

WolfWR Management System

For WolfCity wholesale-store chain

Cole Sanders, Jonas Trepanier, Ishan Mistry, Manav Patel

Assumptions:

1. Memberships are only cancelled if payments aren't made on time.
2. A transaction detail is specific to one item type.
3. Staff who quit or are fired have their information deleted from the database.
4. When a customer signs up, they can use their membership at any store.
5. Returns are handled by warehouse operators. Warehouse operators update the merchandise inventory to reflect the returned items.
6. Items are returned at the same store they were purchased from.
7. If a customer wants to change their membership level they go through the sign up process with the registrar again.
8. Cashback rewards are given for purchases in the last calendar year.
9. Cashback rewards for the previous year are given to customers who have a membership level of platinum on Jan 1st of the current year.
10. Each staff member works at exactly one store.
11. Transferring merchandise between stores only happens once a day.
12. Bills are always generated with the amount owed already calculated.
13. When a product is transferred between two stores the entire quantity is transferred.
14. The jobTitle attribute refers to a staff member's role: warehouse operator, billing staff, cashier, etc.
15. Stores may exist in multiple countries.
16. Each merchandise can only have up to one discount applied at a time.
17. Each discount applies to exactly one merchandise.
18. Customer growth reports report the number of customers that have signed up in a time period.
19. Sales growth reports report the total sales per day over a given time period and are ordered by date.
20. Two people can share an email.

1. Problem Statement

The task we are presented with is to design and implement a database and database management system for a wholesale-store chain with the aim of creating a simple and comprehensive way to store, edit, delete, and organize the relevant data the chain needs to operate. This includes, among other tasks, maintaining and updating inventory and staff records, tracking merchandise movement across different stores, calculating discounts and rewards for certain tiers of store members, and generating cumulative reports on customer and sales growth. The reason a database is proposed as the optimal means of addressing these issues, instead of an assortment of files, is that a database will allow an easier way to manage the vast amount of data a wholesale store is likely to have. Costco, another wholesale corporation, operated 616 warehouses in the United States alone ([US Securities and Exchange Commission](#)). With an

operation of comparable size, it would be a challenge to locate individual records for members and merchandise if they were stored in a series of files. Additionally, a large number of users, including cashiers, billing and warehouse staff, and registrars will all be utilizing the data. If they attempt to modify the data concurrently in a file system, entries can be overwritten and lost. A database provides protections against this, allowing the large amount of users to utilize the large amounts of data efficiently and safely.

2. Intended Users:

All the different kinds of users that would be interacting with the database

- a. **Billing Staff:** The billing staff calculates and collects inventory bills. They are also responsible for collecting checks from customers to maintain their active subscriptions.
- b. **Registration Operators:** They are responsible for signing up new customers and changing membership levels. Each member in the system would have a membership status.
- c. **Warehouse operators:** They are responsible for maintaining the inventory and ordering new items as needed. Sometimes they would also have to initiate transfers between sister stores. The buying, selling, and sharing affect the status of warehouse items. They would be responsible for ensuring that the database is updated with the new stocks and a second level of verification that entries in the database match the actual stock. They also update the stock in the case of returns. They generate merchandise stock reports.
- d. **Cashier:** Cashiers will process transactions with customers. They would be responsible for recording customer information and all the details of a transaction. They will also handle transactions with negative value in the case of returns.

3. Five Main Entities:

- a. **Store** - store ID, manager ID, store address, phone number
- b. **Staff**- staff ID, name , age, home address, job title, phone number, email address, time of employment
- c. **Products** - product ID, product name, quantity in stock, buy price, market price, production date, expiration date, supplier ID
- d. **Transactions** - transaction ID, store id, customer id, cashier id, purchase date, product list, total price
- e. **ClubMembers** - member ID, first name, last name, membership level, email address, phone number, home address, active status

4. Tasks and Operations- Realistic Situations:

Situation 1: A customer comes to the store to return an unused item they decided they didn't want. Update the inventory records to reflect the addition of the returned item.

Situation 2: A store's management wants to do an end of year review of sales for their store. Generate a report of the total sales for that store over last year and for the last four preceding years for comparison.

5. Application Program Interfaces:

- **Information Processing:**

- enterTransactionInfo(transactionID, storeID, memberID, cashierID, purchaseDate, productList, totalPrice)
Return confirmation
- updateTransactionInfo(transactionID, storeID, memberID, cashierID, purchaseDate, productList, totalPrice)
Return confirmation
* If NULL value for any of the fields, then they will not be updated
- deleteTransactionInfo(transactionID)
Return confirmation
- enterStoreInfo(storeID, managerID, storeAddress, phoneNumber)
return confirmation
- updateStoreInfo(storeID, managerID, storeAddress, phoneNumber)
return confirmation
* If NULL value for any of the fields, then they will not be updated
- deleteStoreInfo(storeID)
return confirmation
- enterMemberInfo(memberID, firstName, lastName, membershipLevel, email, phone, homeAddress, activeStatus)
return confirmation
- updateMemberInfo(memberID, firstName, lastName, membershipLevel, email, phone, homeAddress, activeStatus)
return confirmation
* If NULL value for any of the fields, then they will not be updated
- deleteMemberInfo(memberID)
return confirmation
- enterStaffInfo(staffID, storeID, name, age, homeAddress, jobTitle, phoneNumber, emailAddress, timeOfEmployment)
return confirmation
- updateStaffInfo(staffID, storeID, name, age, homeAddress, jobTitle, phoneNumber, emailAddress, timeOfEmployment)
return confirmation
* If NULL value for any of the fields, then they will not be updated
- deleteStaffInfo(staffID)
- enterSupplierInfo(supplierID, supplierName, phone, emailAddress, location)
return confirmation
- updateSupplierInfo(supplierID, supplierName, phone, emailAddress, location)

- return confirmation
 - * If NULL value for any of the fields, then they will not be updated
 - deleteSupplierInfo(supplierID)
 - return confirmation
 - enterDiscountInfo(discountID, productID, discountDetails, validStartDate, validEndDate)
 - return confirmation
 - updateDiscountInfo(discountID, productID, discountDetails, validStartDate, validEndDate)
 - return confirmation
 - * If NULL value for any of the fields, then they will not be updated
 - deleteDiscountInfo(discountID)
 - return confirmation
- **Maintaining Inventory Records:**
 - createNewProduct(productID, productName, quantityInStock, buyPrice, marketPrice, productionDate, expirationDate, supplierID)
 - return true if valid and false otherwise.
 - * If any value for any of the fields are invalid, it returns false
 - receiveNewShipment(productID, quantity, arrivalDate, supplierID)
 - return confirmation
 - updateInventoryOnReturn(productID, returnQuantity)
 - Return confirmation
 - transferProductBetweenStores(productID, quantity, fromStoreID, toStoreID)
 - return confirmation
 - listStoreInventory(storeID)
 - return list of products
- **Maintaining Billing and Transaction Records:**
 - generateBill(supplierID, amountOwed)
 - Return confirmation if the bill is successfully created.
 - editBill(billID, newAmount)
 - Return confirmation if the bill is successfully edited.
 - deleteBill(billID)
 - Return Confirmation if the bill is successfully deleted.
 - createRewardCheck(memberID, year)
 - Return confirmation if the check is successfully created.
 - checkIfOnSale(productID, purchaseDate)
 - Return true if the product is on sale at the input date.
 - Return false if not.
 - getTotalPrice(productID, purchaseDate)
 - Returns the price of the product as of the purchase date, accounting for any current sales.
 - UpdateTransactionWithTotalPrice(TransactionID, totalPrice)
 - Returns confirmation if the totalPrice field in the Transaction is updated.
- **Reports:**

- generateSalesReportByDay(day, month, year)
Returns records of all sales that occurred on the input date.
- generateSalesReportByMonth(month, year)
Return records of all sales that occurred on the given month in the given year.
- generateSalesReportByYear(year)
Return records of all sales that occurred in the given year.
- salesGrowthReport(storeID, startDate, endDate)
Returns the growth in sales of a specific store over the specific dates.
- merchStockReportByProduct(productID, date)
Returns a list of each store and their stock of the particular product at the specified date.
- merchStockReportByStore(storeID, date)
Returns a list of each merchandise and their stock carried by a particular store at the specified date.
- customerGrowthByMonth(month, year)
Returns the growth in members over the specific month in the specific year.
- customerGrowthByYear(year)
Returns the growth in members over the specific year.
- customerPurchaseHistory(memberID, startDate, endDate)
Returns a list of records showing every transaction made by the customer over the course of the specified period.

6. Descriptions of Views:

Cashier: Cashiers can access information on stores, cashier staff, discounts, merchandise, transactions, transaction details, and club members. They will need one table in their view: showing the store, customer identification, cashier identification, and the details of each transaction including all the merchandise purchased, and any discount information.


Warehouse operators: Warehouse operators can access warehouse staff, merchandise, supplier, and store data. They will need one view: one with a list of all merchandise from all stores to update inventory and perform transfers.


Registration operators: Registration operators will only have access to registration staff, club members and store data. They will view one table with members, store, staff and sign up data.

Billing Staff: Billing staff can access billing staff, store, merchandise, transaction, club members, bills, and supplier data. They will need two tables in their view: one with customer,

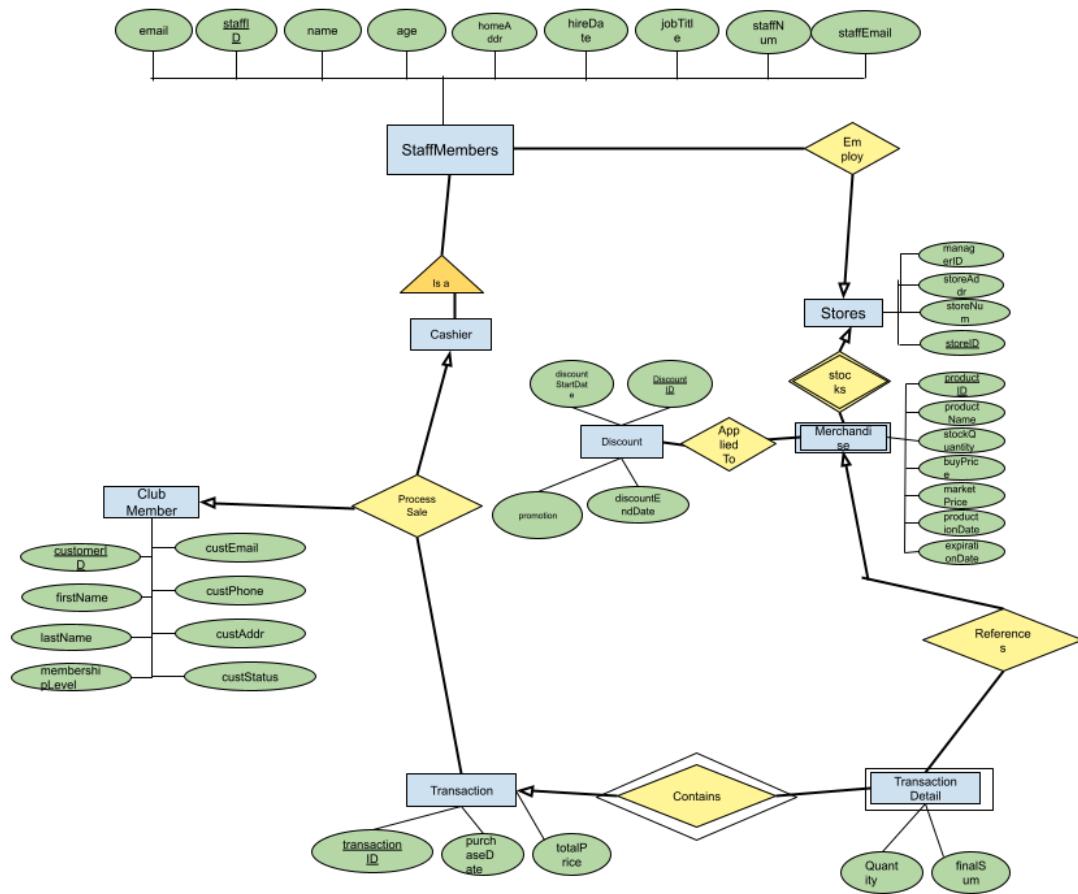
billing staff, store, and transaction information and one with supplier, billing staff, merchandise, store, and bill information.

7. Local E/R Diagrams:

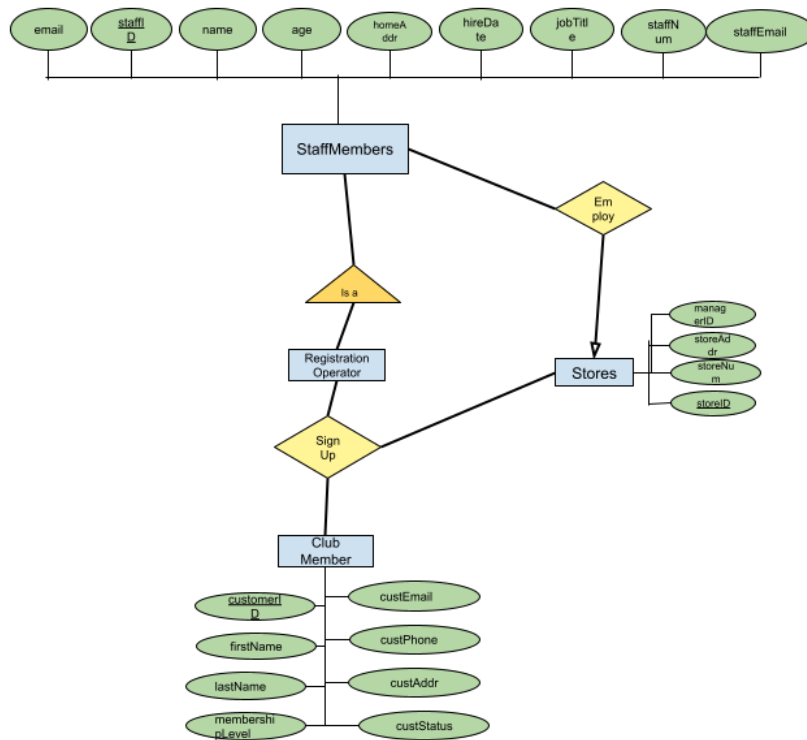
Note: The open arrows (with points not all black) represent the one-to-exactly one relationships (ex. ).

The closed arrows represent the zero or one relationship (ex. .

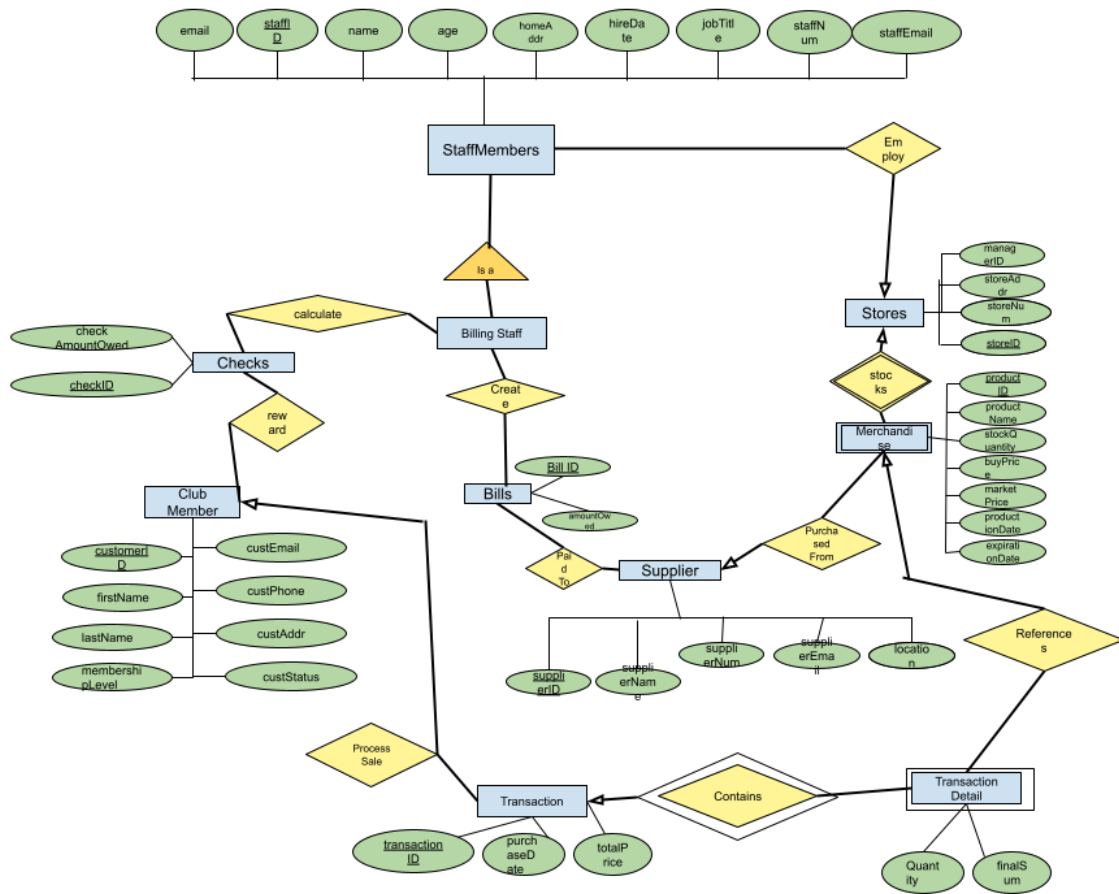
Cashier's View:



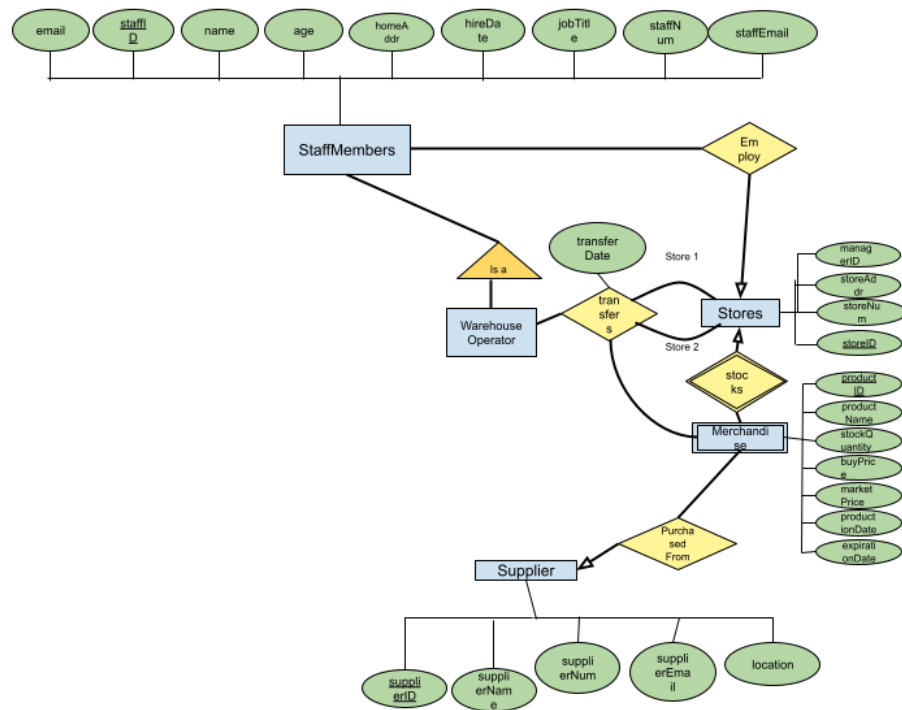
Registration Operator's View:



Billing Operator's View:



Warehouse Operator:



8. Description of Local E/R Diagrams:

- a. Each staff member is uniquely identified by a Staff_ID. Every Staff Member also has a Name, Age, Home Address, Hire Date, Job Title, Staff Number, and Email.
- b. Staff Members can be categorized into a) Registration Operator (handles user sign-ups), Cashier (manages transactions), Billing staff (manages billing and payments), and Warehouse Operator (manages inventory)
- c. Each Staff Member is **employed** at a Store.
- d. Each Store is uniquely identified by a Store_ID and has a Store Address and Store Number. Stores stock various Merchandise. Each Product has a Product ID, Name, Stock Quantity, Market Price, Buy Price, Production Date, and Expiration Date.
- e. Stores can **transfer** products between each other. This is reflected in the set of profiles(Store 1, Store 2) related to the Warehouse Operator by the transfer relationship.
- f. Stores can offer promotions in the form of **Discounts** with a **Discount ID, Start Date, End Date and Promotion**. We have modeled this as a separate entity(kind of merchandise) - Discount Merchandise. The captured interval reflects the seasonality of discounts/promotions and offers on specific merchandise. Promotion represents the percentage discount available for that product.
- g. Products are **purchased from Suppliers**. Each **Supplier** has a Supplier ID, Name, Location, Email, and Phone Number. The billing staff pays bills to the suppliers based on the products purchased from them.
- h. Suppliers **provide** products to the **Stores**. Each product is purchased from exactly one supplier, but a supplier can provide multiple products. Each Bill has a Bill ID and Amount Owed. Each bill is associated with one supplier, but a supplier may have many bills over time.
- i. Merchandise is sent by a particular supplier who we assume is the manufacturer because we are dealing with wholesale products.
- j. Each **Customer** has a Customer ID, First Name, Last Name, Email, Phone, Address, Status, and Membership Level. Customers **sign up** for the store's **membership** by approaching **the Registration Operator**. Customers enroll in the store's membership program, which we represent as a club membership. Every customer must be linked to a membership status.
- k. Customers **buy** products, creating **transactions** in our database. These transactions are created and processed by **cashiers**. This multi-way relationship can be observed in the diagram through the process sale relationship. Hence all of these entities together determine a transaction. Each **Transaction** has a Transaction ID, Staff ID, Purchase Date, and Total Price.
- l. To maintain a list of transactions for various merchandise, we have an entity dependent on the transaction entity that contains the total amount of any

merchandise bought by a customer, and the final sum after discounts and quality can be calculated.

- m. Each transaction can, in turn, be associated with multiple merchandise items. As is typical with weak entity sets, the transaction details entity relies on the primary keys of both transaction and merchandise for identification.
- n. Customers **purchase** products from the store. Each transaction is handled by a single cashier (staff member), but a staff member can handle multiple transactions.

9. Local Relational Schemas:

Cashier View:

StaffMembers(staffID, email, name, age, homeAddr, hireDate, jobTitle, staffNum, staffEmail, storeID)

Stores(storeID, storeNum, storeAddr, managerID)

ClubMember(customerID, firstName, lastName, membershipLevel, custEmail, custPhone, custAddr, custStatus)

Transaction(transactionID, purchaseDate, totalPrice, customerID, staffID)

TransactionDetail(transactionID, quantity, finalSum, storeID, productID)

Merchandise(storeID, productID, productName, stockQuantity, buyPrice, marketPrice, productionDate, expirationDate)

Discount(discountID, discountStartDate, discountEndDate, promotion)

AppliedTo(DiscountID, ProductID, StoreID)

Registration Operator's View:

StaffMembers(staffID, email, name, age, homeAddr, hireDate, jobTitle, staffNum, staffEmail, storeID)

Stores(storeID, storeNum, storeAddr, managerID)

ClubMember(customerID, firstName, lastName, membershipLevel, custEmail, custPhone, custAddr, custStatus)

SignUp(storeID, customerID, staffID)

Billing Operator's View:

StaffMembers(staffID, email, name, age, homeAddr, hireDate, jobTitle, staffNum, staffEmail, storeID)

ClubMember(customerID, firstName, lastName, membershipLevel, custEmail, custPhone, custAddr, custStatus)

Bills(billID, amountOwed)

Checks(checkID, checkAmountOwed)

Supplier(supplierID, supplierName, supplierNum, supplierEmail, location)

Transaction(transactionID, purchaseDate, totalPrice, customerID)

TransactionDetail(transactionID, quantity, finalSum, storeID, productID)

Merchandise(storeID, productID, productName, stockQuantity, buyPrice, marketPrice, productionDate, expirationDate, supplierID)

Stores(storeID, storeNum, storeAddr, managerID)

Calculate(checkID, staffID)

Reward(checkID, customerID)

Create(staffID, billID)

PaidTo(billID, supplierID)

Warehouse Operator's View:

StaffMembers(staffID, email, name, age, homeAddr, hireDate, jobTitle, staffNum, staffEmail, storeID)

Stores(storeID, storeNum, storeAddr, managerID)

Merchandise(storeID, productID, productName, stockQuantity, buyPrice, marketPrice, productionDate, expirationDate, supplierID)

Supplier(supplierID, supplierName, supplierNum, supplierEmail, location)

Transfers(staffID, store1ID, store2ID, product1ID, product2ID, transferDate)

10. Local Relational Schema Documentation:

For the local relation schema translation, two main design decisions were made. The remaining relations were generated mechanically.

Translating Hierarchies:

Firstly, the hierarchy of StaffMembers and Cashier was translated using the NULLS approach. This is because there are no new attributes in the Cashier entity set. Thus the NULLS approach is the best fit because it minimizes duplicating information without wasting storage space. There will be no null fields when the schema is translated this way. This same process and reasoning was used for all relations that were a subclass of StaffMembers including RegistrationOperator, BillingStaff, and WarehouseOperator.

Combining Many to One Relationships:

Secondly, many to one relationships were combined with the relation on the many side of the arrow. This reduces the number of schemas, increases efficiency of querying, and reduces the amount of space the system needs.

For Cashier's View:

This includes the relationships Employ, ProcessSale, and References. This was done by adding the attribute storeID to StaffMembers, the attributes customerID and staffID to Transaction, and the attributes productID and storeID to TransactionDetails.

For Registration Operator's View:

This includes the relationship Employ. The attribute storeID was added to StaffMembers.

For Billing Operator's View:

This includes the relationships ProcessSale, References, and PurchasedFrom. The attribute customerID was added to Transaction, the attributes storeID and productID were added to TransactionDetail, and supplierID was added to Merchandise.

For Warehouse Operator's View:

This includes the relationships Employ and PurchasedFrom. The attribute storeID was added to StaffMembers and the attribute supplierID was added to Merchandise.

The Remaining Translation:

The remaining entity sets and relationships were translated in the mechanical way. Entities became their own schema with all their attributes. Relationships became their own schema with the keys of the entity sets they were connected to as well as their own attributes. Weak entity sets

became their own schema with all their attributes and the additional keys needed to uniquely identify them. Supporting relationships for weak entity sets did not become schemas as they only carry redundant information.

11. Global Relational Database Schema

StaffMembers(staffID, email, name, age, homeAddr, hireDate, jobTitle, staffNum, storeID)

staffID -> **staffID, email, name, age, homeAddr, hireDate, jobTitle, staffNum, storeID** holds because each StaffID is unique, and identifies an individual that has a name, email, age, home address, hire data, job title, phone number, and the store they belong to. In other words, because staffID is the key for this relation. The left hand side is a superkey so this relation does not violate BCNF. If we try to take any combination of other attributes, it would limit the possibilities of the database. For instance if we take email ->, home address ->, or phone number -> do not hold because it's possible for two staff members to share the same email, house, or home phone number.

Stores(storeID, storeNum, storeAddr, managerID)

storeID -> **storeID, storeNum, storeAddr, managerID** holds because each store ID uniquely identifies a store, determining its number, address, and manager ID. Because storeID is a superkey, this does not violate BCNF.

managerID -> **storeID, storeNum, storeAddr, managerID** holds because each store has exactly one manager and a manager can only manage one store. The left hand side is a superkey so this does not violate BCNF.

storeAddr -> **storeNum, storeAddr, managerID, storeID** holds because only one store can be at a physical address at a time. The left hand side is a superkey so this does not violate BCNF.

storeNum -> **managerID, storeAddr, managerID, storeID** does not hold because country code is not captured in the phone numbers of stores, thus two stores in different countries could have the same storeNum.

We decided to use **storeID** as the primary key of our global relational database schema.

ClubMembers(customerID, firstName, lastName, membershipLevel, custEmail, custPhone, custAddr, storeID, custStatus, staffID, signUpDate)

customerID -> **customerID, firstName, lastName, membershipLevel, custEmail, custPhone, custAddr, storeID, staffID, custStatus, signUpDate** holds because each club member with all their respective attributes has a unique customer ID (customerID). CustomerID is the key for this relation. The left hand side is a superkey so this does not violate BCNF.

Customer email ->, phone ->, and address -> do not hold because two customers could possibly share an email, home phone, or address.

staffID -> storeID, staffID violation of BCNF

This holds because each staff only works at one store. This violates BCNF because the left hand side is not a superkey.

Decomposition:

A(customerID, firstName, lastName, membershipLevel, custEmail, custPhone, custAddr, custStatus, staffID)

B(staffID, storeID)

A table that relates staffID to store ID would be redundant given the StaffMembers table. Thus the resulting schema is:

ClubMember(customerID, firstName, lastName, membershipLevel, custEmail, custPhone, custAddr, custStatus, staffID)

Transactions(transactionID, purchaseDate, totalPrice, customerID, staffID)

transactionID -> transactionID, purchaseDate, totalPrice, customerID, staffID holds because each transaction with all their respective attributes is unique, a transactionID. This is also the key for this relation. The left hand side is a superkey so this does not violate BCNF.

TransactionDetail(transactionID, quantity, finalSum, storeID, productID)

transactionID, productID -> transactionID, productID, quantity, finalSum, storeID holds because transactionID, productID is the key for this relation. The left hand side is a superkey so this does not violate BCNF.

productID -> does not hold because productID is a weak entity set and thus has different numbering for each store.

transactionID -> does not hold because transactionID can be repeated at different stores and thus cannot functionally determine the other attributes.

Merchandise(productID, storeID, productName, stockQuantity, buyPrice, marketPrice, productionDate, expirationDate, supplierID)

productID, storeID -> productID, storeID, productName, stockQuantity, buyPrice, marketPrice, productionDate, expirationDate, supplierID holds because productID and storeID is the key for this relation. The left hand side is a superkey so this does not violate BCNF.

supplierID -> does not hold because suppliers can supply multiple different products to many different stores who each have their own numbering system.

productID -> does not hold because productID is a weak entity set and thus has different numbering for each store. It needs storeID in order to determine the other attributes.

storeID -> does not hold because stores can have multiple different merchandise and therefore cannot determine all other attributes.

productName ->, **stockQuantity** ->, **buyPrice** ->, **marketPrice** ->, **productionDate** ->, **expirationDate** -> does not hold because these attributes can be shared among different merchandises.

Discounts(discountID, discountStartDate, discountEndDate, promotion, productID, storeID)

discountID -> **discountID, discountStartDate, discountEndDate, promotion, productID, storeID** holds because each discount with their respective attributes can be determined by a unique discountID. DiscountID is therefore a key for this relation. The left hand side is a superkey so this does not violate BCNF.

promotion -> does not hold because there can be many discounts with the same promotions but different IDs or dates.

productID, storeID -> **discountID, discountStartDate, discountEndDate, promotion, productID, storeID** holds because there is a one to one relationship between discounts and merchandise. Thus the unique identifying attributes of a product determine the values of the discount applied to it. The left hand side is a superkey so this is BCNF and therefore 3NF.

We decided to use the minimal key which is discountID as the primary key for our global schema.

Bills(billID, amountOwed, status, staffID, supplierID)

billID -> **billID, amountOwed, status, staffID, supplierID** holds because each bill with their respective attributes can be determined by a unique billID. Therefore billID is the key for this relation. The left hand side is a superkey so this is BCNF and therefore 3NF.

staffID -> does not hold because billing staff make many bills to many suppliers.

supplierID -> does not hold because they may supply many different products to many different stores and be billed separately.

Rewards(rewardID, checkAmountOwed, staffID, customerID)

rewardID -> **checkID, checkAmountOwed, staffID, customerID** holds because each reward with their respective attributes can be determined by a unique rewardID. Therefore rewardID is the key for this relation. The left hand side is a superkey so this is BCNF and therefore 3NF.

Suppliers(supplierID, supplierName, supplierNum, supplierEmail, location)

supplierID -> **supplierID, supplierName, supplierNum, supplierEmail, location** holds because each supplier with their respective attributes can be determined by a unique supplierID. Therefore supplierID is the key for this relation. The left hand side is a superkey, so this is BCNF and therefore 3NF.

Transfers(store1ID, store2ID, product1ID, product2ID, transferDate, staffID)
staffID, store1ID, store2ID, product1ID, product2ID -> **staffID, store1ID, store2ID, product1ID, product2ID, transferDate** holds because we need the id of the product that is transferred at both stores, both storeID's, and the staffID to determine all attributes.

store1ID -> **staffID** does not hold because staff from any store can initiate the transfer.

store2ID -> **staffID** does not hold because staff from any store can initiate the transfer.

12. Design For Global Schema:

Design Decision for Global Schema:

Our entity sets were converted to relations primarily by using the Nulls approach for hierarchies in order to save table space by not having to keep track of all parent attributes for each of the descendent entity sets of StaffMembers. This means that subclasses like Billing Staff and Cashiers were folded into the StaffMembers parent class.

The many-to-one relationships, such as that of Transactions to Customers, were resolved by utilizing the keys of the "one" entity in the "many" entity, allowing for more efficient queries.

In addition, certain relations, specifically Rewards, were constructed by combining keys and attributes from entity sets in the diagram, in order to streamline the ability to query information about them.

StaffMembers(staffId, email, staffNum, storeID, name, age, homeAddr, hireDate, jobTitle)

- staffID is the primary key.
- storeID is a foreign key retrieved from the Stores entity set.
- staffEmail, hireDate, staffNum, storeID, name, age, homeAddr, jobTitle are not allowed to be NULL.
- jobTitle must be either 'Manager', 'Cashier', 'BillingStaff', 'WarehouseOperator', or 'RegistrationOperator'

Stores(storeID, storeNum, storeAddr, managerID)

- storeID is the primary key.
- storeNum and storeAddr are not allowed to be NULL.
- managerID is a foreign key retrieved from the StaffMembers entity set.

- managerID is allowed to be NULL, for instance if a store has been built but not yet started hiring workers.
- This is implemented using ALTER TABLE in the schema.

Merchandise(storeID, productID, productName, stockQuantity, buyPrice, marketPrice, productionDate, expirationDate, supplierID)

- storeID and productID are the primary keys.
- storeID is a foreign key retrieved from the Stores table.
- supplierID is a foreign key retrieved from the Supplier table.
- productName, stockQuantity, buyPrice, marketPrice, and productionDate are not allowed to be NULL.
- stockQuantity, buyPrice, marketPrice all must be an integer greater than or equal to zero.
- supplierID and storeID are not allowed to be NULL.
- expirationDate is allowed to be NULL in case of a merchandise item that does not have an expiration date such as a piece of furniture or clothing.

Discounts(discountID, discountStartDate, discountEndDate, promotion, productID, storeID)

- discountID is the primary key.
- productID and storeID are foreign keys referencing the Merchandise and Stores tables respectively. They are not allowed to be NULL.
- discountStartDate, discountEndDate, promotion are not allowed to be NULL.

Bills(billID, amountOwed, status, staffID, supplierID)

- billID is the primary key.
- amountOwed is not allowed to be NULL.
- Status is not allowed to be NULL, it must be either 'paid' or 'unpaid'
- staffID and supplierID are foreign keys retrieved from the StaffMembers and Suppliers table respectively

Rewards(checkID, checkAmountOwed, staffID, customerID)

- checkID is the primary key.
- customerID is a foreign key retrieved from the Customer entity set and can not be NULL.
- checkAmountOwed is not allowed to be NULL.
- staffID is a foreign key and is not allowed to be NULL

ClubMembers(customerID, firstName, lastName, membershipLevel, custEmail, custPhone, custAddr, custStatus, staffID, signUpDate)

- customerID is the primary key.
- firstName, lastName, membershipLevel, custEmail, custPhone, custAddr, custStatus, and signUpDate are not allowed to be NULL.
- staffID is a foreign key and is not allowed to be NULL.
- custStatus must be equivalent to: ('Active', 'Inactive', 'Suspended')

Suppliers(supplierID, supplierName, supplierNum, supplierEmail, location)

- supplierID is the primary key.
- supplierName, supplierNum, supplierEmail, location are not allowed to be NULL.

Transactions(transactionID, purchaseDate, totalPrice, customerID, staffID)

- transactionID is the primary key.
- customerID and staffID are foreign keys retrieved from the Customer and StaffMembers entity sets respectively.
- customerID and staffID are allowed to be NULL in case the staff member is fired or the customer's
- purchaseDate and totalPrice are not allowed to be NULL.

TransactionDetail(transactionID, quantity, finalSum, storeID, productID)

- transactionID, productID are the primary keys.
- transactionID, storeID, and productID are foreign keys retrieved from the Transaction, Store, and Merchandise entity sets respectively.
- quantity and finalSum are not allowed to be NULL.

Transfers(store1ID, store2ID, product1ID, product2ID, transferDate, staffID)

- store1ID, store2ID, product1ID, product2ID are all both primary keys and foreign keys, the first two referencing the Stores table and the latter two referencing the merchandise table
- staffID is a foreign key referencing the StaffMembers table.
- transferDate can not be NULL.

Base Relations

```
CREATE TABLE Stores (  
    storeID INT PRIMARY KEY,  
    storeNum VARCHAR(32) NOT NULL,  
    storeAddr VARCHAR(256) NOT NULL  
);
```

```
CREATE TABLE StaffMembers (  
    StaffID INT PRIMARY KEY,  
    Name VARCHAR(128) NOT NULL,  
    Age DATE NOT NULL,  
    homeAddr VARCHAR(128) NOT NULL,  
    hireDate DATE NOT NULL,  
    jobTitle VARCHAR(32) NOT NULL,
```

```
    staffNum VARCHAR(15) NOT NULL,  
    staffEmail VARCHAR(128) NOT NULL,  
    storeID INT NOT NULL,  
    FOREIGN KEY (storeID) REFERENCES Stores(storeID),  
    CHECK (jobTitle IN ('Manager', 'Cashier', 'BillingStaff',  
        'WarehouseOperator', 'RegistrationOperator'))  
);
```

```
ALTER TABLE Stores ADD managerID INT, ADD CONSTRAINT FK_Stores_Manager FOREIGN  
KEY (managerID) REFERENCES StaffMembers(staffID) ON DELETE SET NULL;
```

```
CREATE TABLE ClubMembers (  
    customerID INT PRIMARY KEY,  
    firstName VARCHAR(64) NOT NULL,  
    lastName VARCHAR(64) NOT NULL,  
    membershipLevel VARCHAR(32) NOT NULL,  
    custEmail VARCHAR(128) NOT NULL,  
    custPhone VARCHAR(15) NOT NULL,  
    custAddr VARCHAR(256) NOT NULL,  
    custStatus VARCHAR(32) NOT NULL CHECK (custStatus IN ('Active',  
        'Inactive', 'Suspended')),  
    staffID INT NOT NULL,  
    signUpDate DATE NOT NULL,  
    FOREIGN KEY (staffID) REFERENCES StaffMembers(staffID)  
);
```

```
CREATE TABLE Transactions (  
    transactionID INT PRIMARY KEY,  
    purchaseDate DATE NOT NULL,  
    totalPrice DECIMAL(10,2),  
    customerID INT,  
    staffID INT,  
    FOREIGN KEY (customerID) REFERENCES ClubMembers(customerID) ON DELETE  
    SET NULL,  
    FOREIGN KEY (staffID) REFERENCES StaffMembers(staffID) ON DELETE SET  
    NULL  
);
```

```
CREATE TABLE Suppliers (  
    supplierID INT PRIMARY KEY,  
    supplierName VARCHAR(128) NOT NULL,  
    supplierNum VARCHAR(15) NOT NULL UNIQUE,
```

```

        supplierEmail VARCHAR(128) NOT NULL UNIQUE,
        location VARCHAR(256) NOT NULL
    );

CREATE TABLE Merchandise (
    storeID INT NOT NULL,
    productID INT NOT NULL,
    productName VARCHAR(128) NOT NULL,
    stockQuantity INT NOT NULL CHECK (stockQuantity >= 0),
    buyPrice DECIMAL(10,2) NOT NULL CHECK (buyPrice >= 0),
    marketPrice DECIMAL(10,2) NOT NULL CHECK (marketPrice >= 0),
    productionDate DATE NOT NULL,
    expirationDate DATE,
    supplierID INT NOT NULL,
    FOREIGN KEY (storeID) REFERENCES Stores(storeID) ON DELETE CASCADE,
    FOREIGN KEY (supplierID) REFERENCES Suppliers(supplierID) ON DELETE CASCADE,
    PRIMARY KEY (storeID, productID)
);

CREATE TABLE Bills (
    billID INT PRIMARY KEY,
    amountOwed DECIMAL(10,2) NOT NULL,
    status VARCHAR(32) NOT NULL CHECK (status IN ('paid', 'unpaid')),
    staffID INT NOT NULL,
    supplierID INT NOT NULL,
    FOREIGN KEY (staffID) REFERENCES StaffMembers(staffID),
    FOREIGN KEY (supplierID) REFERENCES Suppliers(supplierID)
);

CREATE TABLE Rewards (
    rewardID INT PRIMARY KEY,
    checkAmountOwed DECIMAL(10,2) NOT NULL CHECK (checkAmountOwed > 0),
    staffID INT NOT NULL,
    customerID INT NOT NULL,
    FOREIGN KEY (staffID) REFERENCES StaffMembers(staffID),
    FOREIGN KEY (customerID) REFERENCES ClubMembers(customerID)
);

CREATE TABLE TransactionDetails (
    transactionID INT NOT NULL,
    productID INT NOT NULL,

```

```

    quantity INT NOT NULL CHECK (quantity > 0),
    finalSum DECIMAL(10,2) NOT NULL CHECK (finalSum >= 0),
    storeID INT NOT NULL,
    PRIMARY KEY (transactionID, productID),
    FOREIGN KEY (transactionID) REFERENCES Transactions(transactionID) ON
    DELETE CASCADE,
    FOREIGN KEY (storeID, productID) REFERENCES Merchandise(storeID,
    productID) ON DELETE CASCADE
);

```

```

CREATE TABLE Discounts (
    discountID INT PRIMARY KEY,
    productID INT NOT NULL,
    storeID INT NOT NULL,
    discountStartDate DATE NOT NULL,
    discountEndDate DATE NOT NULL,
    promotion DECIMAL(10,2) NOT NULL,
    FOREIGN KEY (storeID, productID) REFERENCES Merchandise(storeID,
productID)
    ON DELETE CASCADE
);

```

```

CREATE TABLE Transfers (
    store1ID INT NOT NULL,
    store2ID INT NOT NULL,
    product1ID INT NOT NULL,
    product2ID INT NOT NULL,
    transferDate DATE NOT NULL,
    staffID INT NOT NULL,
    PRIMARY KEY (store1ID, store2ID, product1ID, product2ID),
    FOREIGN KEY (store1ID, product1ID) REFERENCES Merchandise(storeID,
productID) ON DELETE CASCADE,
    FOREIGN KEY (store2ID, product2ID) REFERENCES Merchandise(storeID,
productID) ON DELETE CASCADE,
    FOREIGN KEY (staffID) REFERENCES StaffMembers(staffID) ON DELETE CASCADE
);

```

SQL > SELECT * FROM Stores;

```

+-----+-----+-----+-----+
| storeID | storeNum | storeAddr | managerID |
+-----+-----+-----+-----+

```


1	101-101-1001	100 Main St, Raleigh NC	1
2	202-202-2002	200 Center St, Durham NC	6
3	303-303-3003	300 High St, Charlotte NC	NULL
4	404-404-4004	400 Lakeview Dr, Asheville NC	NULL
5	505-505-5005	500 Park Ave, Greensboro NC	NULL

5 rows in set (0.0034 sec)

SQL > SELECT * FROM StaffMembers;

StaffID	Name	Age	homeAddr	hireDate	jobTitle	staffNum	staffEmail	storeID
1	Alice Johnson	1990-05-12	123 Elm St, Raleigh NC	2023-01-01	Manager	111-111-1111	alice@example.com	1
2	Bob Smith	1985-07-19	456 Pine St, Durham NC	2022-06-15	Cashier	222-222-2222	bob@example.com	2
3	Charlie Davis	1992-08-30	789 Oak St, Charlotte NC	2021-09-10	BillingStaff	333-333-3333	charlie@example.com	3
4	Diana King	1998-11-25	101 Maple St, Asheville NC	2024-02-20	WarehouseOperator	444-444-4444	diana@example.com	4
5	Ethan Wright	1995-04-15	555 Birch St, Greensboro NC	2020-05-22	RegistrationOperator	555-555-5555	ethan@example.com	5
6	Fiona Adams	1987-09-03	666 Cedar St, Winston-Salem NC	2019-11-10	Manager	666-666-6666	fiona@example.com	2

6 rows in set (0.0036 sec)

SQL > SELECT * FROM ClubMembers;

customerID	firstName	lastName	membershipLevel	custEmail	custPhone	custAddr	custStatus	staffID	signUpDate
------------	-----------	----------	-----------------	-----------	-----------	----------	------------	---------	------------

1	John	Doe	Gold	johndoe@gmail.com		
555-555-5551	111 John Rd,	Raleigh NC	Active	1	2024-01-01	
2	Jane	Smith	Silver	janesmith@gmail.com		
555-555-5552	222 Jane Rd,	Durham NC	Active	2	2024-02-15	
3	Mike	Brown	Platinum	mikebrown@gmail.com		
555-555-5553	333 Mike Rd,	Charlotte NC	Inactive	3	2023-06-10	
4	Sara	Lee	Gold	saralee@gmail.com		
555-555-5554	444 Sara Rd,	Asheville NC	Suspended	4	2022-09-05	
5	Tom	Wilson	Silver	tomwilson@gmail.com		
555-555-5555	555 Tom Rd,	Greensboro NC	Active	5	2024-03-01	
6	Emma	White	Platinum	emmawhite@gmail.com		
555-555-5556	666 Emma Rd,	Winston-Salem NC	Inactive	3	2021-05-20	
7	Jack	Taylor	Gold	jacktaylor@gmail.com		
555-555-5557	777 Jack Rd,	Charlotte NC	Active	4	2020-08-15	

7 rows in set (0.0038 sec)

SQL > SELECT * FROM Transactions;

transactionID	purchaseDate	totalPrice	customerID	staffID	
1	2024-03-01	150.75	1	1	
2	2024-03-02	230.40	2	2	
3	2024-03-03	89.99	3	3	
4	2024-03-04	300.25	4	4	
5	2024-03-05	175.00	5	5	
6	2024-03-06	225.50	6	6	

6 rows in set (0.0032 sec)

SQL > SELECT * FROM Suppliers;

supplierID	supplierName	supplierNum	
supplierEmail	location		

```

|          1 | Fresh Foods Co.          | 666-666-6661 |
freshfoods@gmail.com      | 100 Market St, Raleigh NC      |
|          2 | Healthy Supplies Ltd.    | 666-666-6662 |
healthysupplies@gmail.com | 200 Wholesale Ave, Durham NC   |
|          3 | Quality Goods Inc.       | 666-666-6663 |
qualitygoods@gmail.com    | 300 Retail Rd, Charlotte NC    |
|          4 | Premium Products LLC     | 666-666-6664 |
premiumproducts@gmail.com | 400 Distributor Ln, Asheville NC|
|          5 | Budget Essentials        | 666-666-6665 |
budgetessentials@gmail.com| 500 Outlet Rd, Greensboro NC   |
+-----+-----+-----+-----+
-----+-----+
5 rows in set (0.0032 sec)

```

SQL > SELECT * FROM Merchandise;

```

+-----+-----+-----+-----+-----+
-----+-----+-----+-----+
| storeID | productID | productName          | stockQuantity | buyPrice |
marketPrice | productionDate | expirationDate | supplierID |
+-----+-----+-----+-----+-----+
-----+-----+-----+-----+
|          1 |          1 | Organic Apples       |          50 |    1.50 |
2.50 | 2024-01-01      | 2024-06-01      |          1 |
|          1 |          2 | Whole Wheat Bread    |          30 |    2.00 |
3.50 | 2024-02-01      | 2024-05-01      |          2 |
|          2 |          3 | Milk 2%              |          40 |    3.00 |
4.50 | 2024-01-15      | 2024-04-15      |          3 |
|          2 |          4 | Cheddar Cheese       |          25 |    5.00 |
7.50 | 2024-02-20      | 2024-08-20      |          4 |
|          3 |          5 | Tomato Sauce         |          35 |    1.75 |
3.00 | 2024-01-05      | 2024-06-10      |          5 |
|          3 |          6 | Canned Beans         |          60 |    1.25 |
2.25 | 2024-02-10      | 2025-02-10      |          1 |
|          4 |          7 | Frozen Peas          |          45 |    2.50 |
4.00 | 2024-03-15      | 2025-03-15      |          2 |
|          5 |          8 | Rice 5lb             |          20 |    6.00 |
10.00 | 2024-04-01      | NULL             |          3 |

```

```

+-----+-----+-----+-----+-----+
-----+-----+-----+-----+
8 rows in set (0.0032 sec)

```

SQL > SELECT * FROM Bills;

```

+-----+-----+-----+-----+-----+
| billID | amountOwed | status | staffID | supplierID |
+-----+-----+-----+-----+-----+
|      1 |      500.00 | unpaid |      1 |          1 |
|      2 |      750.00 | paid   |      2 |          2 |
|      3 |     1200.00 | unpaid |      3 |          3 |
|      4 |      300.00 | paid   |      4 |          4 |
|      5 |      900.00 | unpaid |      5 |          5 |
+-----+-----+-----+-----+-----+
5 rows in set (0.0033 sec)

```

SQL > SELECT * FROM Rewards;

```

+-----+-----+-----+-----+-----+
| rewardID | checkAmountOwed | staffID | customerID |
+-----+-----+-----+-----+-----+
|      1 |          25.00 |      1 |          1 |
|      2 |          50.00 |      2 |          2 |
|      3 |          10.00 |      3 |          3 |
|      4 |          75.00 |      4 |          4 |
|      5 |          40.00 |      5 |          5 |
|      6 |          60.00 |      6 |          6 |
+-----+-----+-----+-----+-----+
6 rows in set (0.0034 sec)

```

SQL > SELECT * FROM TransactionDetails;

```

+-----+-----+-----+-----+-----+
| transactionID | productID | quantity | finalSum | storeID |
+-----+-----+-----+-----+-----+
|              1 |          1 |         3 |      7.50 |          1 |
|              1 |          2 |         2 |      7.00 |          1 |
|              2 |          3 |         1 |      4.50 |          2 |

```

	2	4	2	15.00	2
	3	5	4	12.00	3
	3	6	5	18.50	3
	4	7	3	10.00	4
	5	8	2	20.00	5

8 rows in set (0.0033 sec)

SQL > SELECT * FROM Discounts;

discountID	productID	storeID	discountStartDate	discountEndDate	promotion
1	1	1	2024-03-01	2024-04-01	0.80
2	2	1	2024-03-01	2024-04-01	0.85
3	3	2	2024-03-01	2024-04-01	0.90
4	4	2	2024-03-01	2024-04-01	0.75
5	5	3	2024-03-01	2024-04-01	0.70

5 rows in set (0.0032 sec)

SQL > SELECT * FROM Transfers;

store1ID	store2ID	product1ID	product2ID	transferDate	staffID
1	2	1	3	2024-03-05	1
1	3	2	5	2024-03-06	2
2	3	3	6	2024-03-07	3
3	4	5	7	2024-03-08	4
5	1	8	1	2024-03-09	5

5 rows in set (0.0033 sec)

SQL Queries

13.1

Information Processing:

Enter Store Info:

Assumption: There is a newly built store with the ID 1, the phone number 111-111-1111, the address of 111 Store Rd, Raleigh NC, but there has not been a manager assigned yet.

```
SQL > INSERT INTO Stores VALUES (1, '111-111-1111', '111  
Store Rd, Raleigh NC', NULL);
```

Query OK, 1 row affected (0.0049 sec)

Update Store Info:

Assumption: The store with ID 1 is changing its phone number to 111-111-1112.

```
SQL > UPDATE Stores SET storeNum = '111-111-1112' WHERE  
storeID = 1;
```

Query OK, 1 row affected (0.0051 sec)

Delete Store Info:

Assumption: The store with ID 1 has closed.

```
SQL > DELETE FROM Stores WHERE storeID = 1;
```

Query OK, 1 row affected (0.0051 sec)

Enter Customer Info:

Assumption: A customer has signed up with the customer ID of 1, the name John Doe, the Platinum membership level, the email JohnDoe@gmail.com, the phone number 211-111-1111, the address 111 John Rd, Raleigh NC, an active status, and was signed up by a staff member with staff ID 1 on 01-01-2024.

```
SQL > INSERT INTO ClubMembers VALUES (1, 'John', 'Doe',  
'Platinum', 'JohnDoe@gmail.com', '211-111-1111', '111 John  
Rd, Raleigh NC', 'Active', 1, '2024-01-01');
```

Query OK, 1 row affected (0.0047 sec)

Update Customer Info:

Assumption: A customer with the ID of 1 got married and wants to change their last name to Smith.

```
SQL > UPDATE ClubMembers SET lastName = 'Smith' WHERE  
customerID = 1;
```

Query OK, 1 row affected (0.0054 sec)

Rows matched: 1 Changed: 1 Warnings: 0

Delete Customer Info:

Assumption: There is a club member with ID 1 in the database.

```
SQL> DELETE FROM ClubMembers WHERE customerID=1;
```

Query OK, 1 row affected (0.0047 sec)

Enter Staff Info:

Assumption: There is a staff member hired with the staffID of 1, name of Jim Staff, birth date of 01-01-2000, address of 111 Jim Rd, Raleigh NC, the hire date of 01-01-2024, the role of cashier, with a phone number 311-111-1111, the email JimStaff@gmail.com at store with the ID 1.

```
SQL > INSERT INTO StaffMembers VALUES (1, 'Jim Staff',  
'2000-01-01', '111 Jim Rd, Raleigh NC', '2024-01-01',  
'Cashier', '311-111-1111', 'JimStaff@gmail.com', 1);
```

Query OK, 1 row affected (0.0072 sec)

Update Staff Info:

Assumption: The staff member changes their email address to JimStaff2@gmail.com and needs it updated.

```
SQL > UPDATE StaffMembers SET  
staffEmail='JimStaff2@gmail.com' WHERE StaffID=1;
```

Query OK, 1 row affected (0.0059 sec)

Rows matched: 1 Changed: 1 Warnings: 0

Delete Staff Info:

Assumption: The staff member with ID 1 quits and their data is removed from the database.

SQL > DELETE FROM StaffMembers WHERE StaffID=1;

Query OK, 1 row affected (0.0071 sec)

Enter Supplier Info:

Assumption: There is a new supplier with ID 1, name Chili Supplier Co., phone number 411-111-1111, email ChiliCo@gmail.com, and address of 121 Chili Rd, Raleigh NC.

SQL > INSERT INTO Suppliers VALUES (1, 'Chili Supplier Co.', '411-111-1111', 'ChiliCo@gmail.com', '121 Chili Rd, Raleigh NC');

Query OK, 1 row affected (0.0064 sec)

Update Supplier Info:

Assumption: The supplier with ID 1 wants to change their phone number to 411-111-1112.

SQL > UPDATE Suppliers SET supplierNum='411-111-1112' WHERE supplierID=1;

Query OK, 1 row affected (0.0051 sec)

Rows matched: 1 Changed: 1 Warnings: 0

Delete Supplier Info:

Assumption: Suppliers with ID 1 went out of business and are no longer relevant so they need to be removed from the database.

SQL > DELETE FROM Suppliers WHERE supplierID=1;

Query OK, 1 row affected (0.0047 sec)

Create Promotion:

Assumption: There is a new discount for the product with storeID 1 and product ID 1.

The new discount has an ID of 1, starts 01-01-2024, ends 01-01-2025 and is for 20% off.


```
SQL > INSERT INTO Discounts VALUES (1, 1, 1, '2024-01-01',  
'2025-01-01', 0.80);
```

Query OK, 1 row affected (0.0056 sec)

Update Promotion:

Assumption: The discount with discount ID 1 has been changed to 40% off.

```
SQL > UPDATE Discounts SET promotion=0.60 WHERE  
discountID=1;
```

Query OK, 1 row affected (0.0048 sec)

Rows matched: 1 Changed: 1 Warnings: 0

Delete Promotion:

Assumption: The discount with discount ID 1 has been canceled.

```
SQL > DELETE FROM Discounts WHERE discountID=1;
```

Query OK, 1 row affected (0.0044 sec)

Manage Inventory Records:

Create Inventory Info:

Assumption: A new merchandise has been delivered with product ID 9 at store ID 1 named Chili. There are 100 cans bought at \$5 a can and with a market price of \$6 a can. They were produced 2025-03-01 and expire 2025-12-01.

```
SQL > INSERT INTO Merchandise VALUES(1, 9, 'Chili', 100,  
5.00, 6.00, '2025-03-01', '2025-12-01', 1);
```

Query OK, 1 row affected (0.0049 sec)

Update Inventory Info:

Assumption: Handle a return for merchandise with productID 9 at store ID 1.

```
SQL > UPDATE Merchandise SET stockQuantity = stockQuantity  
+ 1 WHERE storeID = 1 AND productID = 9;
```

Query OK, 1 row affected (0.0053 sec)
Rows matched: 1 Changed: 1 Warnings: 0

Delete Inventory Info:

Assumption: Delete merchandise from store with ID 1 with product ID 9 as it is all sold out.

**SQL > DELETE FROM Merchandise WHERE storeID = 1 AND
productID = 9;**

Query OK, 1 row affected (0.0051 sec)

Perform Transfers:

Assumption: Merchandise with product ID 1 exists in store with ID 1 is transferred to be merchandise with product ID 100 in store with ID 2. The transfer of this merchandise to store 2 is performed by staff ID 1 on March 5th, 2025.

**SQL> UPDATE Merchandise SET storeID=2, productID=100
WHERE storeID=1 AND productID=1;**

Query OK, 1 row affected (0.0048 sec)
Rows matched: 1 Changed: 1 Warnings: 0

**SQL> INSERT INTO Transfers VALUES(1, 2, 1, 100,
'2025-03-05', 1);**

Query OK, 1 row affected (0.0041 sec)

Maintain Billing and Transaction Records:

Generate Bill:

Assumption: A bill created by billing staff with ID 1 needs to be paid to the supplier with ID 1. It has not been paid yet, and is for the amount of \$1000.

SQL> INSERT INTO Bills VALUES(1, 1000.00, 'unpaid', 1, 1);
Query OK, 1 row affected (0.0046 sec)

Update Bill:

Assumption: The bill with ID 1 needs to be updated to reflect new purchased supplies.
Change the value to \$2000.

```
SQL> UPDATE Bills SET amountOwed=2000.00 WHERE billID=1;
```

Query OK, 1 row affected (0.0049 sec)
Rows matched: 1 Changed: 1 Warnings: 0

Create Reward Check:

Assumption: Create a reward check made by billing staff with ID 1 being paid to customer with ID 2 for the amount of \$30.

```
SQL> INSERT INTO Rewards VALUES(1, 30.00, 1, 1);
```

Query OK, 1 row affected (0.0058 sec)

Update Reward Check:

Assumption: Update a reward check to reflect new purchases where the new value of the check should be \$50 and the rewardID is 1.

```
SQL> UPDATE Rewards SET checkAmountOwed=50.00 WHERE checkID=1;
```

Query OK, 1 row affected (0.0052 sec)
Rows matched: 1 Changed: 1 Warnings: 0

Add Transaction Info:

Assumption: A transaction occurs at a store with ID 1 with staff with ID 1 and a customer with ID 1. The date is 01-01-2025, the customer purchases one can of chili with product ID 1 that costs 10 dollars. The transaction has an ID of 1.

```
SQL>INSERT INTO Transactions VALUES(1, '2025-01-01', NULL, 1, 1);
```

Query OK, 1 row affected (0.0046 sec)

```
SQL>INSERT INTO TransactionDetails VALUES(1, 1, 1, 10.00, 1);
```

Query OK, 1 row affected (0.0043 sec)

Update Transaction Info (Calculate total transaction cost with discounts):

Assumption: There is a transaction with ID 1 involving two distinct items. The first item has a 20% discount and the second item has a 15% discount.

Apply discounts to products for a given transaction:

```
SQL> UPDATE TransactionDetails t SET  
finalSum=(COALESCE((SELECT promotion FROM Discounts d WHERE  
d.storeID=t.storeID AND d.productID=t.productID), 1) *  
t.finalSum) WHERE t.transactionID=1;
```

Query OK, 2 rows affected (0.0053 sec)

Rows matched: 2 Changed: 2 Warnings: 0

Calculate the total transaction amount:

```
SQL> UPDATE Transactions SET totalPrice=(SELECT  
SUM(finalSum)  
FROM TransactionDetails WHERE TransactionID=1) WHERE  
TransactionID=1;
```

Query OK, 1 row affected (0.0046 sec)

Rows matched: 1 Changed: 1 Warnings: 0

Reports:

Calculate Total Sales Report by Day:

Assumption: Find the total sales for the chain on January 5th 2024. \$5 worth of goods was sold on this date.

```
SQL> SELECT SUM(totalPrice) FROM Transactions  
WHERE purchaseDate='2025-01-01';
```

+-----+

```
| SUM(totalPrice) |
+-----+
|          5.00 |
+-----+
1 row in set (0.0032 sec)
```

Calculate Total Sales Report by Month:

Assumption: Find the total sales for the chain in January 2024. There were \$400 of total sales.

```
SQL> SELECT SUM(totalPrice) FROM Transactions
WHERE purchaseDate>='2024-01-01' AND
purchaseDate<'2024-02-01';
```

```
+-----+
| SUM(totalPrice) |
+-----+
|        400.00 |
+-----+
1 row in set (0.0034 sec)
```

Calculate Total Sale Report by Year:

Assumption: Find the total sales for the chain in 2024. There were \$496 of total sales.

```
SQL> SELECT SUM(totalPrice) FROM Transactions
WHERE purchaseDate>='2024-01-01' AND
purchaseDate<'2025-01-01';
```

```
+-----+
| SUM(totalPrice) |
+-----+
|        496.00 |
+-----+
1 row in set (0.0034 sec)
```

Calculate Sales Growth by Store and Time Period:

Assumption: Find how the total purchases change over time at the store with ID 1 in the month of January 2024. The store just opened and has a total purchase of \$1 for day one. It grows by \$1 every day for January.

```
SQL> SELECT t.purchaseDate, SUM(t.totalPrice)
FROM Transactions t, StaffMembers s
WHERE s.storeID=1 AND s.staffID=t.staffID AND
t.purchaseDate>='2024-01-01' AND
t.purchaseDate<'2024-02-01'
GROUP BY t.purchaseDate
ORDER BY t.purchaseDate;
```

purchaseDate	SUM(t.totalPrice)
2024-01-01	1.00
2024-01-02	2.00
2024-01-03	3.00
2024-01-04	4.00
2024-01-05	5.00
2024-01-06	6.00
2024-01-07	7.00
2024-01-08	8.00
2024-01-09	9.00
2024-01-10	10.00
2024-01-11	11.00
2024-01-12	12.00
2024-01-13	13.00
2024-01-14	14.00
2024-01-15	15.00
2024-01-16	16.00
2024-01-17	17.00
2024-01-18	18.00
2024-01-19	19.00
2024-01-20	20.00
2024-01-21	21.00

2024-01-22		22.00	
2024-01-23		23.00	
2024-01-24		24.00	
2024-01-25		25.00	
2024-01-26		26.00	
2024-01-27		27.00	
2024-01-28		28.00	
2024-01-29		29.00	
2024-01-30		30.00	
2024-01-31		31.00	

+-----+-----+

31 rows in set (0.0049 sec)

Merchandise Stock Report by Store:

Assumption: Find the stock in store with ID 1. There are 10 counts of chili and 20 counts of beans.

```
SQL> SELECT productName, Sum(stockQuantity) FROM
Merchandise
WHERE storeID=1 GROUP BY productName;
```

productName		Sum(stockQuantity)
Beans		20
Chili		10

+-----+-----+

2 rows in set (0.0039 sec)

Merchandise Stock Report for Certain Product:

Assumption: Find the stock of chili across all stores. The stock is 101 items.

```
SQL> SELECT Sum(stockQuantity) FROM Merchandise
WHERE productName='Chili';
```

Sum(stockQuantity)	
--------------------	--

```

+-----+
|                  101 |
+-----+
1 row in set (0.0041 sec)

```

Customer Growth Report by Month:

Assumption: Report the customer growth in January 2024. There were two new customers in January 2024.

```

SQL> SELECT Count(*) FROM ClubMembers
WHERE signUpDate>='2024-01-01' AND signUpDate<'2024-02-01';

```

```

+-----+
| Count(*) |
+-----+
|          2 |
+-----+
1 row in set (0.0042 sec)

```

Customer Growth Report by Year:

Assumption: Report the customer growth in 2024. There were two new customers in 2024.

```

SQL> SELECT Count(*) AS CustomerGrowth2024 FROM ClubMembers
WHERE signUpDate>='2024-01-01' AND signUpDate<'2025-01-01';

```

```

+-----+
| CustomerGrowth2024 |
+-----+
|                      2 |
+-----+
1 row in set (0.0046 sec)

```

Customer Activity Report (Total purchase amount in given time period):

Assumption: Report for total purchased amount for customer with ID of 1 in 2024. The customer with ID 1 has purchased \$100 of goods.

```

SQL> SELECT customerID, Sum(totalPrice) AS TotalPurchases
FROM Transactions
WHERE purchaseDate>='2024-01-01' AND
purchaseDate<'2025-01-01'
GROUP BY customerID

```


ORDER BY customerID;

```
+-----+-----+
| customerID | TotalPurchases |
+-----+-----+
|          1 |          100.00 |
+-----+-----+
1 row in set (0.0046 sec)
```

13.2

Return Transactions from a certain day

```
1. EXPLAIN SELECT *
   FROM Transactions
   WHERE purchaseDate = '2024-03-06';
```

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	Transactions	ALL	NULL	NULL	NULL	NULL	6	Using where

```
2. CREATE INDEX pDateIndex ON Transactions(purchaseDate);
```

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	Transactions	ref	pDateIndex	pDateIndex	3	const	1	

Select all Merchandise Names from a certain supplier

```
1. EXPLAIN SELECT m.productName
   FROM Merchandise m
   JOIN Suppliers s ON m.supplierID = s.supplierID
   WHERE s.supplierName = 'Fresh Foods Co.';
```

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	s	ALL	PRIMARY	NULL	NULL	NULL	5	Using where
1	SIMPLE	m	ALL	Merchandise_ibfk_2	NULL	NULL	NULL	10	Using where; Using join buffer (flat, BNL join)

2. CREATE INDEX sNameIndex ON Suppliers(supplierName);

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	s	ref	PRIMARY,sNameIndex	sNameIndex	130	const	1	Using where; Using index
1	SIMPLE	m	ref	Merchandise_ibfk_2	Merchandise_ibfk_2	4	mhpate14.s.supplierID	1	

13.3

QUERY 1: Get the amount owed and the status of the bills owed to each supplier

```
SELECT s.supplierName, b.amountOwed, b.status
FROM Bills b
JOIN Suppliers s ON b.supplierID = s.supplierID;
```

Specification: Return a list of the suppliers' names, the amount owed to them, and whether they've been paid.

$$\Pi_{Suppliers.supplierName, Bills.amountOwed, Bills.status} (Bills \bowtie_{Bills.supplierID=Suppliers.supplierID} Suppliers)$$

Suppose *b* is any tuple in the Bills relation and *s* is any tuple in the Suppliers relation, such that the value *s.supplierID* is equal to the value of *b.supplierID*. This combination of the tuples *b* and *s* will yield all details about one bill and the supplier it's meant to be paid to. For such a join (*s*, *b*) the query returns the values of *supplierName*, *amountOwed*, and *status*. These values are the name of the supplier, the amount owed to them from Bills, and whether or not that bill has been paid. According to the specification, this is exactly what the query was meant to return.

QUERY 2 :

Return the checkAmountOwed of all the rewards scheduled to be paid to customers with suspended accounts

```
1. SELECT Rewards.checkAmountOwed
   FROM Rewards
  JOIN ClubMembers ON Rewards.customerID =
    ClubMembers.customerID
 WHERE ClubMembers.custStatus = 'Suspended';
```

Specification: Return the amount owed of all the rewards scheduled to be paid to customers with suspended accounts

$$\Pi_{\text{Rewards.checkAmountOwed}} (\sigma_{\text{ClubMembers.custStatus} = \text{'suspended'}} (\text{Rewards} \bowtie_{\text{Rewards.customerID} = \text{ClubMembers.customerID}} \text{ClubMembers}))$$

Suppose r is any tuple in the Rewards relation and c is any tuple in the ClubMembers relation where $c.\text{customerID} = r.\text{customerID}$ and $c.\text{custStatus} = \text{'suspended'}$. This join of tuples provides all the information on the club members who have a suspended account and the rewards they are potentially owed. From such a join (c, r) the query returns the values of checkAmountOwed for each club member in the union. This corresponds to the amount of rewards money the customers with a suspended account can expect, which is equivalent to what is being asked in the above specification.

Transaction 1:

```
//Create a reward object if the input customer is a Platinum Member with their membership active
private static String createReward(Integer rewardID, Double checkAmountOwed, Integer staffID, Integer
customerID, String startDate, String endDate) throws SQLException {
    String insertSQL = "INSERT INTO Rewards (rewardID, checkAmountOwed, staffID, customerID)
VALUES (?, ?, ?, ?)";
    String selectSQL = "SELECT membershipLevel, custStatus FROM ClubMembers WHERE customerID
= ?";

    checkAmountOwed = 0.0;

    try {
        //Start the transaction.
        connection.setAutoCommit(false);

        int changes = 0;
```

```

try (PreparedStatement ps = connection.prepareStatement(insertSQL)) {
    ps.setInt(1, rewardID);
    ps.setDouble(2, checkAmountOwed);
    ps.setInt(3, staffID);
    ps.setInt(4, customerID);
    changes = ps.executeUpdate();
}
//If there are no changes from the insert statement, rollback
if (changes == 0) {
    connection.rollback();
    return "Reward creation failed.";
}

try (PreparedStatement ps2 = connection.prepareStatement(selectSQL)) {
    ps2.setInt(1, customerID);
    try (ResultSet rs = ps2.executeQuery()) {
        if (rs.next()) {
            String level = rs.getString("membershipLevel");
            String status = rs.getString("custStatus");

            if ("Platinum".equals(level) && "Active".equals(status)) {
                String message = calculateReward(customerID, startDate, endDate);
                //If the reward calculation and update fails, rollback
                if (!"Reward calculated successfully.".equals(message)) {
                    connection.rollback();
                    return "Reward calculation failed.";
                }
                //If the reward is calculated and updated successfully, commit.
                connection.commit();
                return "Reward successfully created.";
            }
            //If the customer is not an active platinum customer, rollback
        } else {
            connection.rollback();
            return "Not an active platinum customer.";
        }
    }
    //If there is no customer with the given ID in the database, rollback
} else {
    connection.rollback();
    return "Invalid club member ID.";
}
}
}

//Handle an exception
} catch (Exception error) {
    //Rollback if the connection is valid.
    if (connection != null) {
        connection.rollback();
        return "Reward calculation failed.";
    }
    // Return an error if the connection is null.
}

```

```

    } else {
        return "Connection null.";
    }
    // If the connection is not null, reset autocommit to true before returning.
} finally {
    if (connection != null) {
        connection.setAutoCommit(true);
    }
}
}
}

```

Transaction 2:

//Transfer operations (for moving stock between stores)

private static String processTransfer(Integer store1ID, Integer store2ID, Integer product1ID, Integer product2ID, String transferDate, Integer staffID) throws SQLException {

```

    try {
        //Start the transaction.
        connection.setAutoCommit(false);

        int changes = 0;
        String sql = "UPDATE Merchandise SET storeID = ?, productID = ? WHERE storeID = ? AND
productID = ?";
        PreparedStatement ps = connection.prepareStatement(sql);
        ps.setInt(1, store2ID);
        ps.setInt(2, product2ID);
        ps.setInt(3, store1ID);
        ps.setInt(4, product1ID);
        changes = ps.executeUpdate();

        //If there were no changes after executing the update, rollback
        if (changes == 0) {
            connection.rollback();
            connection.setAutoCommit(true);
            return "Transfer failed.";
        }

        String sql2 = "INSERT INTO Transfers (store1ID, store2ID, product1ID, product2ID, transferDate,
staffID) " +
            "VALUES (?, ?, ?, ?, ?, ?)";
        ps = connection.prepareStatement(sql2);
        ps.setInt(1, store1ID);
        ps.setInt(2, store2ID);
        ps.setInt(3, product1ID);

```

```

ps.setInt(4, product2ID);
ps.setDate(5, java.sql.Date.valueOf(transferDate));
ps.setInt(6, staffID);
changes = ps.executeUpdate();

//If there were no changes after executing the insert, rollback
if (changes == 0) {
    connection.rollback();
    connection.setAutoCommit(true);
    return "Transfer failed.";
//If there were changes, commit
} else {
    connection.commit();
    connection.setAutoCommit(true);
    return "Transfer processed successfully.";
}

} catch (Exception error) {
    //If an error is thrown, rollback
    if (connection != null) {
        connection.rollback();
        connection.setAutoCommit(true);
        return "Transfer failed.";
    //If there is no connection, output an error message
    } else {
        return "Connection null.";
    }
}
}

```

Design Decisions:

The system is a command-line interface that allows for different views with different allowed commands for each. The user first selects a view by inputting a number associated with each view “Registration Office”, “Admin”, “Warehouse Operator”, “Billing Staff”, and “Cashier”. When prompted, the user enters the name of the command corresponding with what action they would like to take. For each menu option it first lists the allowed commands for that view. Our system is all in 1 main class that facilitates switching between views on the command-line. This was done because many of the need to access similar db actions and to not have too many duplicates we put it all in 1 class. However, adding the code to this class is easy because much of the code is modular and only needs to add associated db alteration function and into the command-line interface list. Our program also contains helper functions to connect and disconnect from the database.