CS338 Course Project

Milestone 1: Project Proposal

High-Level Description:

Our application is designed to create a management system for HR headquarters to manage tasks. We assume the existence of multiple branches. Each branch has different brands that provide various items to our branches. Since each brand does not have its own delivery system, they choose different third-party transport services. Additionally, our system will record each customer's personal information and their order details. We also plan to allow users to see information about orders made by each branch, such as the names of sold items and the customers who purchased them. We will have multiple input CSV files for the information of branches, brands, customers, employees, orders, products, and shipping. The user will be the manager of the company, who will also be regarded as the administrator. Once the information is imported, the user can check any information by accessing the system. The goal is to let the user easily import their data first. Additionally, we want the user to add, delete, edit any information in a visualized way to highly increase convenience and efficiency.

System Support:

We plan to choose MySQL for backend support, and we use HTML, CSS, and JavaScript for the frontend, with the Flask framework in Python for the backend. To make connection between the frontend and the backend, we define the HTTP methods (GET, POST, DELETE) that can be accessed by making fetch call to those APIs in JavaScript. By this way, we implement a way of seamless data transportation from database to frontend. We are also planning to use a bash script to handle all preparation steps and data importing steps, which should be based on Linux/MacOS.

Plan for getting sample data:

The first step is to identify the data we need. As the description for this project, we need the information of branches, brands, customers, employees, orders, products, and shipping. Each one will be regarded as a table, and we will design the schema for each table. We will prepare the toy data manually. We stored our sample data into folder `sample_data`.

The second step is importing the data, we need a `set_up.sql` file to set up all the schema based on our design. After that, we will use `load_sample.sql` to import data from the CSV files into their corresponding tables. For example, we will have a CSV file called `brand.csv`, then the data in this file will be imported into the `brand` table. We expect user to run `mysql -u root -p < sql/set_up.sql` and then run `mysql -u root -p –local-infile < sql/load_sample.sql` which should guarantee the database is correctly established. (This should change when MySQL has password. For example, if the password is 12345678, the command should be updated to `mysql -u root -p”12345678” < sql/set_up.sql`) For detailed instruction, please see `README.md`.

Schema Design:

R5a.Assumptions:

1.Unique identifiers:

- Each Brand has a unique BrandID.
- Each Product has a unique ProductID.
- Each Customer has a unique CustomerID.
- Each Employee has a unique EmployeeID.
- Each Branch has a unique BranchID.
- Each Order has a unique OrderNum.
- Each Shipping entry has a unique ShippingID.

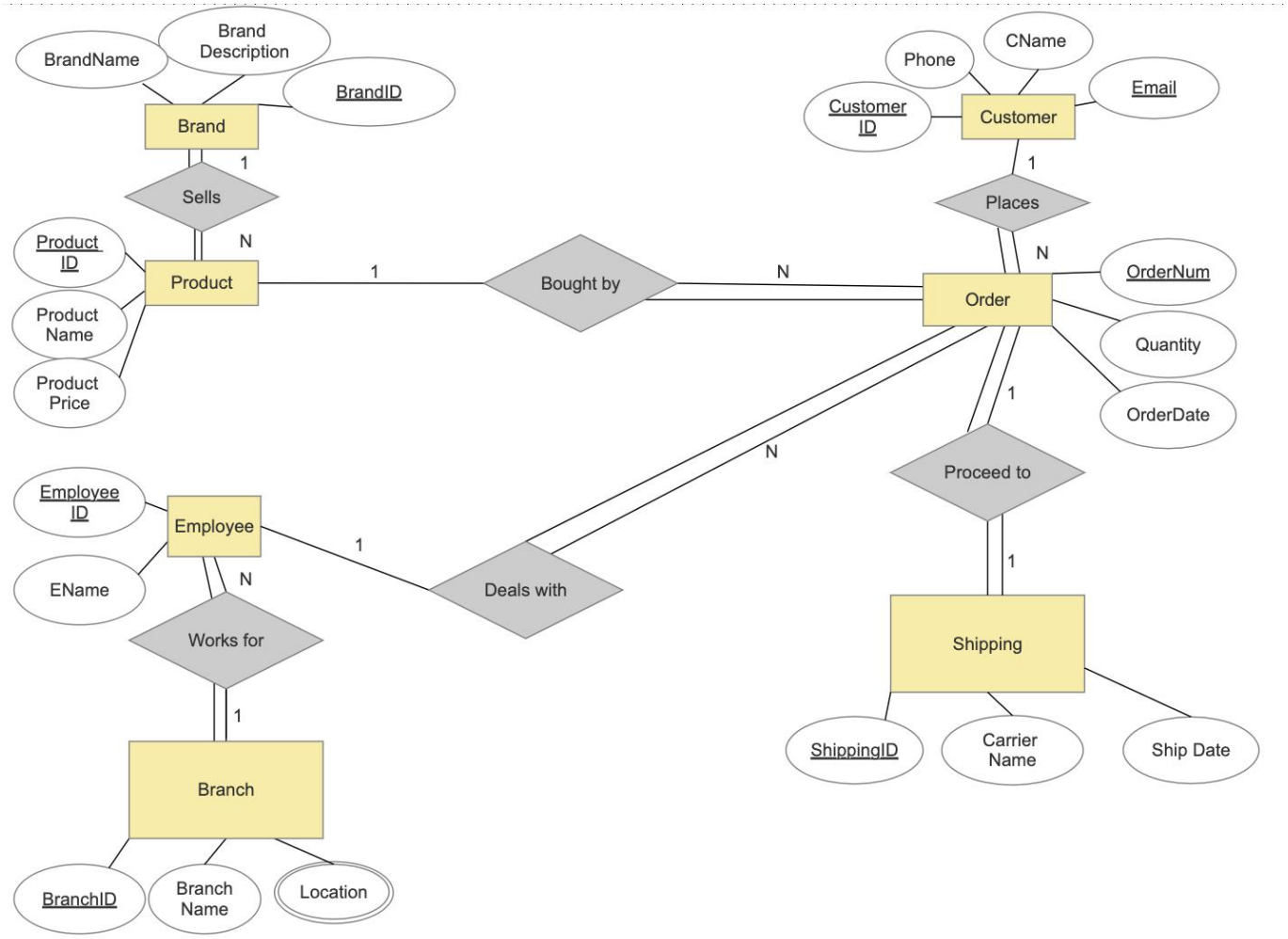
2.Relationship:

- Each Brand can sell multiple Products, but each Product is sold by only one Brand.
- Each Product can be bought in multiple Orders, but each Order contains at least one Product.
- Each Customer can place multiple Orders, but each Order is placed by only one Customer.
- Each Order proceeds to only one Shipping entry, but each Shipping entry must be associated with one Order.
- Each Employee works for only one Branch, but each Branch can have multiple Employees.
- Each Branch deals with multiple Orders, but each Order is dealt with by only one Branch

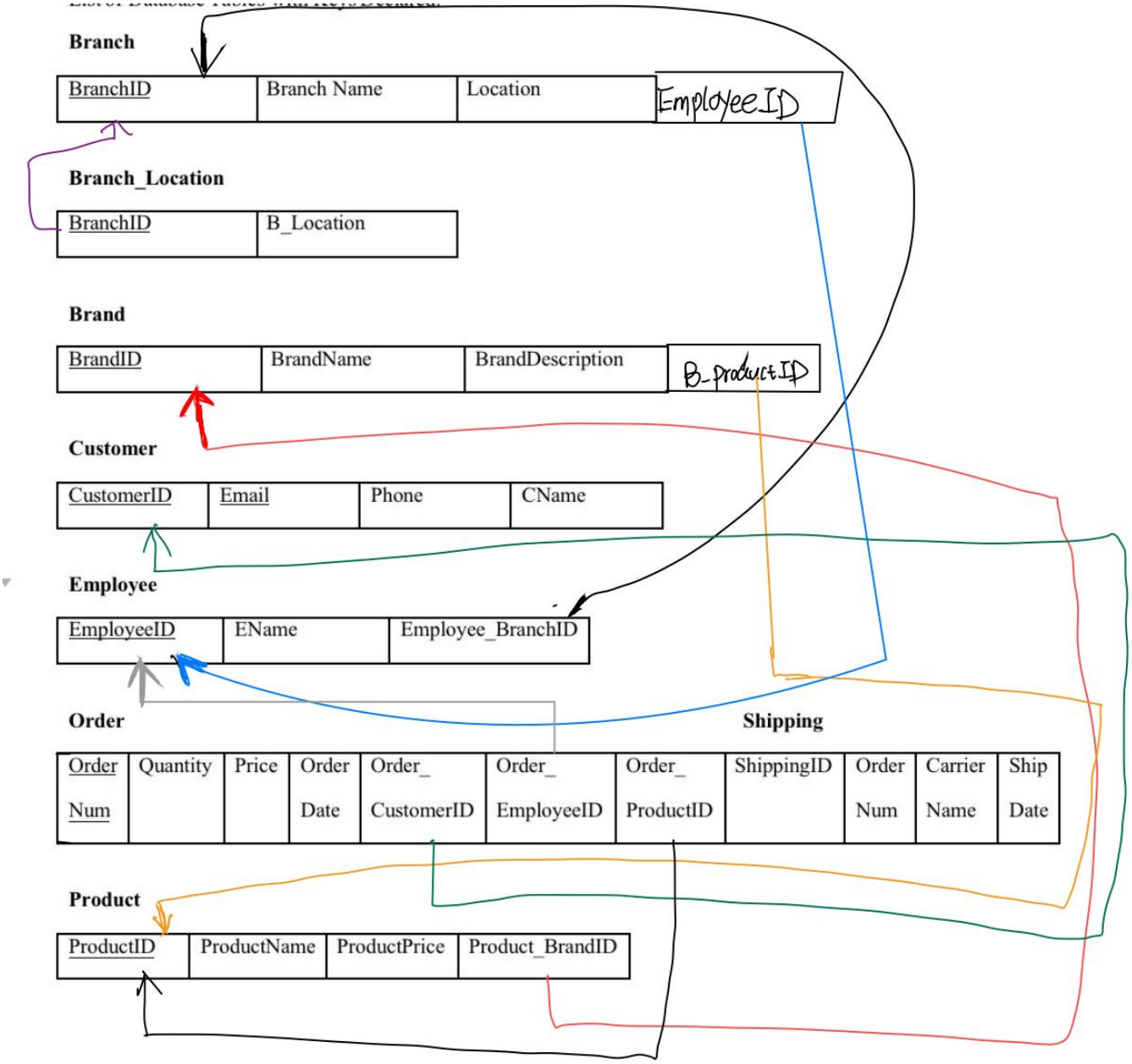
3. Integrity Constraints:

- Order_CustomerID must refer to an existing CustomerID in Customer entity.
- Order_EmployeeID must refer toan existing EmployeeID in Employee entity
- Order_ProductID must refer to an existing ProductID in Product entity
- Employee_BranchID must refer to an existing BranchID in Branch entity.
- Product_BrandID must refer to an existing BrandID in Brand entity.

E/R Diagram:


Brand Description
BrandName
BrandID
Brand
1
Sells
N
Product ID
Product
Product Name
Product Price
1
Bought by
N
Phone
CName
Email
Customer ID
Customer
1
Places
N
OrderNum
Order
Quantity
OrderDate
1
Proceed to
1
Shipping
ShippingID
Carrier Name
Ship Date
N
Employee ID
Employee
EName
1
N
Works for
1
Deals with
Branch
BranchID
Branch Name
Location

Relational Data Model:



Feature

Currently, the four features are the same for all different tables. Thus, we will use the Brand table as an example to show our feature.

1. The first feature is to retrieve the Brand table entries for users to see available Brands. Our default home page is `/frontend/index.html`, and users will access this feature by navigating to `/frontend/Web/Brand.html`. Specifically, users will click the `brand` button on the homepage to enter.

From the frontend, a query will be made to the backend to retrieve all brands' information to be displayed on the user's screen. The core SQL query we use for this feature is: `SELECT * FROM Brand`. That is, when users entering `/frontend/Web/Brand.html`, all the information of brands should be displayed.

Sample Output:

```
+---------+-----------+
| BrandID | BrandName |
+---------+-----------+
|       1 | Brand A   |
|       2 | Brand B   |
|       3 | Brand C   |
|       4 | Brand D   |
|       5 | Brand E   |
|       6 | Brand F   |
|       7 | Brand G   |
|       8 | Brand H   |
|       9 | Brand I   |
|      10 | Brand J   |
+---------+-----------+
```

2.The second feature allows users to update an existing brand name. In case a brand decides to rename, we provide an option for users to edit the existing names. This can be done by pressing the "edit" button corresponding to the brand they want to edit, then changing the fields they want to update. To be detailed, users can edit any column we allow, such as `BrandID` or `BrandName`. The core SQL query we use for this feature is: `UPDATE Brand SET BrandName = '{brandName}' WHERE BrandID = '{brandID}'`. There is no output for this feature. After users click update, they should see the data change.

Sample:

Previous:

```
+---------+-----------+
| BrandID | BrandName |
+---------+-----------+
|       1 | Brand A   |
|       2 | Brand B   |
+---------+-----------+
```

Once the edition succeeds:

```
+---------+-----------+
| BrandID | BrandName |
+---------+-----------+
|       1 | Brand C   |
|       2 | Brand B   |
+---------+-----------+
```

3.The third feature allows users to delete any existing brand. This can be done by directly pressing the `delete` button corresponding to the brand they want to delete. After clicking `delete`, such row should disappear. The core SQL query we use for this feature is: `DELETE FROM Brand WHERE BrandID = '{brandID}'`. There is no output for this feature.

Sample:

Previous:

```
+---------+-----------+
| BrandID | BrandName |
+---------+-----------+
|       1 | Brand A   |
|       2 | Brand B   |
+---------+-----------+
```

Once deletion succeeds:

```
+---------+-----------+
| BrandID | BrandName |
+---------+-----------+
|       2 | Brand B   |
+---------+-----------+
```

4.The fourth feature allows users to insert a new brand. This can be done by pressing the `add` button (red "+" button), entering the relevant information inside the form, and pressing submit. In this case, since we assume each brand has a unique BrandID, if the new insertion has the same BrandID as an existing row, it will perform like an edit. This only works for new BrandIDs. Once it succeeds, users should see an extra row appear. The core SQL query is: `INSERT INTO Brand (BrandID, BrandName) VALUES ('{brandID}', '{brandName}')`. There is no output for this feature.

Sample:

Previous:

```
+---------+-----------+
| BrandID | BrandName |
+---------+-----------+
|       2 | Brand B   |
+---------+-----------+
```

Once the insertion succeeds:

```
+---------+-----------+
| BrandID | BrandName |
+---------+-----------+
|       1 | Brand A   |
|       2 | Brand B   |
+---------+-----------+
```

Members' Contribution:

Mara Ma's main duty was to determine the theme of our project. After deciding on the theme, we focused on designing the database schema and creating the E/R diagram. Once these tasks were completed, Mara finished the related part of the report.

Franda Zhu's main duty was the frontend, including UI design and making fetch calls to the backend. Additionally, Franda completed most of the report's content. Currently, the basic UI design is done.

Rebecca Zhao focused on the backend, including the implementation of the Flask framework and SQL queries. Rebecca Zhao and Mara Ma also provided a README file explaining how to run the backend service. Currently, we has finished four features for the brands section.

GitHub Repo link:

https://github.com/marama01/CS338-project.git