Bilkent University

Project Design Report

CraftValley: Online Handcrafted Goods Marketplace Platform

Group 9

Berkay Gündüz, 22103402

Ege Şirvan, 22102289

Eren Aslan, 22102329

Kemal Sarper Şahin, 22103801

Öykü Elis Türegün, 21902976

Department of Computer Engineering

CS353 Database Systems

Özgür Ulusoy

25 March 2024

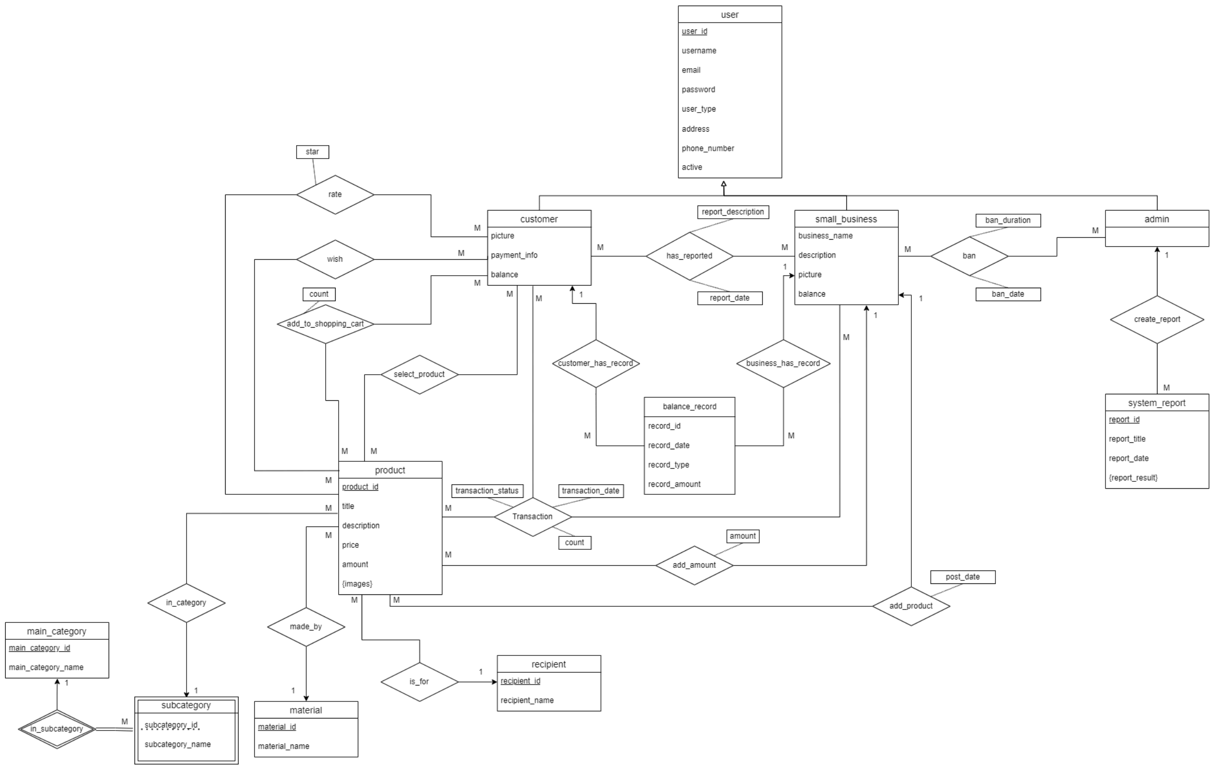
Table of Contents

1. Design of the Database

1.1 Revised E-R Diagram

Changes:

- ‘has_reported’ became a relationship table between the customer table and the small business table to show who reported which small business.
- ‘ban’ became a relationship table between small_business and admin tables in order to show that admins have the power to ban small_businesses.
- Connections of ‘rate’, ‘wish’, and ‘add_to_shopping_cart’ are fixed.
- ‘select_product’ is added to comply with new requirements.
- Instead of ‘has_record’, ‘customer_has_record’ and ‘business_has_record’ were created since the ‘has_records’ had a ternary relationship, which is unsuitable for this case.
- The ‘transaction_status’ attribute was added to the Transaction table in order to show whether the product of that transaction is returned or not.
- The ‘add_amount’ table is added in order to allow small businesses to add more of the same product to the market.
- ‘has_stock’ is changed to ‘add_product’ in order to allow businesses to add new products to the market.
- ‘recipient’ and ‘is_for’ tables were added to comply with the new requirements.
- Connections surrounding the ‘in_category’ table were fixed. Now ‘in_category’ table is only connected with the ‘product’ and ‘subcategory’ tables. Also ‘made_by’ table was added in order to connect the ‘material’ table with the ‘product’ table.
- ‘system_report’ table and ‘create_report’ table that connects ‘system_report’ with ‘admin’ were added to show that admins can generate system_reports.



1.2 Table Schemas

User

Relational Model:

```
User(user_id, user_name, email, password, user_type, address, phone_number,
active)
```

Functional Dependencies:

```
user_id -> user_name, email, password, user_type, address, phone_number,
active
email -> user_id
phone_number -> user_id
```

Candidate Keys:

```
{user_id}, {email}, {phone_number}
```

Primary Key:

```
user_id
```

Foreign Keys:

```
None
```

Normal Form: BCNF

SQL Definition:

```
CREATE TABLE IF NOT EXISTS User (
    user_id         INT NOT NULL AUTO_INCREMENT,
    user_name       VARCHAR(255) NOT NULL,
    email           VARCHAR(255) NOT NULL,
    password        VARCHAR(255) NOT NULL,
    user_type       VARCHAR(255) NOT NULL,
    address         VARCHAR(255),
    phone_number    VARCHAR(20) NOT NULL,
    active          INT NOT NULL CHECK(active IN(0,1)),
    PRIMARY KEY(user_id),
    UNIQUE KEY(email),
    UNIQUE KEY(phone_number)
);
```

Small Business

Relational Model:

```
Small_Business(user_id, business_name, title, description, picture, balance)
```

Functional Dependencies:

```
user_id -> business_name, title, description, picture, balance
```

Candidate Keys:

```
{user_id}
```

Primary Key:

```
user_id
```

Foreign Keys:

```
user_id -> User(user_id)
```

Normal Form: BCNF

SQL Definition:

```
CREATE TABLE IF NOT EXISTS Small_Business (
    user_id         INT NOT NULL,
    business_name   VARCHAR(255) NOT NULL,
    title           VARCHAR(255) NOT NULL,
    description     VARCHAR(255),
    picture         BLOB,
    balance         DECIMAL(10,2),
    PRIMARY KEY(user_id),
    FOREIGN KEY(user_id) REFERENCES User(user_id) ON DELETE CASCADE
);
```

Customer

Relational Model:

```
Customer(user_id, picture, payment_info, balance)
```

Functional Dependencies:

```
user_id -> picture, payment_info, balance
```

Candidate Keys:

```
{user_id}
```

Primary Key:

```
user_id
```

Foreign Keys:

```
user_id -> User(user_id)
```

Normal Form: BCNF

SQL Definition:

```
CREATE TABLE IF NOT EXISTS Customer(
    user_id         INT NOT NULL,
    picture         BLOB,
    payment_info    VARCHAR(255) NOT NULL,
    balance         DECIMAL(10,2) NOT NULL,
    PRIMARY KEY(user_id),
    FOREIGN KEY(user_id) REFERENCES User(user_id) ON DELETE CASCADE
);
```

Admin

Relational Model:

```
Admin(user_id)
```

Functional Dependencies:

```
user_id -> (No other attributes)
```

Candidate Keys:

```
{user_id}
```

Primary Key:

```
user_id
```

Foreign Keys:

```
user_id -> User(user_id)
```

Normal Form: BCNF

SQL Definition:

```
CREATE TABLE IF NOT EXISTS Admin(
    user_id INT NOT NULL,
    PRIMARY KEY(user_id),
    FOREIGN KEY(user_id) REFERENCES User(user_id) ON DELETE CASCADE
);
```

Product

Relational Model:

```
Product(product_id, title, description, price, amount, images)
```

Functional Dependencies:

```
product_id -> title, description, price, amount, images
```

Candidate Keys:

```
{product_id}
```

Primary Key:

```
product_id
```

Foreign Keys:

```
None
```

Normal Form: BCNF

SQL Definition:

```
CREATE TABLE IF NOT EXISTS Product(
    product_id      INT NOT NULL,
    title           VARCHAR(255) NOT NULL,
    description     VARCHAR(255),
    price           DECIMAL(10,2) NOT NULL,
    amount          INT NOT NULL,
    images          BLOB,
    PRIMARY KEY(product_id)
);
```

Balance Record

Relational Model:

```
Balance_Record(record_id, record_date, record_type, record_amount)
```

Functional Dependencies:

```
record_id -> record_date, record_type, record_amount
```

Candidate Keys:

```
{record_id}
```

Primary Key:

```
record_id
```

Foreign Keys:

```
None
```

Normal Form: BCNF

SQL Definition:

```
CREATE TABLE IF NOT EXISTS Balance_Record(
    record_id       INT NOT NULL,
    record_date     DATE NOT NULL,
    record_type     VARCHAR(255) NOT NULL,
    record_amount   DECIMAL(10,2) NOT NULL,
    PRIMARY KEY(record_id)
);
```

Recipient

Relational Model:

```
Recipient(recipient_id, recipient_name)
```

Functional Dependencies:

```
recipient_id -> recipient_name
```

Candidate Keys:

```
{recipient_id}, {recipient_name}
```

Primary Key:

```
recipient_id
```

Foreign Keys:

```
None
```

Normal Form: BCNF

SQL Definition:

```
CREATE TABLE IF NOT EXISTS Recipient(
    recipient_id          INT NOT NULL,
    recipient_name        VARCHAR(255) NOT NULL,
    PRIMARY KEY(recipient_id),
    UNIQUE KEY(recipient_name)
);
```

Material

Relational Model:

```
Material(material_id, material_name)
```

Functional Dependencies:

```
material_id -> material_name
```

Candidate Keys:

```
{material_id},{material_name}
```

Primary Key:

```
material_id
```

Foreign Keys:

```
None
```

Normal Form: BCNF

SQL Definition:

```
CREATE TABLE IF NOT EXISTS Material(
    material_id     INT NOT NULL,
    material_name   VARCHAR(255) NOT NULL,
    PRIMARY KEY(material_id),
    UNIQUE KEY(material_name)
);
```

Main Category

Relational Model:

```
Main_Category(main_category_id, main_category_name)
```

Functional Dependencies:

```
main_category_id -> main_category_name
```

Candidate Keys:

```
{main_category_id}, {main_category_name}
```

Primary Key:

```
main_category_id
```

Foreign Keys:

```
None
```

Normal Form: BCNF

SQL Definition:

```
CREATE TABLE IF NOT EXISTS Main_Category(
    main_category_id       INT NOT NULL,
    main_category_name     VARCHAR(255) NOT NULL,
    PRIMARY KEY(main_category_id)
);
```

Subcategory

Relational Model:

```
Sub_Category(sub_category_id, main_category_id, sub_category_name)
```

Functional Dependencies:

```
sub_category_id, main_category_id -> subcategory_name
```

Candidate Keys:

```
{sub_category_id, main_category_id}, {sub_category_name, main_category_name}
```

Primary Key:

```
(sub_category_id, main_category_id)
```

Foreign Keys:

```
main_category_id -> Main_Category(main_category_id)
```

Normal Form: BCNF

SQL Definition:

```
CREATE TABLE IF NOT EXISTS Sub_Category(
    sub_category_id        INT NOT NULL,
    main_category_id       INT NOT NULL,
    sub_category_name      VARCHAR(255) NOT NULL,
    PRIMARY KEY(sub_category_id, main_category_id),
    FOREIGN KEY(main_category_id) REFERENCES Main_Category(main_category_id)
ON DELETE CASCADE
);
```

In Category

Relational Model:

```
In_Category(sub_category_id, main_category_id, product_id)
```

Functional Dependencies:

```
sub_category_id, main_category_id, product_id → {no other attributes}
```

Candidate Keys:

```
{sub_category_id, main_category_id, product_id}
```

Primary Key:

```
(sub_category_id, main_category_id, product_id)
```

Foreign Keys:

```
main_category_id -> Sub_Category(main_category_id) ON DELETE CASCADE

sub_category_id -> Sub_Category(sub_category_id) ON DELETE CASCADE
product_id -> Product(product_id) ON DELETE CASCADE
```

Normal Form: BCNF

SQL Definition:

```
CREATE TABLE IF NOT EXISTS In_Category(
    sub_category_id        INT NOT NULL,
    main_category_id       INT NOT NULL,
    product_id             NOT NULL,
    PRIMARY KEY(sub_category_id, main_category_id, product_id),
    FOREIGN KEY(main_category_id) REFERENCES Main_Category(main_category_id)
ON DELETE CASCADE,

    FOREIGN KEY(sub_category_id) REFERENCES Sub_Category(sub_category_id) ON
DELETE CASCADE,
    FOREIGN KEY(product_id) REFERENCES Product(product_id)ON DELETE CASCADE
);
```

Has Reported

Relational Model:

```
Has_Reported(customer_id, small_business_id,
             report_description, report_date)
```

Functional Dependencies:

```
customer_id, small_business_id -> report_description, report_date
```

Candidate Keys:

```
{customer_id, small_business_id}
```

Primary Key:

```
(customer_id, small_business_id)
```

Foreign Keys

```
customer_id -> Customer(user_id)
small_business_id -> Small_Business(user_id)
```

Normal Form: BCNF

SQL Definition:

```
CREATE TABLE IF NOT EXISTS Has_Reported(
    customer_id           INT NOT NULL,
    small_business_id     INT NOT NULL,
    report_description    VARCHAR(255),
    report_date           DATE,
    PRIMARY KEY(customer_id, small_business_id),
    FOREIGN KEY(customer_id) REFERENCES Customer(user_id) ON DELETE CASCADE,
    FOREIGN KEY(small_business_id) REFERENCES Small_Business(user_id) ON
DELETE CASCADE
);
```

Ban

Relational Model:

```
Ban(admin_id, small_business_id, ban_duration, ban_date)
```

Functional Dependencies:

```
admin_id, small_business_id -> ban_duration, ban_date
```

Candidate Keys:

```
{admin_id, small_business_id}
```

Primary Key:

```
(admin_id, small_business_id)
```

Foreign Keys:

```
admin_id -> Admin(user_id)
small_business_id -> Small_Business(user_id)
```

Normal Form: BCNF

SQL Definition:

```
CREATE TABLE IF NOT EXISTS Ban(
    admin_id              INT NOT NULL,
    small_business_id     INT NOT NULL,
    ban_duration          VARCHAR(255),
    ban_date              DATE,
    PRIMARY KEY(admin_id, small_business_id),
    FOREIGN KEY(admin_id) REFERENCES Admin(user_id) ON DELETE CASCADE,
    FOREIGN KEY(small_business_id) REFERENCES Small_Business(user_id) ON
DELETE CASCADE
);
```

System Report

Relational Model:

```
System_Report(report_id, report_title, report_date, report_results)
```

Functional Dependencies:

```
report_id → report_title, report_date, report_results
```

Candidate Keys:

```
{report_id}
```

Primary Key:

```
(report_id)
```

Foreign Keys:

```
None
```

Normal Form: BCNF

SQL Definition:

```
CREATE TABLE IF NOT EXISTS System_Report(
    report_id              INT NOT NULL,
    report_title           VARCHAR(255) NOT NULL,
    report_date            DATE NOT NULL,
    report_results         VARCHAR(255),
    PRIMARY KEY(report_id)
    );
```

Create Report

Relational Model:

```
Create_Report(admin_id, report_id)
```

Functional Dependencies:

```
admin_id, report_id → {no other attributes}
```

Candidate Keys:

```
{admin_id, report_id}
```

Primary Key:

```
(admin_id, report_id)
```

Foreign Keys:

```
admin_id -> Admin(user_id)
report_id -> System_Report(report_id)
```

Normal Form: BCNF

SQL Definition:

```
CREATE TABLE IF NOT EXISTS Create_Report(
    admin_id        INT NOT NULL,
    report_id       INT NOT NULL,
    PRIMARY KEY(admin_id, report_id),
    FOREIGN KEY(admin_id) REFERENCES Admin(user_id) ON DELETE CASCADE,
    FOREIGN KEY(report_id) REFERENCES System_Report(report_id) ON DELETE
CASCADE
);
```

Rate

Relational Model:

```
Rate(customer_id, product_id, star)
```

Functional Dependencies:

```
customer_id, product_id -> star
```

Candidate Keys:

```
{customer_id, product_id}
```

Primary Key:

```
(customer_id, product_id)
```

Foreign Keys:

```
customer_id -> Customer(user_id)
product_id -> Product(product_id)
```

Normal Form: BCNF

SQL Definition:

```
CREATE TABLE IF NOT EXISTS Rate(
    customer_id     INT NOT NULL,
    product_id      INT NOT NULL,
    star            DECIMAL(2,1) NOT NULL,
    PRIMARY KEY(customer_id, product_id),
    FOREIGN KEY(customer_id) REFERENCES Customer(user_id) ON DELETE CASCADE,
    FOREIGN KEY(product_id) REFERENCES Product(product_id) ON DELETE CASCADE
);
```

Wish

Relational Model:

```
Wish(customer_id, product_id)
```

Functional Dependencies:

```
customer_id, product_id -> (No other attributes)
```

Candidate Keys:

```
{customer_id, product_id}
```

Primary Key:

```
(customer_id, product_id)
```

Foreign Keys:

```
customer_id -> Customer(user_id) ON DELETE CASCADE
product_id -> Product(product_id) ON DELETE CASCADE
```

Normal Form: BCNF

SQL Definition:

```
CREATE TABLE IF NOT EXISTS Wish(
    customer_id     INT NOT NULL,
    product_id      INT NOT NULL,
    PRIMARY KEY(customer_id, product_id),
    FOREIGN KEY(customer_id) REFERENCES Customer(user_id) ON DELETE CASCADE,
    FOREIGN KEY(product_id) REFERENCES Product(product_id) ON DELETE CASCADE
);
```

Add To Shopping Cart

Relational Model:

```
Add_To_Shopping_Cart(customer_id, product_id, count)
```

Functional Dependencies:

```
customer_id, product_id -> count
```

Candidate Keys:

```
{customer_id, product_id}
```

Primary Key:

```
(customer_id, product_id)
```

Foreign Keys:

```
customer_id -> Customer(user_id)
product_id -> Product(product_id)
```

Normal Form: BCNF

SQL Definition:

```
CREATE TABLE IF NOT EXISTS Add_To_Shopping_Cart(
    customer_id     INT NOT NULL,
    product_id      INT NOT NULL,
    count           INT NOT NULL,
    PRIMARY KEY(customer_id, product_id),
    FOREIGN KEY(customer_id) REFERENCES Customer(user_id) ON DELETE CASCADE,
    FOREIGN KEY(product_id) REFERENCES Product(product_id) ON DELETE CASCADE
);
```

Select Product

Relational Model:

```
Select_Product(customer_id, product_id)
```

Functional Dependencies:

```
customer_id, product_id -> (No other attributes)
```

Candidate Keys:

```
{customer_id, product_id}
```

Primary Key:

```
(customer_id, product_id)
```

Foreign Keys:

```
customer_id -> Customer(user_id)
product_id -> Product(product_id)
```

Normal Form: BCNF

SQL Definition:

```
CREATE TABLE IF NOT EXISTS Select_Product(
    customer_id     INT NOT NULL,
    product_id      INT NOT NULL,
    PRIMARY KEY(customer_id, product_id),
    FOREIGN KEY(customer_id) REFERENCES Customer(user_id) ON DELETE CASCADE,
    FOREIGN KEY(product_id) REFERENCES Product(product_id) ON DELETE CASCADE
);
```

Business Has Record

Relational Model:

```
Business_Has_Record(small_business_id, record_id)
```

Functional Dependencies:

```
small_business_id, record_id -> (No other attributes)
```

Candidate Keys:

```
{small_business_id, record_id}
```

Primary Key:

```
(small_business_id, record_id)
```

Foreign Keys:

```
small_business_id -> Small_Business(user_id)
record_id -> Balance_record(record_id)
```

Normal Form: BCNF

SQL Definition:

```
CREATE TABLE IF NOT EXISTS Business_Has_Record(
    small_business_id      INT NOT NULL,
    record_id              INT NOT NULL,
    PRIMARY KEY(record_id, small_business_id),
    FOREIGN KEY(small_business_id) REFERENCES Small_Business(user_id) ON
DELETE CASCADE,
    FOREIGN KEY(record_id) REFERENCES Balance_Record(record_id) ON DELETE
CASCADE
);
```

Customer Has Record

Relational Model:

```
Customer_Has_Record(customer_id, record_id)
```

Functional Dependencies:

```
customer_id, record_id -> (No other attributes)
```

Candidate Keys:

```
{customer_id, record_id}
```

Primary Key:

```
(customer_id, record_id)
```

Foreign Keys:

```
customer_id -> Customer(user_id)
record_id -> Balance_Record(record_id)
```

Normal Form: BCNF

SQL Definition:

```
CREATE TABLE IF NOT EXISTS Customer_Has_Record(
    customer_id            INT NOT NULL,
    record_id              INT NOT NULL,
    PRIMARY KEY(record_id, customer_id),
    FOREIGN KEY(customer_id) REFERENCES Customer(user_id) ON DELETE CASCADE,
    FOREIGN KEY(record_id) REFERENCES Balance_Record(record_id) ON DELETE
CASCADE
);
```

Transaction

Relational Model:

```
Transaction(product_id, customer_id, small_business_id, transaction_status,
transaction_date, count)
```

Functional Dependencies:

```
product_id, customer_id, small_business_id -> transaction_date, count,
transaction_status
```

Candidate Keys:

```
{product_id, customer_id, small_business_id}
```

Primary Key:

```
(product_id, customer_id, small_business_id)
```

Foreign Keys:

```
product_id -> Product(product_id)
customer_id -> Customer(user_id)
small_business_id -> Small_Business(user_id)
```

Normal Form: BCNF

SQL Definition:

```
CREATE TABLE IF NOT EXISTS Transaction(
    product_id            INT NOT NULL,
    customer_id           INT NOT NULL,
    small_business_id     INT NOT NULL,
    transaction_date      DATE NOT NULL,
    count                 INT NOT NULL,
    transaction_status    VARCHAR(255)NOT NULL,
    PRIMARY KEY(product_id, small_business_id),
    FOREIGN KEY(product_id) REFERENCES Product(product_id) ON DELETE
CASCADE,
    FOREIGN KEY(small_business_id) REFERENCES Small_Business(user_id) ON
DELETE CASCADE,
    FOREIGN KEY(customer_id) REFERENCES Customer(user_id) ON DELETE CASCADE
);
```

Add Amount

Relational Model:

```
Add_Amount(product_id, small_business_id, amount)
```

Functional Dependencies:

```
product_id, small_business_id -> amount
```

Candidate Keys:

```
{product_id, small_business_id}
```

Primary Key:

```
(product_id, small_business_id)
```

Foreign Keys:

```
product_id -> Product(product_id)
small_business_id -> Small_Business(user_id)
```

Normal Form: BCNF

SQL Definition:

```
CREATE TABLE IF NOT EXISTS Add_Amount(
    product_id            INT NOT NULL,
    small_business_id     INT NOT NULL,
    amount                INT NOT NULL,
    PRIMARY KEY(product_id, small_business_id),
    FOREIGN KEY(product_id) REFERENCES Product(product_id) ON DELETE
CASCADE,
    FOREIGN KEY(small_business_id) REFERENCES Small_Business(user_id) ON
DELETE CASCADE
);
```

Add Product
Relational Model:

Add_Product(<u>product_id</u>, <u>small_business_id</u>, post_date)

Functional Dependencies:

```
product_id, small_business_id -> post_date
```

Candidate Keys:

```
{product_id, small_business_id}
```

Primary Key:

```
(product_id, small_business_id)
```

Foreign Keys:

```
product_id -> Product(product_id)
small_business_id -> Small_Business(user_id)
```

Normal Form: BCNF
SQL Definition:

```
CREATE TABLE IF NOT EXISTS Add_Product(
    product_id            INT NOT NULL,
    small_business_id     INT NOT NULL,
    post_date             DATE NOT NULL,
    PRIMARY KEY(product_id, small_business_id),
    FOREIGN KEY(product_id) REFERENCES Product(product_id) ON DELETE
CASCADE,
    FOREIGN KEY(small_business_id) REFERENCES Small_Business(user_id) ON
DELETE CASCADE
);
```

Made By
Relational Model:

Made_By(<u>product_id</u>, <u>material_id</u>)

Functional Dependencies:

```
product_id, material_id -> (No other attributes)
```

Candidate Keys:

```
{product_id, material_id}
```

Primary Key:

```
(product_id, material_id)
```

Foreign Keys:

```
product_id -> Product(product_id)
material_id -> Material(material_id)
```

Normal Form: BCNF
SQL Definition:

```
CREATE TABLE IF NOT EXISTS Made_By(
    product_id      INT NOT NULL,
    material_id     INT NOT NULL,
    PRIMARY KEY(product_id, material_id),
    FOREIGN KEY(product_id) REFERENCES Product(product_id) ON DELETE
CASCADE,
    FOREIGN KEY(material_id) REFERENCES Material(material_id) ON DELETE
CASCADE
);
```

Is For

Relational Model:

Is_For(<u>product_id</u>, <u>recipient_id</u>)

Functional Dependencies:

```
product_id, recipient_id -> (No other attributes)
```

Candidate Keys:

```
{product_id, recipient_id}
```

Primary Key:

```
(product_id, recipient_id)
```

Foreign Keys:

```
product_id -> Product(product_id)
recipient_id -> Recipient(recipient_id)
```

Normal Form: BCNF

SQL Definition:

```
CREATE TABLE IF NOT EXISTS Is_For(
    product_id      INT NOT NULL,
    recipient_id    INT NOT NULL,
    PRIMARY KEY(product_id, recipient_id),
    FOREIGN KEY(product_id) REFERENCES Product(product_id) ON DELETE
CASCADE,
    FOREIGN KEY(recipient_id) REFERENCES Recipient(recipient_id) ON DELETE
CASCADE
);
```

1.3 Views

This view stores an instance of user login without displaying sensitive data

```
CREATE VIEW User_Login_View AS
SELECT user_id, email, user_name, phone_number
FROM User
```

This view shows the most popular products based on the number of transactions.

```
CREATE VIEW Popular_Products AS
SELECT product_id, COUNT(*) AS total_transactions
FROM Transaction
GROUP BY product_id
ORDER BY total_transactions DESC;
```

This view shows the total sales made by each small business.

```
CREATE VIEW Business_Sales AS
SELECT S.user_id, S.business_name, SUM(P.price * T.count) AS total_sales
FROM Small_Business AS S
INNER JOIN Product AS P ON S.user_id = P.user_id
INNER JOIN Transaction AS T ON P.product_id = T.product_id
GROUP BY S.user_id, S.business_name;
```

This view lists all businesses reported by customers along with their report details.

```
CREATE VIEW Admin_Reported_Businesses AS
SELECT R.small_business_id, S.business_name, R.report_description,
R.report_date
FROM Has_Reported AS R
INNER JOIN Small_Business AS S ON R.small_business_id = S.user_id;
```

This view provides a history of balance changes for each customer.

```
CREATE VIEW Customer_Balance_History AS
SELECT CHR.customer_id, BR.record_date, BR.record_type, BR.record_amount
FROM Balance_Record BR, Customer_Has_Record CHR
WHERE BR.record_id = CHR.record_id;
```

This view shows the top-selling products based on the number of transactions.

```
CREATE VIEW Best_Selling_Products AS
SELECT T.product_id, P.title, P.description, P.price, SUM(T.count) AS
total_sales
FROM Transaction T, Product P
WHERE T.product_id = P.product_id
GROUP BY T.product_id
ORDER BY total_sales DESC;
```

1.4 Triggers

This trigger automatically updates the customer's balance after a successful purchase.

```
CREATE TRIGGER Update_Customer_Balance
AFTER INSERT ON Transaction
FOR EACH ROW
BEGIN
    UPDATE Customer
    SET balance = balance - (SELECT (P.price * NEW.count)
                              FROM Product AS P
                              WHERE P.product_id = NEW.product_id)
    WHERE user_id = NEW.customer_id;
END;
```

This trigger can update the stock of a product after a purchase.

```
CREATE TRIGGER Update_Product_Stock
AFTER INSERT ON Transaction
FOR EACH ROW
BEGIN
    UPDATE Product
    SET amount = amount - NEW.count
    WHERE product_id = NEW.product_id;
END;
```

This trigger automatically bans a business when reported by enough customers.

```
CREATE TRIGGER Ban_Business
AFTER INSERT ON Has_Reported
FOR EACH ROW
BEGIN
    DECLARE reported_count INT;
    SET reported_count = (SELECT COUNT(*)
       FROM Has_Reported WHERE small_business_id = NEW.small_business_id);
    IF reported_count >= 10
    THEN
       INSERT INTO Ban (admin_id, small_business_id, ban_duration, ban_date)
              VALUES (@admin_id, NEW.small_business_id, 'Indefinite',
NOW());
    END IF;
END;
```

This trigger can automatically calculate and update the balance of a small business after each transaction.

```
CREATE TRIGGER Calculate_Business_Balance
AFTER INSERT ON Transaction
FOR EACH ROW
BEGIN
    UPDATE Small_Business
    SET balance = balance + (NEW.count * (SELECT price FROM Product WHERE
product_id = NEW.product_id))
    WHERE user_id = NEW.small_business_id;
END;
```

This trigger can check a customer's balance before allowing a purchase and prevent the transaction if the balance is insufficient.

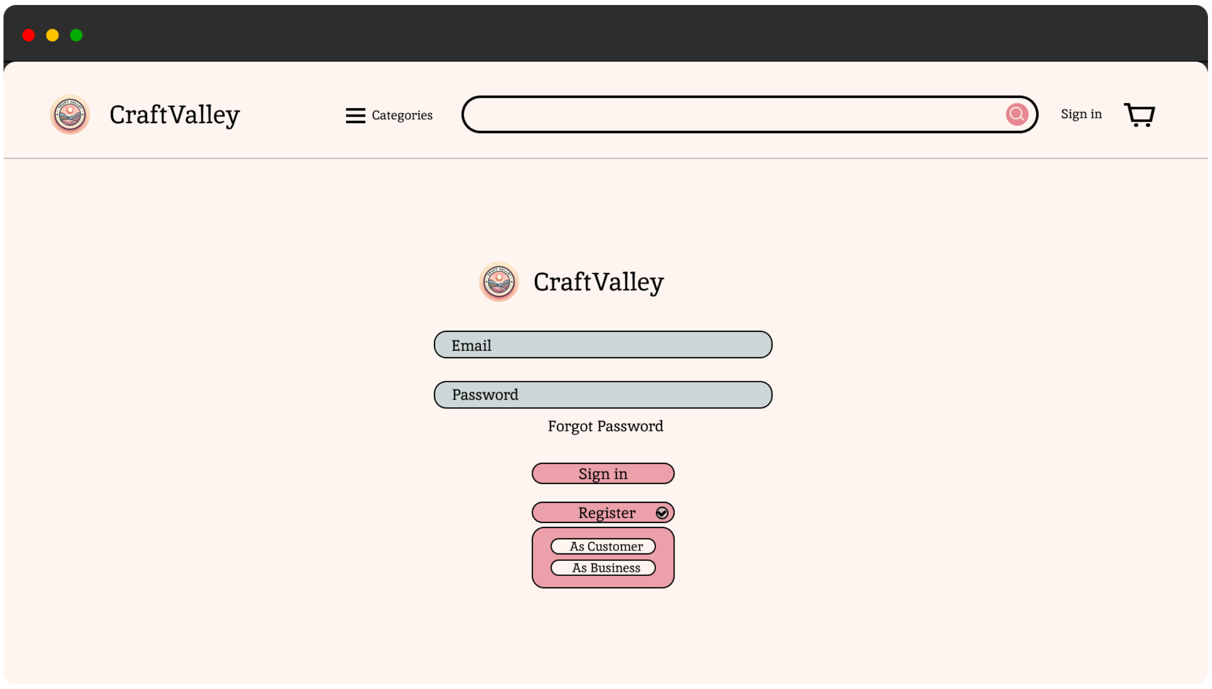
```
CREATE TRIGGER Check_Balance_Before_Purchase
BEFORE INSERT ON Transaction
FOR EACH ROW
BEGIN
    DECLARE customer_balance DECIMAL(10,2);
    SET customer_balance = (SELECT balance FROM Customer WHERE user_id =
NEW.customer_id);
    IF customer_balance < (NEW.count * (SELECT price FROM Product WHERE
product_id = NEW.product_id)) THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Insufficient balance for
purchase';
    END IF;
END;
```

2. Table Schemas

2.1 Common Functionalities

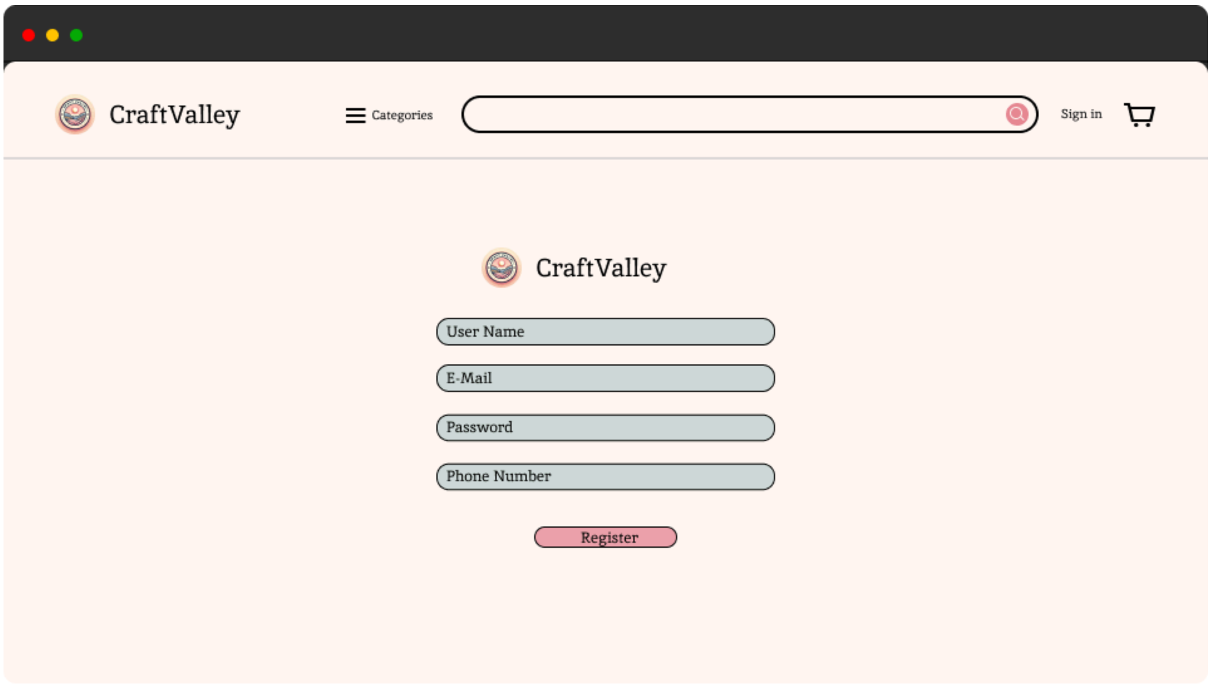
SQL Statements:

To login:



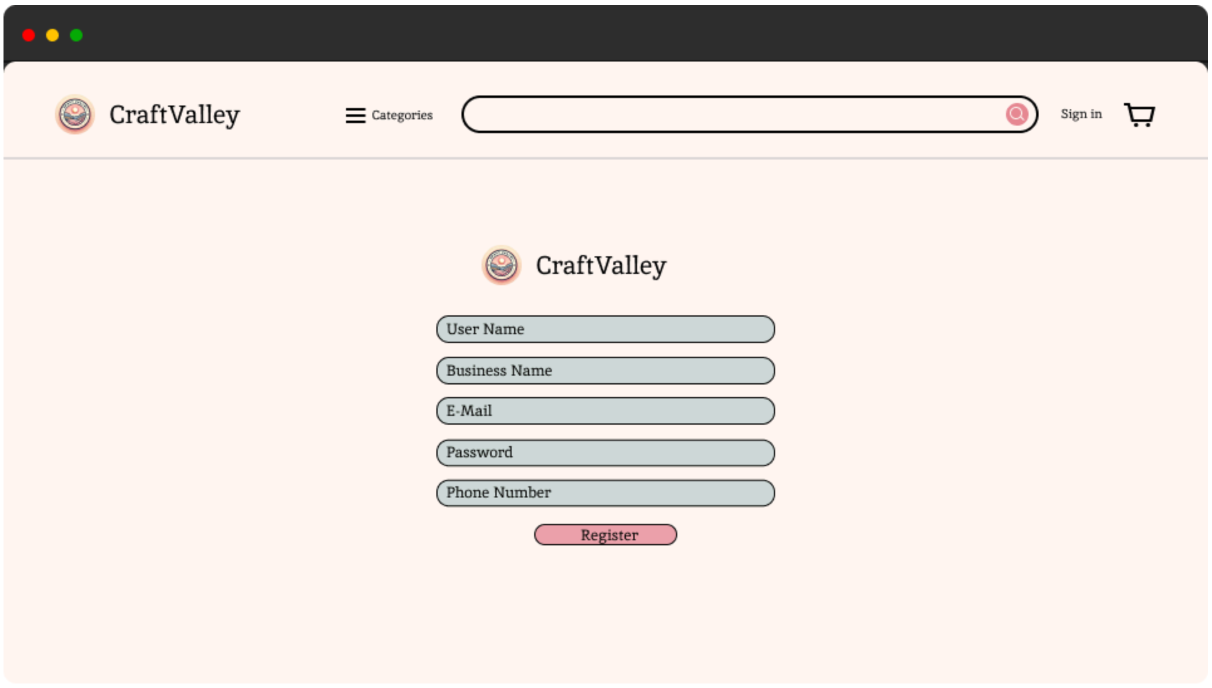
```
SELECT user_id, user_name
FROM User
WHERE email = @email AND password = @password;
```

To register as a customer:



```
INSERT INTO User (user_name, email, password,
                  user_type, phone_number, active)
VALUES (@user_name, @email, @password,
        'Customer', @phone_number, 1);
```

To register as a small business:

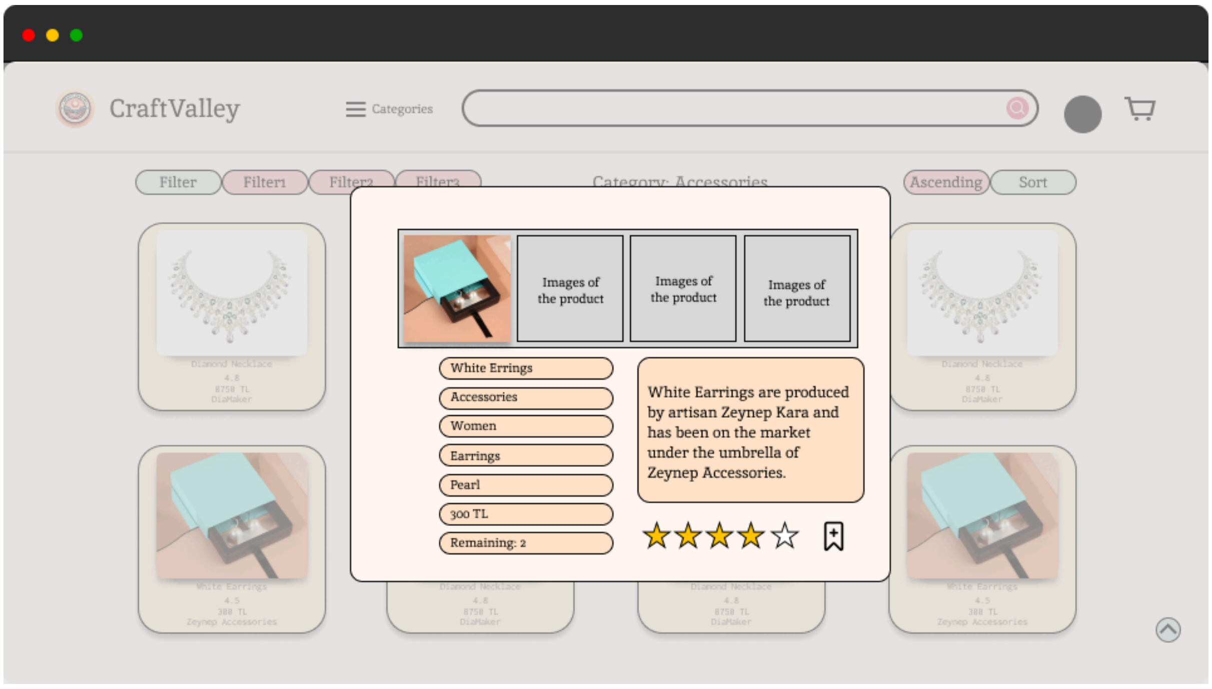


```
INSERT INTO User (user_name, email, password,
                  user_type, phone_number, active)
VALUES (@user_name, @email, @password,
        'Small Business', @phone_number, 1);
```

2.2 Additional Functionalities

SQL Statements:

To rate products:



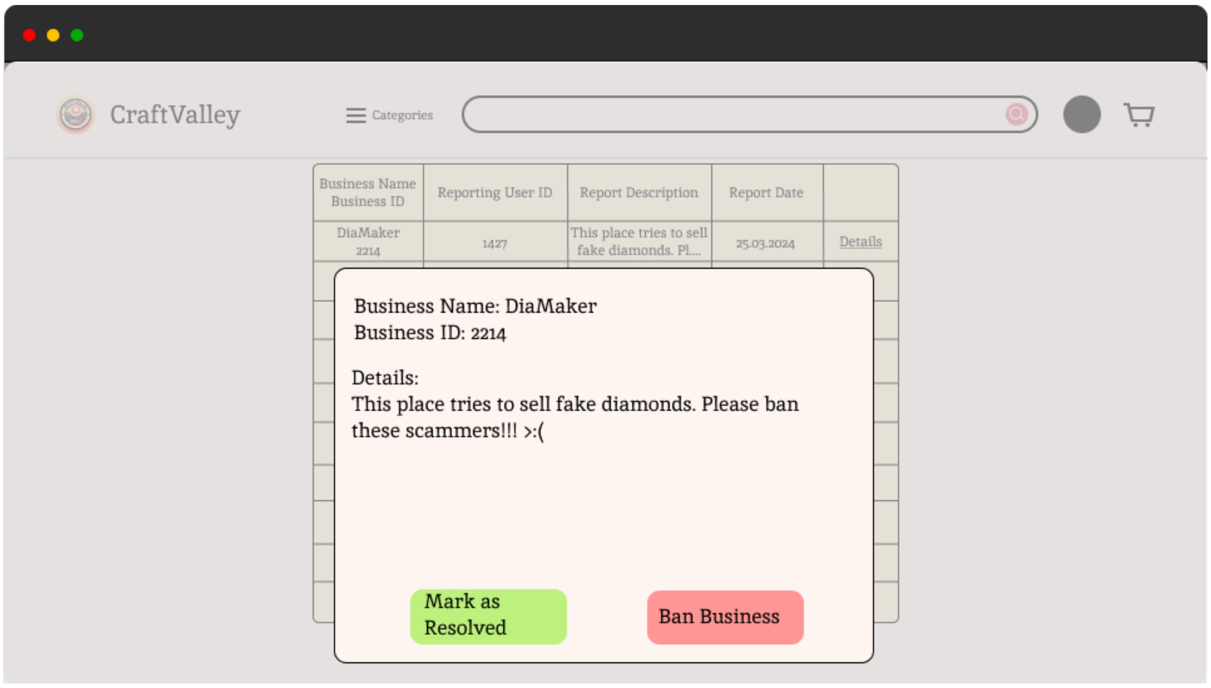
White Earrings are produced by artisan Zeynep Kara and has been on the market under the umbrella of Zeynep Accessories.

```
INSERT INTO Rate (customer_id, product_id, star)
VALUES (@customer_id, @product_id, @star);
```

To add a product to a wishlist:

```
INSERT INTO Wish (customer_id, product_id)
VALUES (@customer_id, @product_id);
```

ADMIN PANEL:



Business Name Business ID	Reporting User ID	Report Description	Report Date	
DiaMaker 2214	1427	This place tries to sell fake diamonds. Pl....	25.03.2024	Details

Business Name: DiaMaker
Business ID: 2214

Details:
This place tries to sell fake diamonds. Please ban these scammers!!! >:(

To ban a business:

```
INSERT INTO Ban (admin_id, small_business_id, ban_duration, ban_date)
VALUES (@admin_id, @small_business_id, @ban_duration, @ban_date);
```

To report a business:

```
INSERT INTO Has_Reported (customer_id, small_business_id,
                          report_description, report_status)
VALUES (@customer_id, @small_business_id,
        @report_description, @report_status);
```

To see Reported Businesses

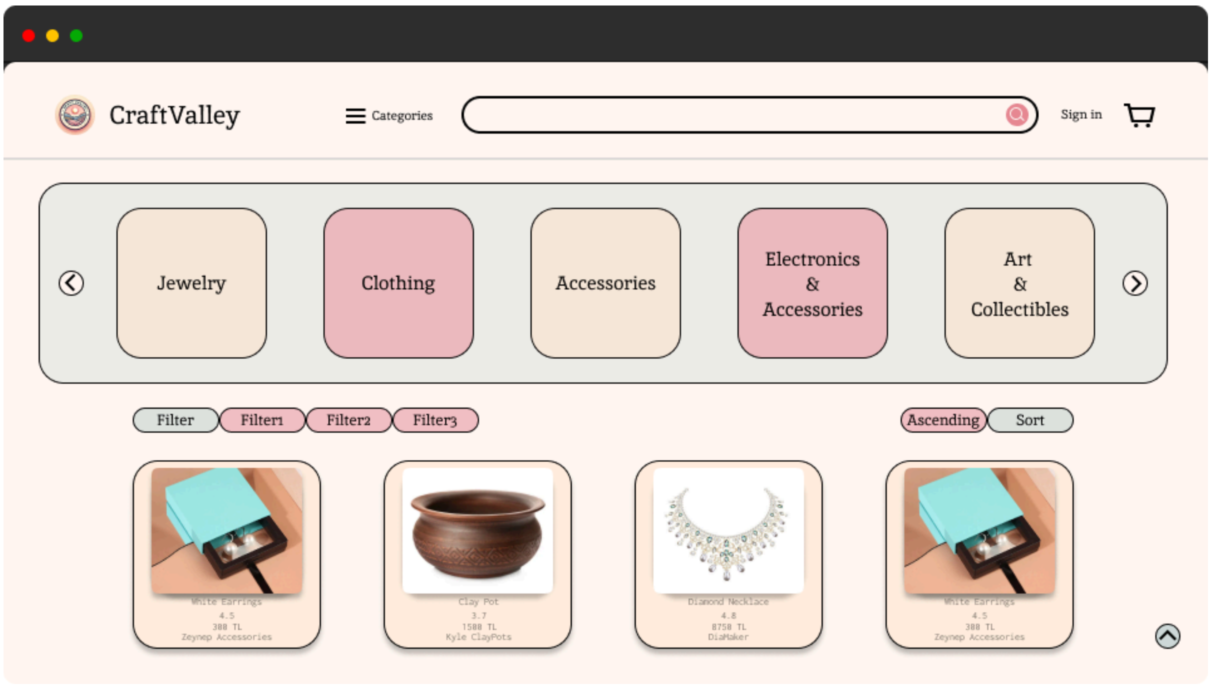
```
SELECT B.user_id, B.business_name, C.user_id, H.report_description,
H.report_date
FROM Small_Business as B, Customer as C, Has_reported as H
WHERE B.user_id = H.business_id AND C.user_id = H.customer_id
```

2.3 Topic Specific Functionalities

Purchasing a Product

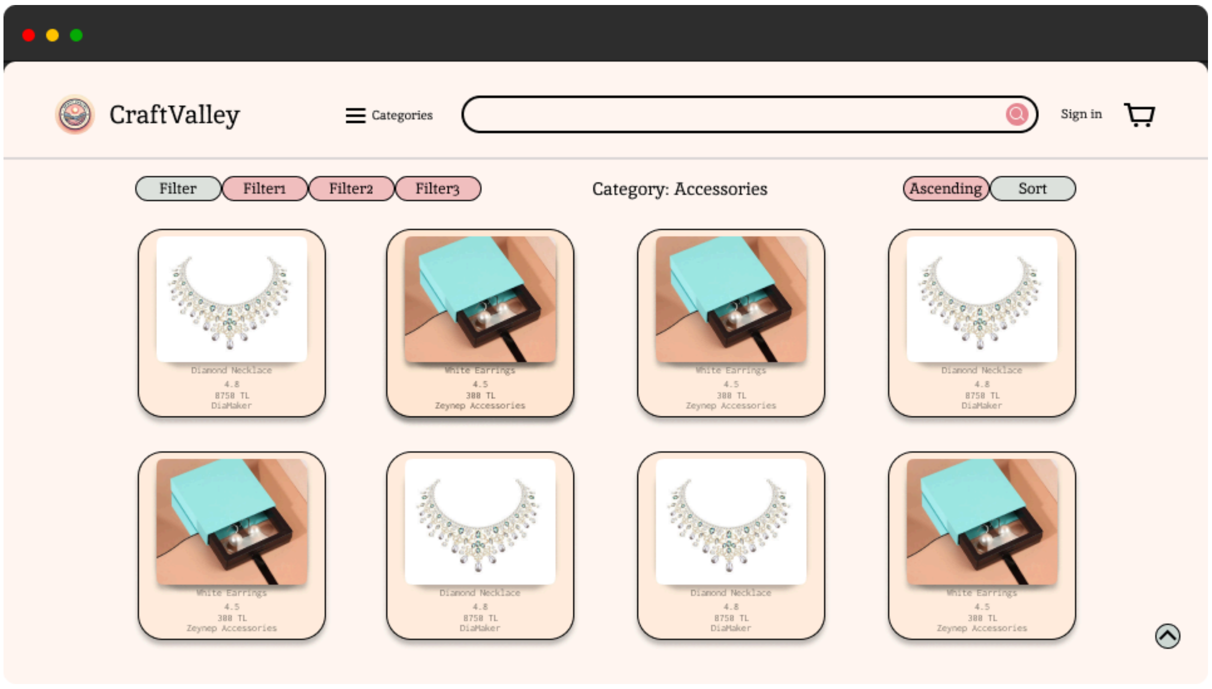
SQL Statements:

To list all products:


CraftValley
Categories
Sign in
Jewelry
Clothing
Accessories
Electronics & Accessories
Art & Collectibles
Filter
Filter1
Filter2
Filter3
Ascending
Sort
White Earrings 4.5 388 TL Zeynep Accessories
Clay Pot 3.7 1588 TL Kyle ClayPots
Diamond Necklace 4.8 8758 TL DiaMaker
White Earrings 4.5 388 TL Zeynep Accessories

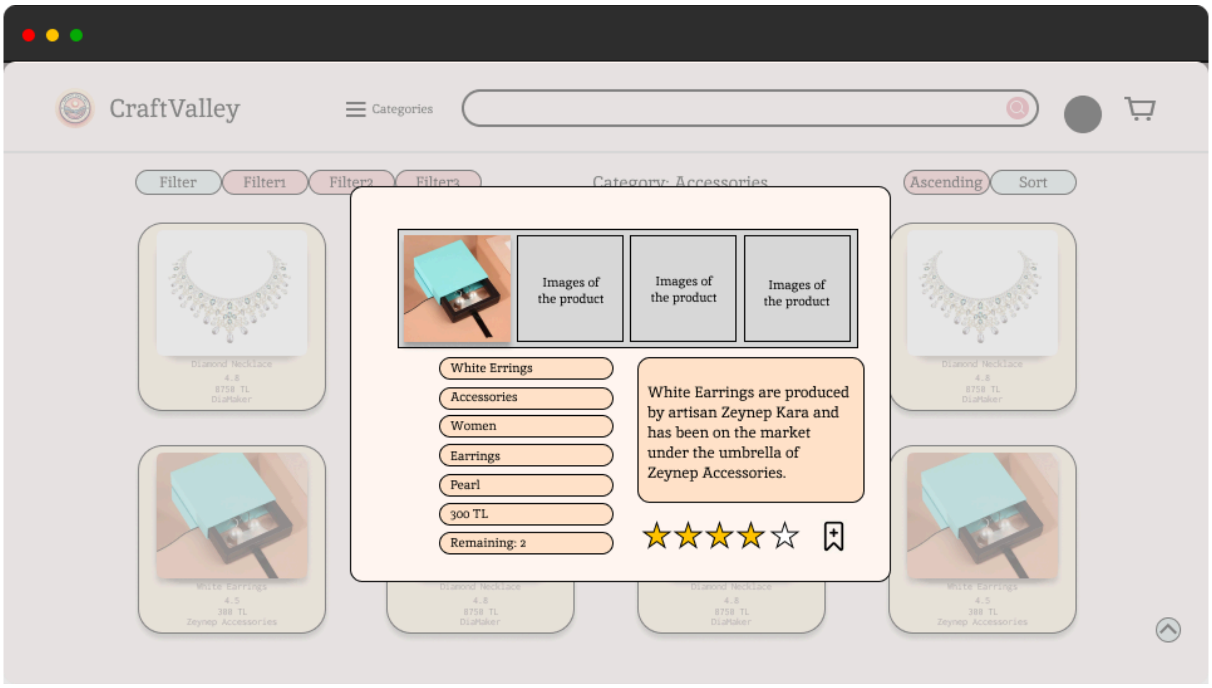
```
SELECT P.product_id, P.title, P.price
FROM Product
```

To apply filters:



```
SELECT P.product_id, P.title, P.price
FROM Product as P, Small_Business as B
WHERE P.product_id IN (
        SELECT P.product_id
        FROM Product as P, Subcategory as S, In_Category as I
        WHERE S.main_category_name = @main_category_name
                AND S.sub_category_name = @sub_category_name
                AND S.main_category_id = I.main_category_id
                AND S.sub_category_id = I.sub_category_id
                AND P.product_id = S.product_id)
        AND P.price < @price AND B.business_name = @business_name
```

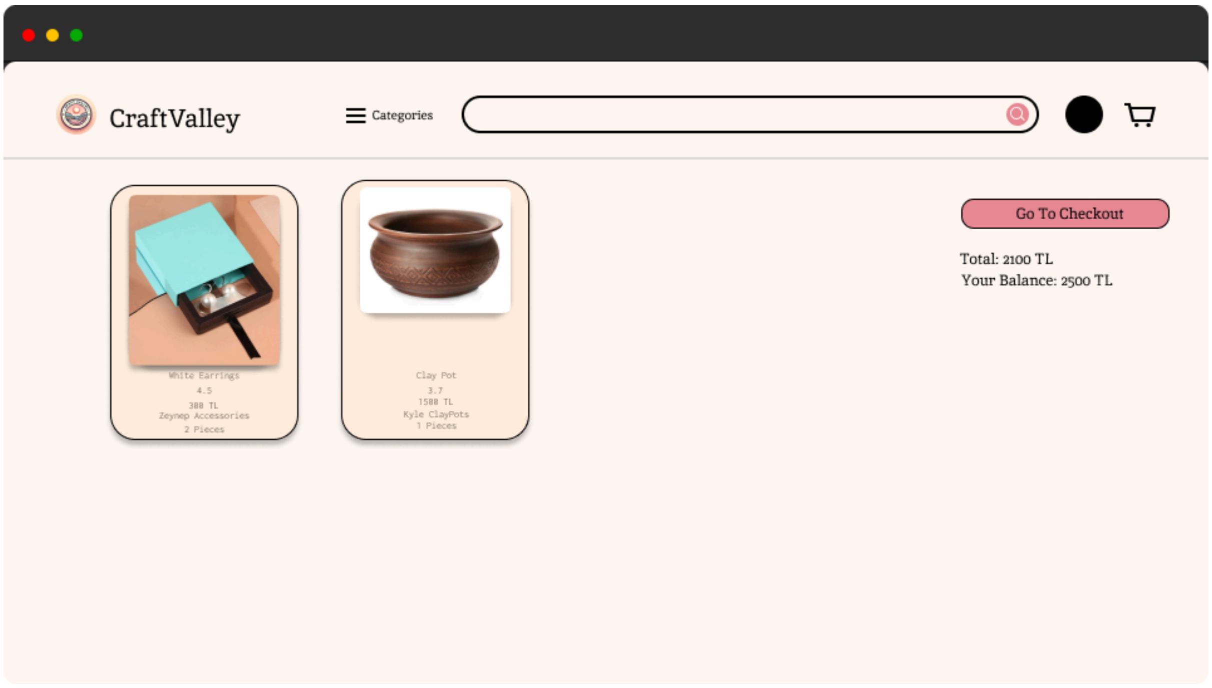
To select a good:



White Earrings are produced by artisan Zeynep Kara and has been on the market under the umbrella of Zeynep Accessories.

```
SELECT P.description, I.sub_category_name, I.main_category_name, P.title,
FROM Product as P, In_Category as I, Made_by as MB, Material as M
WHERE P.product_id = @product_id AND P.amount > 0
```

To add amount and add to the cart:



White Earrings
4.5
300 TL
Zeynep Accessories
2 Pieces

Clay Pot
3.7
1500 TL
Kyle ClayPots
1 Pieces

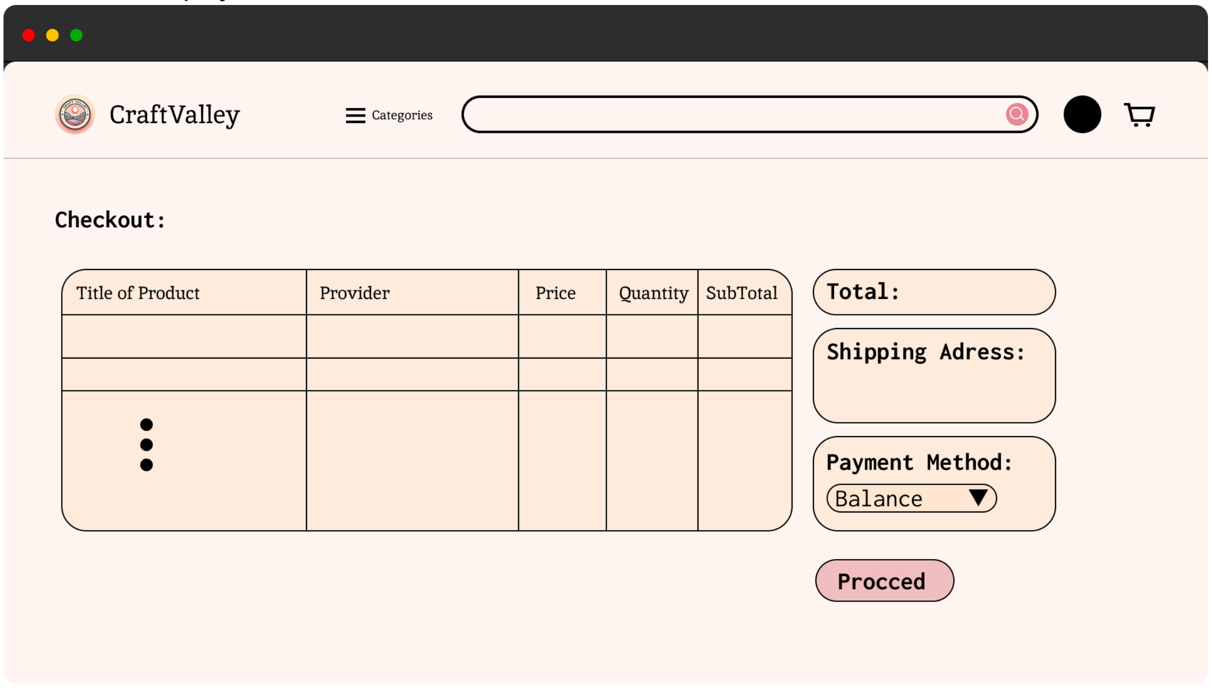
Total: 2100 TL
Your Balance: 2500 TL

```
INSERT INTO Add_to_shopping_cart (customer_id, product_id, amount)
VALUES (@user_ID, @post_ID, CASE WHEN EXISTS(SELECT 1
FROM Product
WHERE post_id = @post_id
      AND amount >= @amount)
THEN @amount
ELSE (SELECT amount
      FROM Product
      WHERE product_id = @product_id)
      END)
```

To show user's payment method:

```
SELECT C.payment_info
FROM Customer as C
WHERE C.user_id = @user_id
```

To make the payment:



Title of Product	Provider	Price	Quantity	SubTotal
⋮				

```
UPDATE Customer
SET Customer.balance = (Customer.balance - (
        SELECT (P.price * S.amount)
        FROM Add_to_shopping_cart as S, Customer as C, Product as P
        WHERE C.user_id = S.customer_id AND P.product_id = S.product_id))
WHERE Customer.user_id = @user_id


UPDATE Small_Business
SET Small_Business.balance = (Small_Business.balance + (
        SELECT (P.price * S.amount)
        FROM Add_to_shopping_cart as S, Customer as C, Product as P,
        Small_Business as B, Add_product as A
        WHERE C.user_id = S.customer_id AND P.product_id = S.product_id AND
        B.user_id = A.business_id AND A.product_id = P.product_id))


UPDATE Product
SET Product.amount = (Product.amount - (
        SELECT S.amount
        FROM Add_to_shopping_cart as S, Customer as C, Product as P
        WHERE C.user_id = S.customer_id AND P.product_id = S.product_id))
WHERE Product.product_id = @product_id
```

To add product to the user's purchased list:

```
INSERT INTO Transaction (product_id, small_business_id,
                         transaction_date, count)
VALUES (@product_id, (
       SELECT B.id
       FROM Small_Business as B, Add_product as A
       WHERE B.user_id = A.business_id AND A.product_id = @product_id),
       GETDATE(), @amount)
```

At the end purchased product should be deleted from the cart:

```
DELETE FROM Add_to_shopping_cart
WHERE product_id = @product_id AND user_id = @user_id
```

3. Implementation Plan

We'll craft our platform with Python Django at the backend, orchestrating user management, purchase processing, and data storage with MySQL. On the front end, we'll leverage Jinja templates along with jQuery for interactivity and Bootstrap for a polished, responsive layout suitable for all screen sizes. We'll handle all interactions between our application and database directly through SQL queries, rather than automation tools or libraries like ORM.