CSE 3241 Project Checkpoint 04

Functional Dependencies, Normal Forms, Indexes, Transactions

In a NEATLY TYPED document, provide the following:

1. Provide a current version of your ER Diagram and Relational Model as per Project Checkpoint 03. If you were instructed to change the model for Project Checkpoint 03, make sure you use the revised versions of your models.

Account (Account Number, Username, Type, Address, Karma Points, Phone Number, Name)
Buyer(Account Number)
Seller(Account Number)
Product(ProductID, StoreID, Name, Quantity, Availability, Price)
Image(ImageID, ProductID, Picture, Creation date, link)
Virtual Storefront(StoreID, Account Number, Name)
Payment_methods(Method_name, StoreID)
Payment(PaymentID, Order Number, Account Number, Type of payment, ExpDate)
Order(Order Number, Account Number, CartID, Order Date)
Shopping Cart(CartID, Account Number, Purchased)
Wishlist(WishID, Account Number, NumProducts)
Wish_Product(WishID, ProductID, Quantity)
Shop_Product(CartID, ProductID, Quantity)
Product_Image(ProductID, ImageID)
BuyerFeedback(ProductID, Account_Number, Feedback)

2. For each relation schema in your model, indicate the functional dependencies. Think carefully about what you are modeling here - make sure you consider all the possible dependencies in each relation and not just the ones from your primary keys. For example, a customer's credit card number is unique, and so will uniquely identify a customer even if you have another key in the same table (in fact, if the customer can have multiple credit card numbers, the dependencies can get even more involved).
Account:
{Account_Number} -> {Username, Type, Address, Karma Points, Phone Number, Name}
Username is a candidate key

Product:
{ProductID} → {StoreID, Name, Quantity, Availability, Price}

Image:
{ImageID} → {ProductID, Creation date, Link, Picture}

Virtual Storefront:
{StoreID} → {AccountNumber, Name}

Payment:
{PaymentID} → {ExpDate}

Order:
{Order Number} → {Order Date, CartID}
{CartID} → {Account Number}

Shopping Cart:
{Account Number, CartID} → {Purchased}

Wishlist:
{WishID, Account Number} -> {NumProducts}

Wish_Product:
{WishID, ProductID} -> {Quantity}

Shop_Product:
{CartID, ProductID} -> {Quantity}

BuyerFeedback:
{ProductID, Account Number} → {Feedback}

3. For each relation schema in your model, determine the highest normal form of the relation. If the relation is not in 3NF, rewrite your relation schema so that it is in at least 3NF.

Order:
{Order Number} → {Order Date, CartID}
{CartID} → {Account Number} Transitive dependency

Order is in 2NF since there is a transitive dependency. To fix this, we are removing account number from the relation since the CartID can give us account number information

Order:
{Order Number} → {Order Date, CartID}

All other relations in question 2 are in 3NF.

4. For each relation schema in your model that is in 3NF but not in BCNF, either rewrite the relation schema to BCNF or provide a short justification for why this relation should be an exception to the rule of putting relations into BCNF.

All relations listed above are in BCNF because for every functional dependency is dependent only on the candidate keys for every relation.

5. For your database, propose at least two interesting views that can be built from your relations. These views must involve joining at least two tables together each and must include some kind of aggregation in the view. Each view must also be able to be described by a one or two sentence description in plain English. Provide the code for constructing your views along with the English language description of what the view is supposed to be providing.

Finds the total number of IP Items purchased by each buyer.

```
CREATE VIEW [Total IP Purchases By Each Buyer] AS
SELECT BUYER.Account_number, SUM(SHOP_PRODUCT.Quantity)
FROM (((BUYER NATURAL JOIN ORDERS) NATURAL JOIN SHOPPING_CART) NATURAL
JOIN SHOP_PRODUCT)
WHERE SHOPPING_CART.purchased = TRUE
GROUP BY account_number
```

Finds the total number of money each buyer spent.

```
CREATE VIEW [Total spent by each buyer] AS
SELECT SUM(PRODUCT.price * SHOP_PRODUCT.Quantity) AS total
FROM BUYER, ORDERS, SHOPPING_CART, SHOP_PRODUCT, PRODUCT
WHERE BUYER.Account_number = ORDERS.Account_number AND ORDERS.CartID =
SHOPPING_CART.CartID AND SHOPPING_CART.CartID = SHOP_PRODUCT.CartID AND
PRODUCT.ProductID = SHOP_PRODUCT.ProductID AND SHOPPING_CART.purchased =
TRUE
GROUP BY BUYER.Account_Number
```

6. Description of two indexes that you want to implement in your DB. Explain their purpose and what you want to achieve by implementing them. Explain what type of indexing would be most appropriate for each one of them (Clustering, Hash, or B-tree) and why.

Find the images that are created on a specific date. A hash-based index could be used because we are only indexing based on a single date.

Find all the products that cost more than $10. For this one, a tree-based index should be used because we have a range based query.

7. Two sample transactions that you want to establish in your DB. Clearly document their purpose and function. Include the sample SQL code for each transaction. Each transaction should include read and/or write operations on at least two tables, with appropriate error and constraint checks and responses.

Adding account: Used to add an account to the database. If the account type is a buyer, the account will be added to the Buyer table. If the account type is a seller, the account will be added to the Seller table.

```
BEGIN TRANSACTION add_account
    INSERT INTO ACCOUNT (Account_number, Username, Type, Address, Karma_points,
    Phone_number, Name) VALUES (1, 'abc@gmail.com', 'Buyer', '40 Fremont Street
    Vicksburg, MS 39180', 18, '939-261-5642', 'Kadie Wheatley');
    //Inserts a new account
        IF error THEN GO TO UNDO; END IF;
    INSERT INTO BUYER (Account_number) VALUES (1);
        IF error THEN GO TO UNDO; END IF;

        COMMIT;
        GO TO FINISH;
    UNDO:
        ROLLBACK;
    FINISH:
END TRANSACTION;
```

Purchase cart: Used to add an order to the database. If the buyer purchased the product, the purchase of Shopping_cart will be changed to true and the number of products purchased will be subtracted from the quantity of the product.

```
BEGIN TRANSACTION purchase_cart
    INSERT INTO ORDERS (Order Number, Account Number, CartID, Order Date)
     VALUES (3, 1, 1, 10/7/2020);
    // Inserts new order
        IF error THEN GO TO UNDO; END IF;
        UPDATE SHOPPING_CART SET purchased = TRUE WHERE CartID = 1;
        // Updates the purchased of the product to True
            IF error THEN GO TO UNDO; END IF;
                UPDATE PRODUCT SET Quantity = Quantity - 1 WHERE
                SHOPPING_CART.CartID = SHOP_PRODUCT.CartID AND
                PRODUCT.ProductID = SHOP_PRODUCT.ProductID AND
                SHOPPING_CART.purchased = TRUE AND
                SHOPPING_CART.CartID = 1;
                IF error THEN GO TO UNDO; END IF;
        COMMIT;
        GO TO FINISH;
    UNDO:
        ROLLBACK;
    FINISH:
END TRANSACTION;
```