3. Data Management **(Revised by Triet)**

To devise the design and organization of our database systems, we follow these steps:

1. Recall the tasks we want the point-of-sale system to manage and execute.
2. Recall the tools and procedures we implemented and drew to solve such tasks.
3. Brainstorm the types of data that need to be feed into such tools and procedures

for them to produce the results that qualify as solutions to the tasks we intend our point-of-sale system to solve. Such data, their content and their type, will be the data that our database systems must record.

Examples of Formulation

|  |  |  |  |
| --- | --- | --- | --- |
| **Task** | **Data** | **Format** | **Source** |
| Allow employees and admins to login into the system | Login permissions (identity, username, password, storeID) | Password is encrypted. The rest will be string. | Manually chosen by user |
| Track which employee works at which store and their job title (trainee, regular, experienced, manager) | Store location, store’s manager. Employee’s store location, work hours, current length of service | Store location: string  Store’s manager: ID  Employee’s store location: ID  Employee’s work hours: string  Current length of service: int | Most string types will be manually recorded. Current length of service is computed by the system |
| Implement a “loyal customer” program that grants and alerts such customers to chosen deals on items | Customer’s name, contact info, address, membership tier, TransactionID to link to the transaction history | Name: String  Phone no: string  Membership tier: int  TransactionID: int | Data relating to customer’s identity will be manually recorded.  TransactionID will be given by the database |

For the tasks we anticipate, MySQL will be a reliable starting system. Being a relational database, SQL emphasizes structure to data and offers a simple, but effective language in structured query language for retrieving data based on their relations. Items, transactions, and inventory represent entities that are structures and so a point-of-sale system that works with such data will be well-managed by a relational database like SQL.

Some examples of the tasks that our point-of-sale system is intended to manage were described above, such as managing employee logins, employee’s store work location, and a customer list for a “loyal customer” program.

For each entity that we must manage, we will create a table for it in MySQL. For example, we will create a table for all these pieces of data: product, inventory, store, sale transaction, item return, refund. We are using one database with tables for all such entities so to maintain data integrity and help with query performances as multiple tables can be joined together much simpler in one database.

Employee Table has columns: ID (as primary key), name, login credentials, position, and contact information, and transaction id (for transactions done).

Customer Table has columns: ID, name, address, contact information, rewards status, transactionID (referencing foreign key in Transaction Table).

Item Table has columns: ID, name, description, price, quantity (inventory count), and category (referencing foreign key in Category Table).

Transaction Table has columns: ID, transaction date, employee id, customer id, amount paid, items, payment type, and transaction type (any of refund, sale, exchange).

To maintain the integrity and quality of our data, we use multiple tables with the aim to normalize our data and eliminate any redundancies in stored data. With more tables comes a more complex system to design, however, it allows us to make a more efficient system in terms of performance and, perhaps, data integrity.

A second consideration to our data management strategy is to store a hot copy of all the data in a separate cloud server backup. It is important to have multiple copies of data among different storage levels as a preventative measure in case something were to go wrong. We will also make sure to monitor database activities often.

**(Draft by Konrad)**

Our data management strategy revolves around SQL because it is a relational database management system, which should allow us to handle transactions and item management well through its structured form. Specifically, MySQL will be a fine choice for our system. First, we will need our database to manage employees, their logins, transactions, etc., so we will create a table for these. Second, we will have another table to keep a record of customer data. Third, we will have a table for products and inventory. Lastly, we will need a table for sales transactions and refunds. We are using one database with these four tables in order to maintain data integrity and help with query performances as multiple tables can be joined together much simpler in one database. We have designed the employee table to feature employee id (as primary key), name, login credentials, position, and contact information, and transaction id (for transactions done). The customers table will have an id column as well (for easy identification), name, address, contact information, rewards status, as well as transaction id (as transaction history). The products table will have a product id column, name, description perhaps, price, quantity (inventory count), and category. The sales/transaction table will have the columns: transaction id, transaction date, employee id, customer id, amount paid, items, payment type, and finally transaction type (refund, sale, exchange). We have used multiple tables in order to try to normalize our data and eliminate any redundancies in stored data. With more tables comes a more complex system to design, however, it allows us to make a more efficient system in terms of performance and, perhaps, data integrity. The second part to our data management strategy is storing a hot copy of all the data in a separate cloud server backup. It is important to have multiple copies of data among different storage levels as a preventative measure in case something were to go wrong. We will also make sure to monitor database activities often.