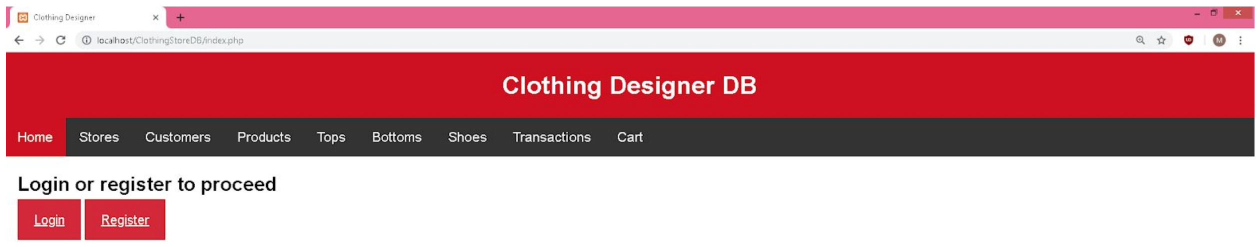
CS157A – Project-Milestone 2

Minh An Cao, Ricky Singh, Alexander Nguyen

Application Design
In this milestone you will concentrate on proposing / designing / implementing the following aspects of your database application: GUI design, application architecture, performance, data integrity and concurrency management.

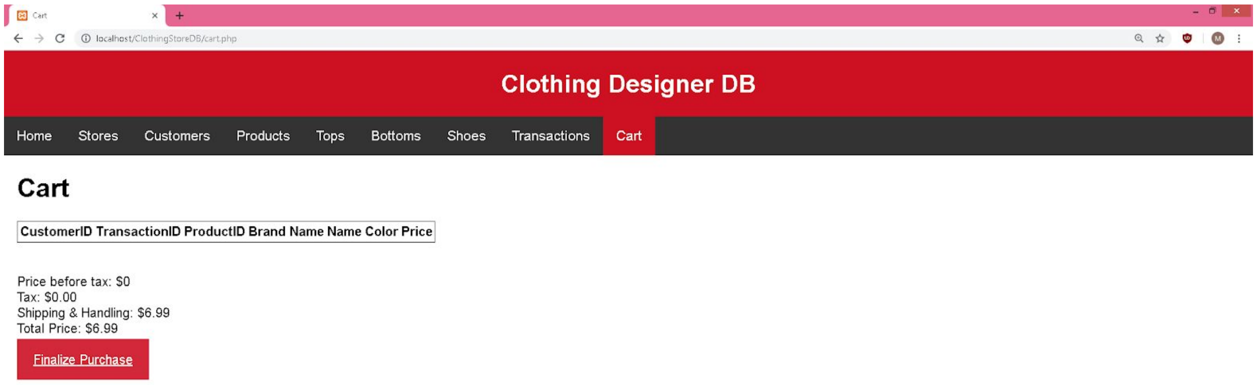
1. Design of the application menu

Propose and design a menu for your application (use ‘mock’ menu).



2. GUI design for the chosen module

Cart/transaction module

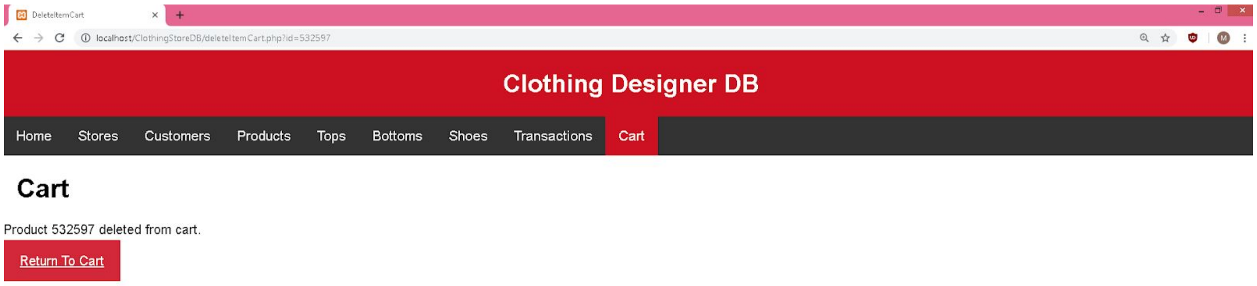


Cart

CustomerID	TransactionID	ProductID	Brand Name	Name	Color	Price

Price before tax: $0
Tax: $0.00
Shipping & Handling: $6.99
Total Price: $6.99

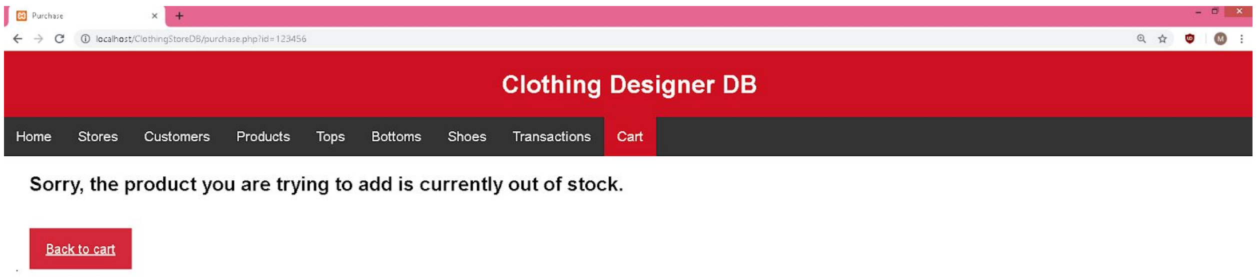
Finalize Purchase



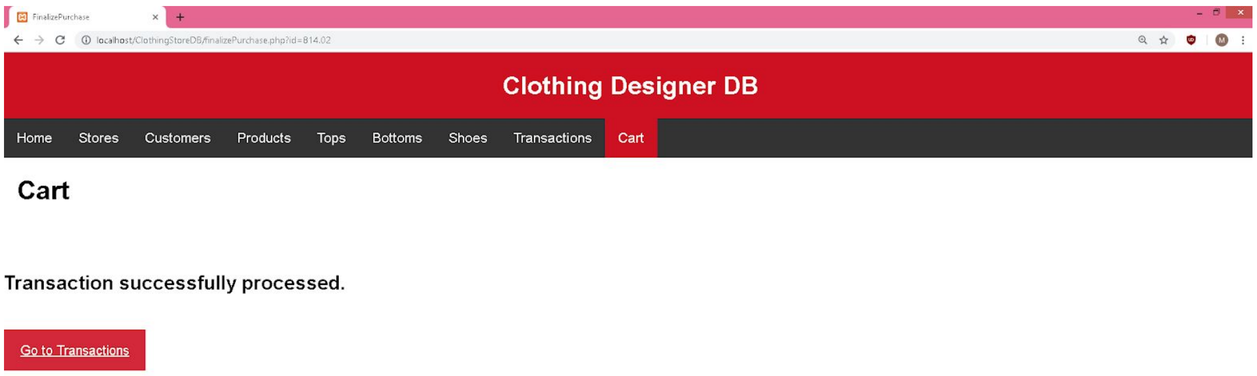
Cart

Product 532597 deleted from cart.

Return To Cart

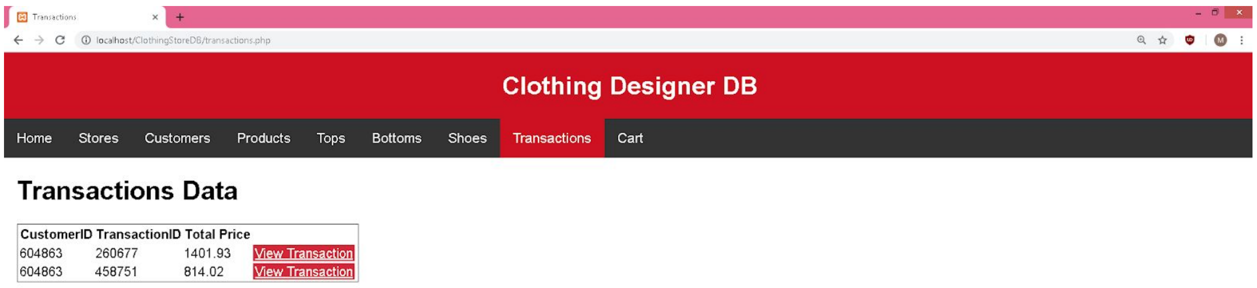


Sorry, the product you are trying to add is currently out of stock.



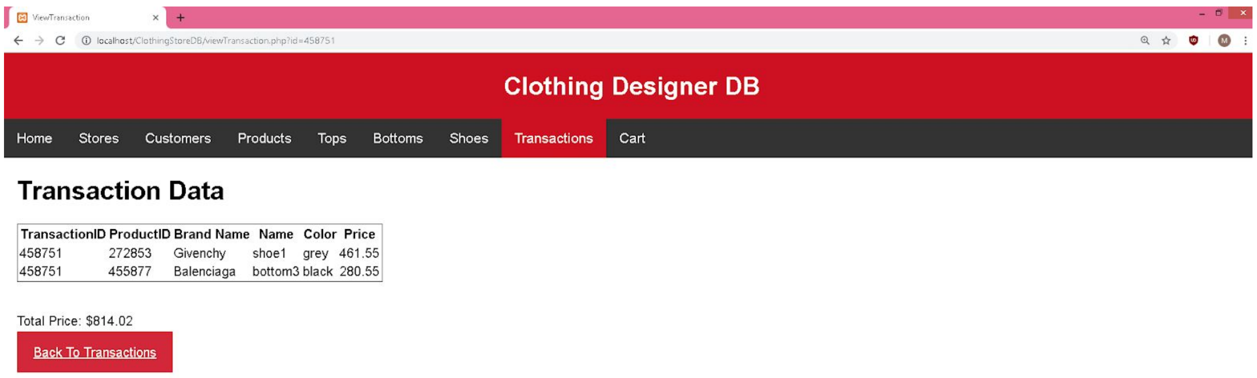
Cart

Transaction successfully processed.



Transactions Data

CustomerID	TransactionID	Total Price	
604863	260677	1401.93	View Transaction
604863	458751	814.02	View Transaction



Transaction Data

TransactionID	ProductID	Brand Name	Name	Color	Price
458751	272853	Givenchy	shoe1	grey	461.55
458751	455877	Balenciaga	bottom3	black	280.55

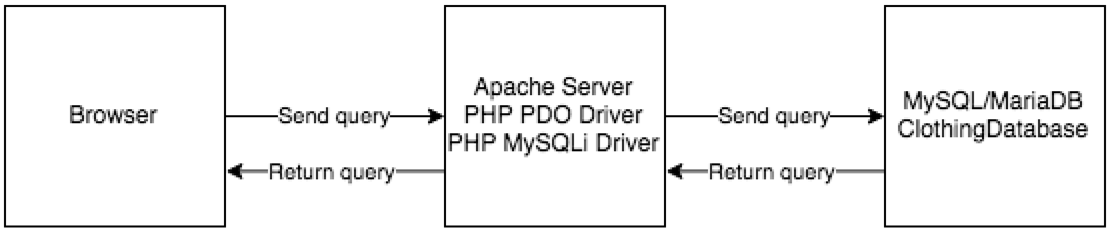
Total Price: $814.02

3. Application architecture

Modules + descriptions of the modules. Include diagram of the Client-Server architecture.

Modules:

- Home menu: Displays name of website, and need to login to proceed
- TransactionModule: Deals with transactions customer had in the past
- CartModule: Deals with current customer's cart of products
- ProductLookup: Deals with look up of products
- CustomerInfo: Displays info for currently logged in customer only



4. Server-side and Client-side functionality.

Describe which functionalities will be implemented on the Client side and which will be implemented on the server side.

Client:

- Look up about product, transaction, customer info, store, tops, bottoms, shoes
- Add/remove products in cart
- Finish purchase of items in cart
- Customer able to view past transactions belonging to that customer

Server:

- Return query requested from client
- Transactions, store, product is not allowed to be changed by client/customer
- Server can add/delete/modify products, stores, and anything else in the database

5. Integrity: constraints and triggers

Describe how integrity constraints will be enforced in your database system application (in the table definition and programmatically with assertion/triggers). Include code for defining constraints and code for triggers.

Constraints: Integrity constraints will be enforced through table definition and triggers. The Store table has storeName defined as the primary key. The Customer table has customerID defined as the primary key and also cardNumber as NOT NULL. The Transaction table has transactionID defined as the primary key. The Product table has productID defined as the primary key. The Top table has productID as the primary key and it references productID from Product table. The Bottom table has productID as the primary key and it references productID from Product table. The Shoe table has productID as the primary key and it references productID from Product table. The Carry table has storeName defined as the primary key and it references storeName from Store table and productID defined as the primary key and it references productID from Product table. The CustomerPurchases table has customerID and transactionID defined as the primary key. The Purchases table has transactionID and productID defined as the primary key and foreign key references to Transaction table and Product table. The Cart table has customerID and productID as primary key and foreign key references to Customer table and Product table.

Triggers:
When deleting a product in Product table, corresponding row in Top, Bottom, or Shoe table should also be deleted.

```
DELIMITER $$
CREATE OR REPLACE TRIGGER deleteProductInCorrespondingTables
   AFTER DELETE ON Product
   FOR EACH ROW
BEGIN
   DELETE FROM Top
  WHERE productID = OLD.productID;
   DELETE FROM Bottom
  WHERE productID = OLD.productID;
   DELETE FROM Shoe
  WHERE productID = OLD.productID;
END$$
DELIMITER ;
```

When inserting new row into Customer, does not allow insert if email or cardNumber is entered invalidly.

```
DELIMITER $$
CREATE OR REPLACE TRIGGER checkCustomer BEFORE INSERT ON
   Customer FOR EACH ROW
BEGIN
      IF(NEW.cardNumber IS NULL) THEN SIGNAL SQLSTATE '45000'
   SET MESSAGE_TEXT
      = 'card error' ;
   END IF ;
   IF(NEW.cardNumber = 0) THEN SIGNAL SQLSTATE '45000'
SET MESSAGE_TEXT
   = 'card error' ;
END IF ;
IF(NEW.cardNumber < 1000000000000000) THEN SIGNAL SQLSTATE '45000'
SET MESSAGE_TEXT
   = 'card error' ;
END IF ;
IF(
   NEW.email NOT REGEXP '^[^@]+@[^@]+.[^@]{2,}$'
) THEN SIGNAL SQLSTATE '45000'
SET MESSAGE_TEXT
   = 'email error' ;
END IF ;
END$$
DELIMITER ;
```

When product count is 0, it cannot be added onto cart.

```
DELIMITER $$
CREATE OR REPLACE TRIGGER productCountIsZero
   BEFORE INSERT ON Cart
   FOR EACH ROW
BEGIN
   DECLARE countCheck INT;

   SET countCheck := (SELECT count FROM Product WHERE productID = NEW.productID);

   IF(countCheck <= 0) THEN SIGNAL SQLSTATE '45000'
   SET MESSAGE_TEXT = 'Count <= 0';
   END IF;
END$$
DELIMITER ;
```

```
CREATE TABLE Store (
        storeName VARCHAR(255),
        popularity FLOAT,
        PRIMARY KEY(storeName)
);

CREATE TABLE Customer (
   customerID INT,
   password VARCHAR(255) NOT NULL,
   firstName VARCHAR(255),
   lastName VARCHAR(255),
   address VARCHAR(255),
   email VARCHAR(255) UNIQUE NOT NULL,
   cardNumber DECIMAL(16,0) NOT NULL,
   PRIMARY KEY(customerID)
);

CREATE TABLE Transaction (
        transactionID INT,
        price FLOAT,
        PRIMARY KEY(transactionID)
);

CREATE TABLE Product (
        productID INT,
        color VARCHAR(255),
        price FLOAT,
        brandName VARCHAR(255),
        name VARCHAR(255),
        count INT,
        PRIMARY KEY(productID)
);

CREATE TABLE Top (
        productID INT,
        hood TINYINT(1),
        size VARCHAR(5),
        pocket TINYINT(1),
        zipper TINYINT(1),
        PRIMARY KEY(productID),
        FOREIGN KEY (productID) REFERENCES Product(productID) ON DELETE
CASCADE
```

```
);

CREATE TABLE Bottom (
	productID INT,
	waistSize INT,
	lengthSize INT,
	PRIMARY KEY(productID),
	FOREIGN KEY (productID) REFERENCES Product(productID) ON DELETE
CASCADE
);

CREATE TABLE Shoe (
	productID INT,
	shoeSize FLOAT,
	PRIMARY KEY(productID),
	FOREIGN KEY (productID) REFERENCES Product(productID) ON DELETE
CASCADE
);

CREATE TABLE Carry (
	storeName VARCHAR(255),
	productID INT,
	PRIMARY KEY(storeName, productID),
	FOREIGN KEY (storeName) REFERENCES Store(storeName) ON DELETE
CASCADE,
	FOREIGN KEY (productID) REFERENCES Product(productID) ON DELETE
CASCADE
);

CREATE TABLE CustomerPurchases (
	customerID INT,
	transactionID INT,
	PRIMARY KEY(customerID, transactionID),
	FOREIGN KEY (customerID) REFERENCES Customer(customerID) ON
DELETE CASCADE,
	FOREIGN KEY (transactionID) REFERENCES Transaction(transactionID) ON
DELETE CASCADE
);

CREATE TABLE Purchases (
	transactionID INT,
	productID INT,
	PRIMARY KEY(transactionID, productID),
        FOREIGN KEY (transactionID) REFERENCES Transaction(transactionID) ON
DELETE CASCADE,
        FOREIGN KEY (productID) REFERENCES Product(productID) ON DELETE
CASCADE
);

CREATE TABLE Cart (
        customerID INT,
        transactionID INT,
        productID INT,
        PRIMARY KEY(transactionID, productID),
        FOREIGN KEY (customerID) REFERENCES Customer(customerID) ON
DELETE CASCADE,
        FOREIGN KEY (productID) REFERENCES Product(productID) ON DELETE
CASCADE
);
```

6. Performance: Indexes

Propose and design a set of indexes suitable for your systems, taking into account most common query transactions performed in your application. Give code for creating these indexes. Consider also designing materialized views to speed up certain queries.

Customers will want to look at products, what brands they are, and price.
```
CREATE INDEX ProductNameIndex on Product(name);
CREATE INDEX ProductBrandNameIndex on Product(brandName);
CREATE INDEX ProducttPriceIndex on Product(price);
```

Customer will want to know their previous transactions on their account.
```
CREATE INDEX CustomerTransactions on CustomerPurchases(customerID, transactionID);
```

Admin wants to see what products are in a particular transaction.
```
CREATE INDEX ProductsInOneTransaction on Purchases(transactionID, productID);
```

Customer will want to know their transaction and its total price.
```
CREATE INDEX transactionPrice on Transaction(transactionID, price);
```

7. Concurrency: Transactions for the chosen module

Transaction 1: Finish purchasing items

- Insert current transactionID and current customerID into customerPurchases table
- Insert the products in cart along with current transactionID into Purchases table
- Get all products in cart 's prices and sum it up and update total price of the current transaction in Transaction table
- Decrement by 1 to count in each product in cart

```
DELIMITER //
CREATE OR REPLACE PROCEDURE decrementCountInProduct
(IN p_productID INT)
BEGIN
  UPDATE Product
  SET count = count - 1
  WHERE productID = p_productID;
END //
DELIMITER ;

DELIMITER //
CREATE OR REPLACE PROCEDURE finalizeCartTransact
(IN p_transactionID INT, IN p_customerID INT)
BEGIN
  DECLARE v_totalprice FLOAT DEFAULT 0;

  SELECT SUM(p1.price) INTO v_totalprice
  FROM Product p1 INNER JOIN (SELECT * FROM Cart WHERE transactionID =
p_transactionID AND customerID = p_customerID) c1 ON p1.productID = c1.productID;

  SET v_totalprice = ROUND(calculateTotal(v_totalprice),2);

  INSERT INTO Transaction(transactionID, price)
  VALUES(p_transactionID, v_totalprice);

  INSERT INTO CustomerPurchases(transactionID, customerID)
  VALUES(p_transactionID, p_customerID);

  INSERT INTO Purchases(transactionID, productID)
  SELECT transactionID, productID
  FROM Cart WHERE transactionID = p_transactionID AND customerID = p_customerID;

  CALL decrementProductsInCart(p_transactionID);

  DELETE FROM Cart WHERE transactionID = p_transactionID AND customerID =
p_customerID;
```

```
END //
DELIMITER ;

DELIMITER //
CREATE OR REPLACE PROCEDURE decrementProductsInCart
(IN p_transactionID INT)
BEGIN
  DECLARE done BOOLEAN DEFAULT FALSE;
  DECLARE _id BIGINT UNSIGNED;
  DECLARE cur CURSOR FOR SELECT productID FROM Cart WHERE transactionID =
p_transactionID;
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET done := TRUE;

  OPEN cur;

  testLoop: LOOP
    FETCH cur INTO _id;
    IF done THEN
      LEAVE testLoop;
    END IF;
    CALL decrementCountInProduct(_id);
  END LOOP testLoop;

  CLOSE cur;
END //
DELIMITER ;

DELIMITER $$
CREATE OR REPLACE FUNCTION calculateTax (input FLOAT) RETURNS FLOAT
    DETERMINISTIC
  BEGIN
    RETURN input * 0.0875;
END$$

DELIMITER $$
CREATE OR REPLACE FUNCTION calculateTotal (input FLOAT) RETURNS FLOAT
    DETERMINISTIC
BEGIN
  DECLARE subtotal FLOAT DEFAULT 0;
  SET subtotal = input;
  SET subtotal = subtotal + calculateTax(subtotal);
    RETURN subtotal + 6.99;
END$$
```

Concurrency issues may occur when a customer places a product in his or her cart, and another customer requests for that item. We will resolve this by “taking items off the shelf”, in that when a user has items ready in the cart and is ready to purchase, he or she has a set time period to enter in their card information before the items are placed back into the registry. A user that is trying to add an item to the cart while another customer is in the purchasing phase will be notified that someone is making a purchase, and will not be allowed to add the item to his or her cart while the transaction is occuring; the user can only add the item once the time limit expires for the customer that was in the buying phase.