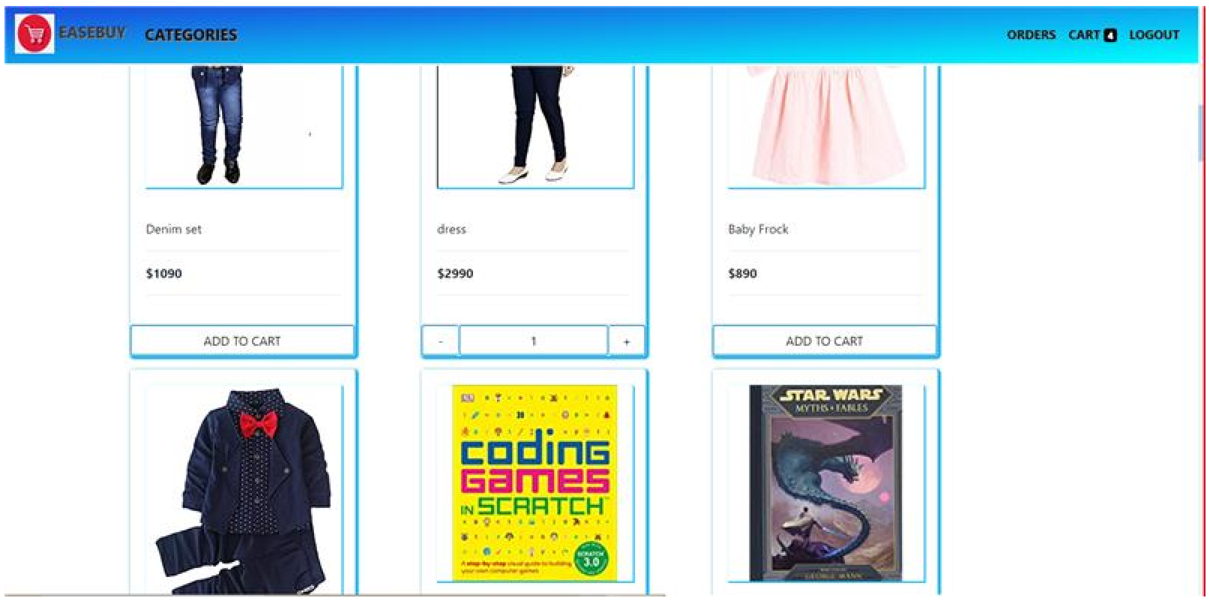


Figure 11: Categories In EaseBuy

8.4 Store



Denim set
$1090

dress
$2990

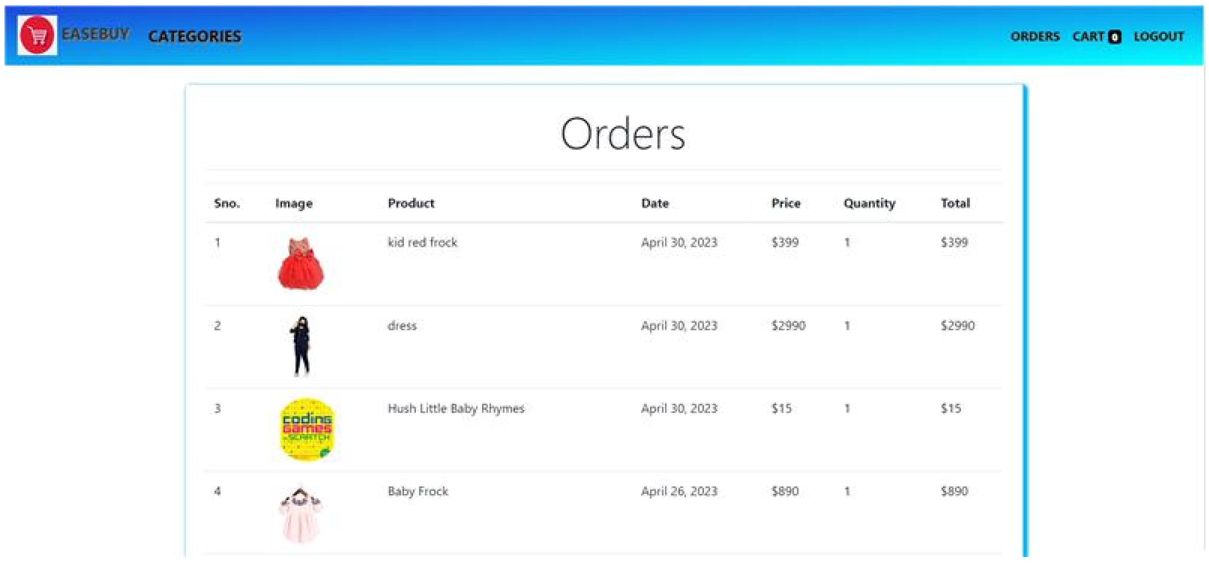
Baby Frock
$890

Figure 12: Store of EaseBuy

8.5 Orders

Upon selecting the orders function, it will using the select query it will fetch all the orders with a particuar customer_id.

This function will the display id, quatity, price, date, status of that order and using the product_id it will fetch the image from the store_products table and then display it.



Orders

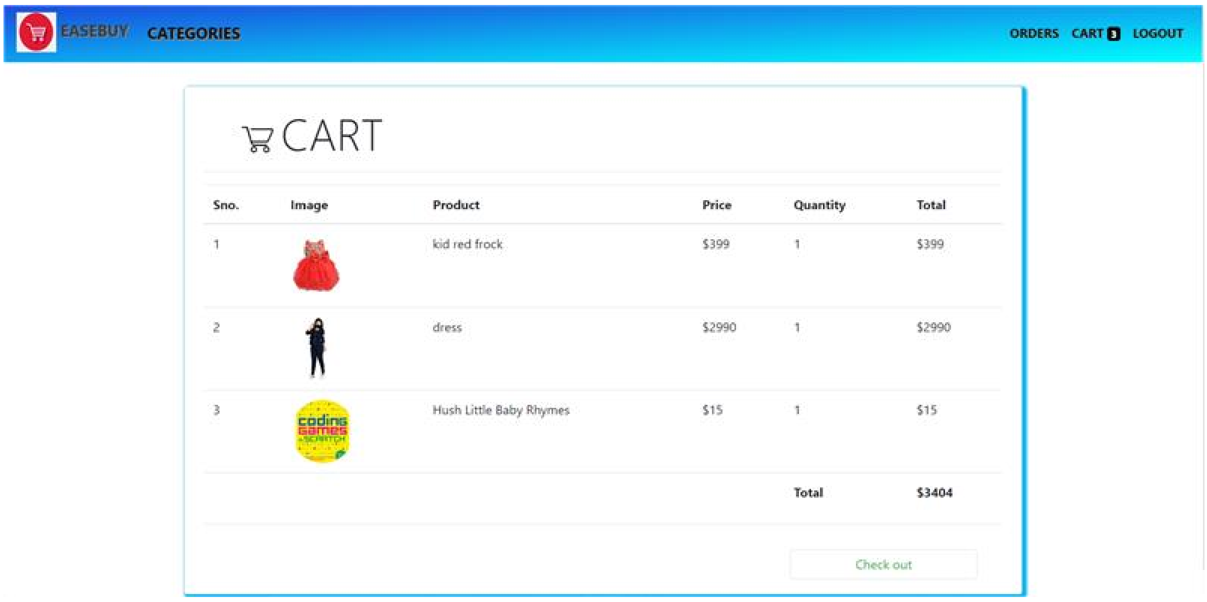
Sno.	Image	Product	Date	Price	Quantity	Total
1		kid red frock	April 30, 2023	$399	1	$399
2		dress	April 30, 2023	$2990	1	$2990
3		Hush Little Baby Rhymes	April 30, 2023	$15	1	$15
4		Baby Frock	April 26, 2023	$890	1	$890

Figure 13: Orders Page for EaseBuy

8.6 Cart

In this no SQL query is implemented.

everything is done using the function variables and displayed them on the UI.

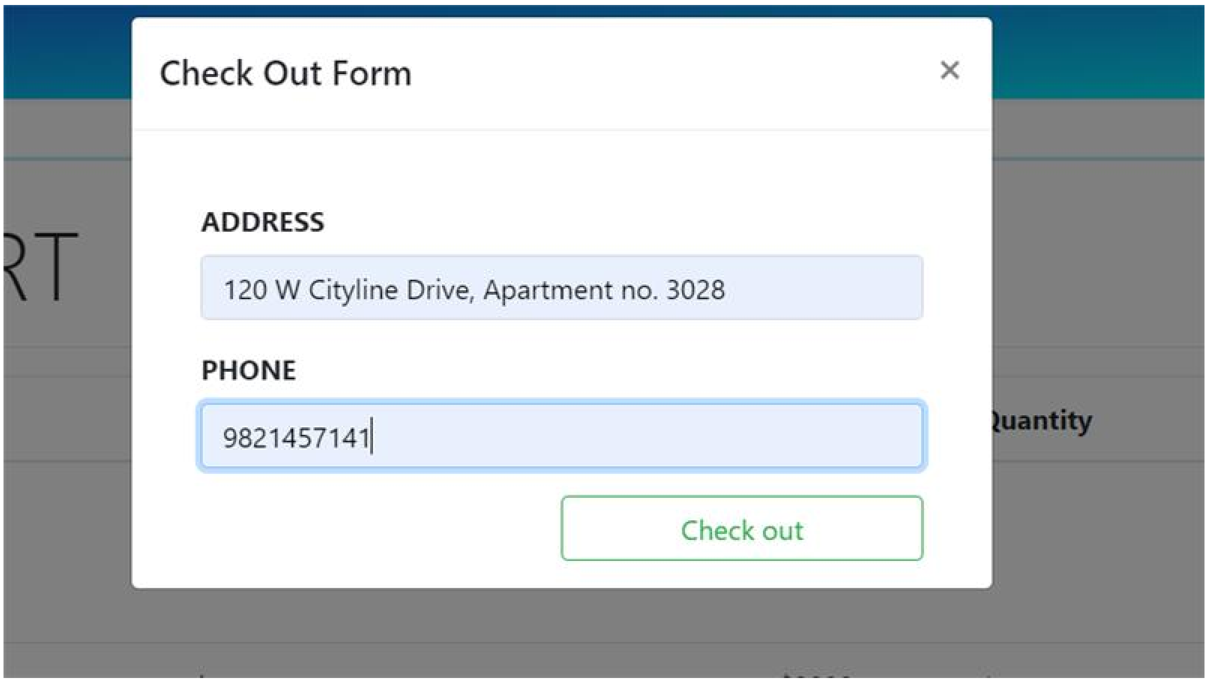


Sno.	Image	Product	Price	Quantity	Total
1		kid red frock	$399	1	$399
2		dress	$2990	1	$2990
3		Hush Little Baby Rhymes	$15	1	$15
				Total	$3404

Figure 14: Cart Page for EaseBuy

8.7 Checkout

As soon as the checkout button is clicked, a select SQL query is generated using the variables, and the order table is appended with selected products.


Check Out Form

ADDRESS

120 W Cityline Drive, Apartment no. 3028

PHONE

9821457141

Figure 15: Checkout Page for EaseBuy

9 Conclusions and Future Work

As we have approached the end of the project, we can successfully allow the customers to create their accounts on the E-commerce website. After the login is successful, the customers have complete access to the website which allows them to view the different products. Our website allows the customers to purchase items from the different categories like clothes, electronics and accessories. The items that are displayed on the website are those which are manually fed to the database developed by our team.

We propose this model to allow online and safe bank transactions to make successful payments. At present the database is limited which we aim to expand and later showcase a large number of products from various categories. Also, we aim to allow different retailers to sell the same product which will allow the customers to obtain different prices for the same category of products.

References

[1] Fundamentals of Database Systems, Elmasri, R and Navathe, Shamkant B and Elmasri, R and Navathe, SB, 2000,Springer.

[2] Dr. George Kocur,Database, Internet, And Systems Integration Technologies, Fall 2013, Massachusetts Institute of Technology: MIT OpenCouseWare, https://ocw.mit.edu/. License: Creative Commons BY-NC-SA.