The State University of New York, Korea

Starbucks Chain Management System

Roshan Poudel, Nilson Chapagain, Crispen Chisina

Bachelors of Science in Computer Science

June 10, 2021

Professor Arthur Lee

Department of Computer Science

Abstract

The goal of this project is to use our fundamental knowledge of Database Systems learned throughout the semester to design and implement a web based full stack application software with a focus on database systems. Starbucks Chain Management System is a DBMS targeted to Starbucks managers and owners, which can be used to store information and to perform administrative tasks like adding or removing employees, updating menu, viewing and storing sales information, checking and updating their supply stock, etc. There is a central admin dashboard that can be used by the Starbucks CEO to get information about all the Starbucks chains, and to make any updates if necessary.

Contents

Chapter 1

Project Overview

1.1 Introduction

Starbucks is an American multinational company which has multiple coffee chains across the globe. Each chain sells a subset of coffee and beverages selected from a menu of items set by the Starbucks corporation. While the menu selection and the facilities differ across different Starbucks chains, to provide a similar quality of service, each Starbucks chain uses the same coffee beans from the same suppliers. Each chain has multiple departments and employees, and each department is managed by an employee. When an employee is hired, information about that employee should be recorded and be available to the corporation. After a customer makes an order, the sales information is stored and accessible to the chain manager and the Starbucks corporation. Furthermore, the customers can leave reviews for the chains and the reviews can be used by the Starbucks corporation to gain insights about customer satisfaction. Our project is a centralized system designed to help the executives to manage the tasks in their respective chains smoothly, and for the Starbucks corporation to oversee the chains across the globe.

1.2 Requirements

- Sign up and login for branch managers, head of Starbucks corporation, and customers.
- Manager dashboard to access and update order history, branch menu, employee records, supply inventory, and customer reviews.
- Admin dashboard to access and update information for all Starbucks chains.

- Customer portal to read and leave reviews for different chains, join loyalty program, and check main menu.

1.3 Tech Stack

- HTML/CSS
- BootStrap
- Javascript
- Node.js, Express
- MySQL 8.1

1.4 User Perspectives

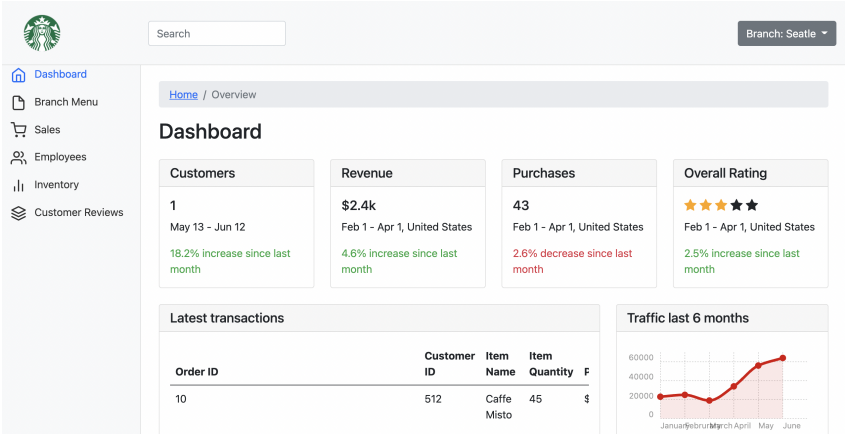
The project maintains three levels of users:

- Chain Manager
- Starbucks Corporation head (Admin)
- Customers

1.4.1 Chain Manager Perspective / Guide

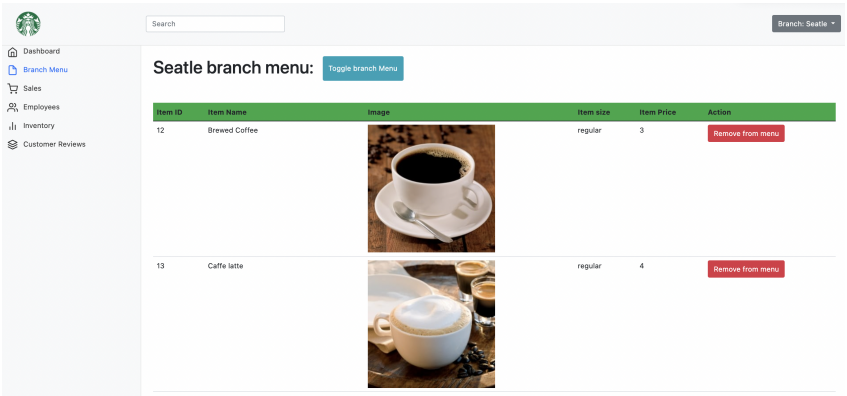
Dashboard

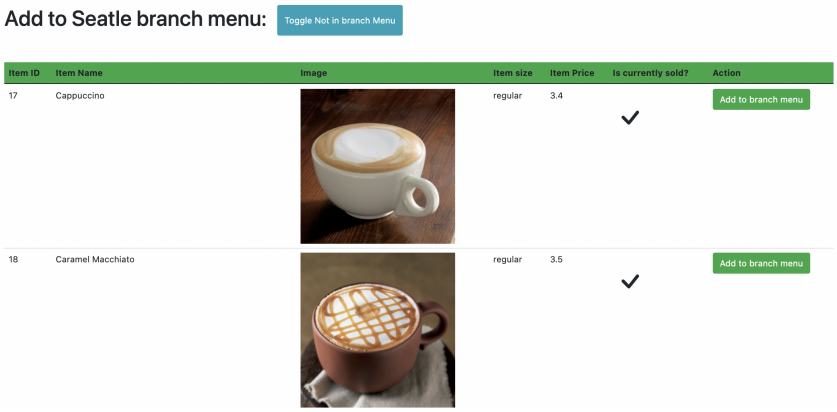
The dashboard shows overall summary for a given chain. The summary includes information about total customers and revenue in the last month, overall rating and latest transactions.



Branch Menu

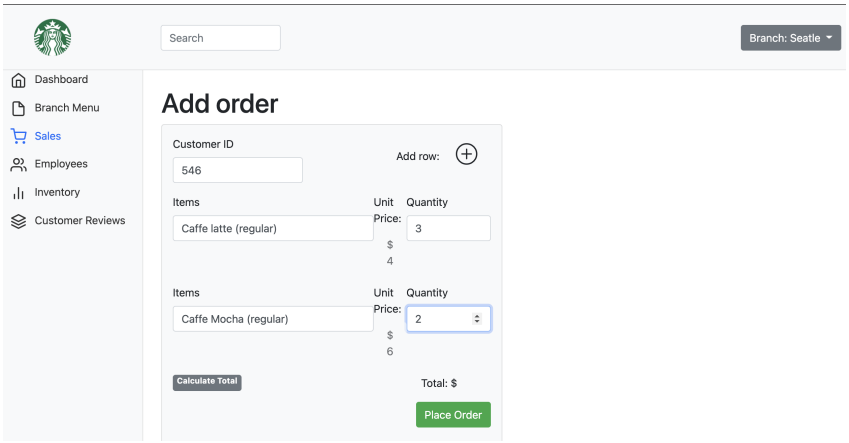
The branch menu shows information (item name, price, image, size) about items currently sold in the respective branch. The branch manager can remove items from their branch menu. The items not being sold in the branch are shown in a separate table that can be used to add items to the current branch menu.





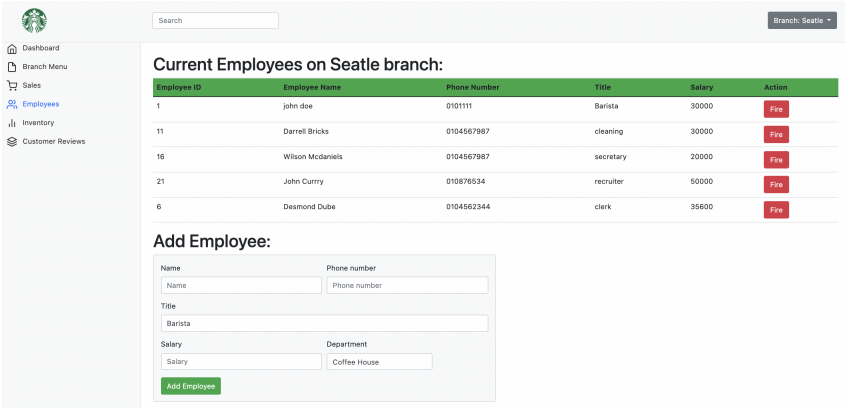
Sales

When a customer makes an order, the order information can be added to the Starbucks database using the sales tab. The order information includes customer id, order id (auto generated), items, quantity, and total price.



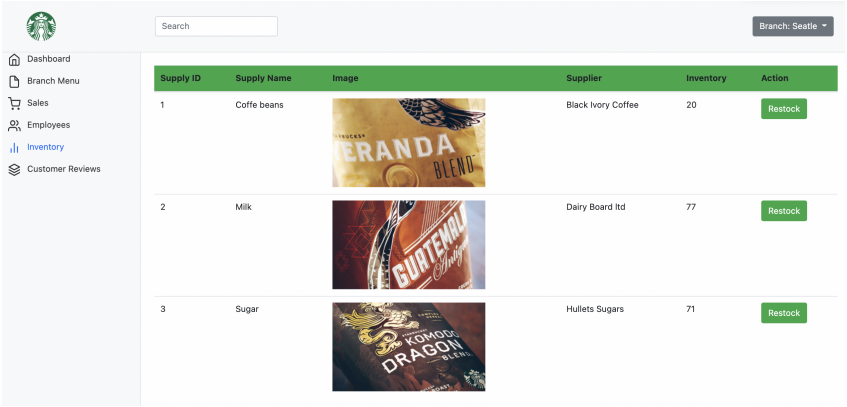
Employees

The branch manager can view information (employee id, name, phone number, salary, department) about all employees working in the respective branch. The manager can also add and remove employees from the branch.



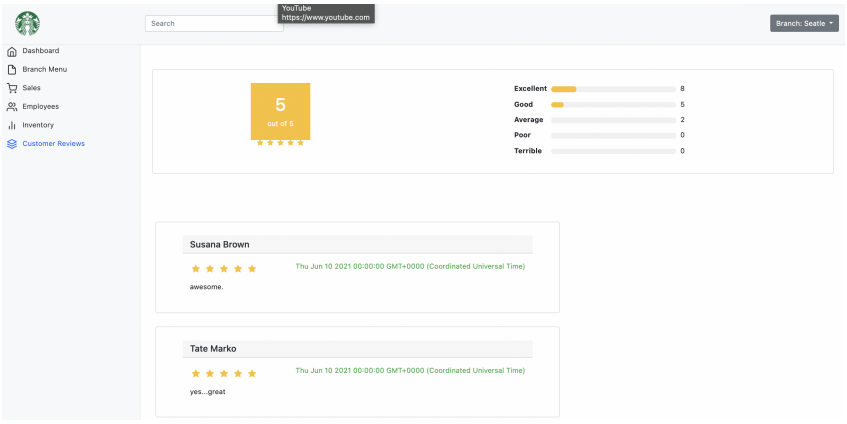
Inventory

The Inventory tab shows information about the current inventory of different supplies used at the Starbucks chain. When a supply is low in stock, the manager can send a restock request to the appropriate supplier to get the item restocked.



Customer Reviews

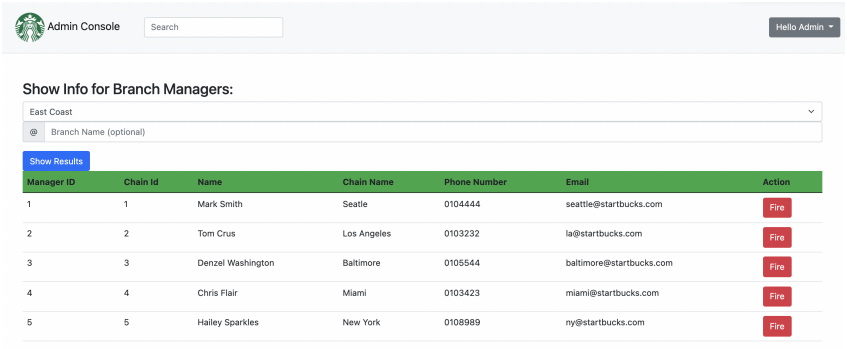
The Customer Reviews tab shows all the reviews from the customers for the Starbucks chain. The manager can view information like review date, reviewer name, and star ratings for each review. The manager can also see the overall rating for their branch.



1.4.2 Starbucks Corporation head (Admin) Perspective

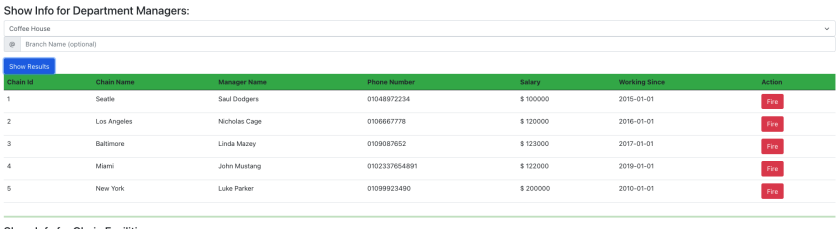
Branch Managers

The admin can view information (manager id, name, chain id, name, phone number, email) regarding all managers of Starbucks chains. If the admin wants to view manager information for a given chain, they can filter it by using chain name. The admin can also fire manager of a given chain.



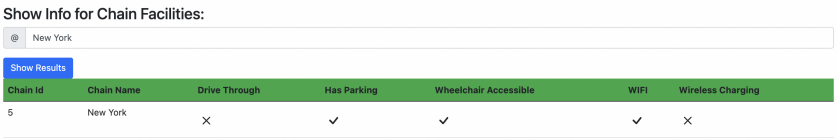
Department Managers

The admin can view information (department manager, chain id, name, phone number, email, salary, start date) regarding managers of a particular department of all Starbucks chains. If the admin wants to view department manager information for a given chain, they can filter it by using chain name. The admin can also fire department manager of a given chain.



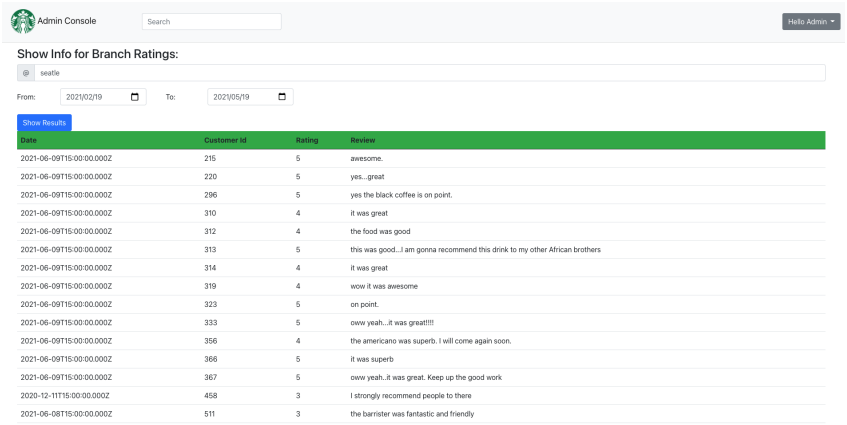
Chain Facilities

The admin can view information (Chain Id, Chain Name, Drive Through, Parking, Wheelchair accessibility, WIFI) regarding a particular Starbucks chain.



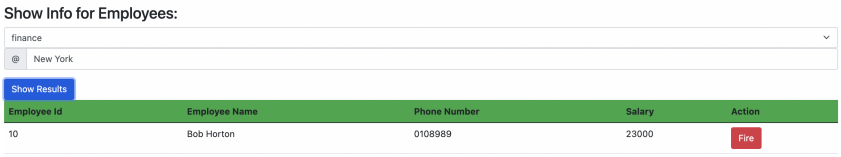
Branch Ratings

The admin can find all the customer reviews for a certain Starbucks branch for a specified time period.



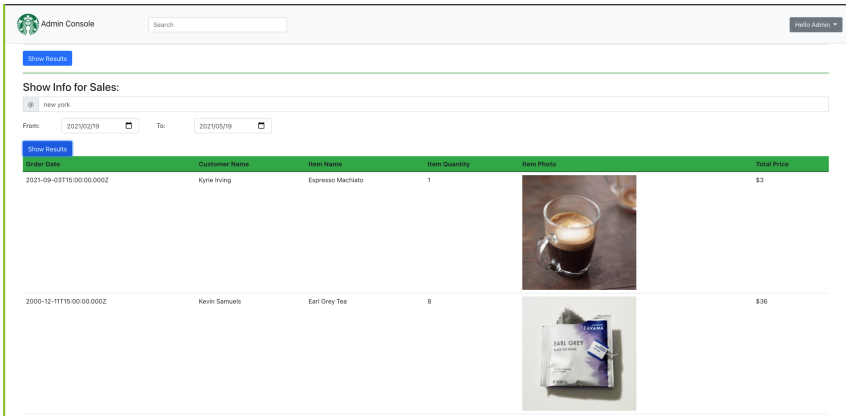
Employees

The admin can access and update information regarding employees in a certain department of a particular branch.



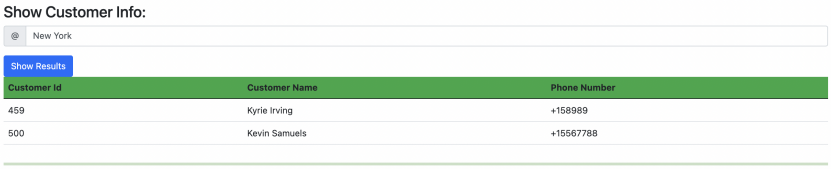
Sales

When a customer makes an order, the order information is added to the Starbucks database. The order information includes Order Date, Customer Name, Item Name, Item Quantity, Item Photo, and Total Price. This information is accessible to the admin from each branch for a specified period of time.



Customer Information

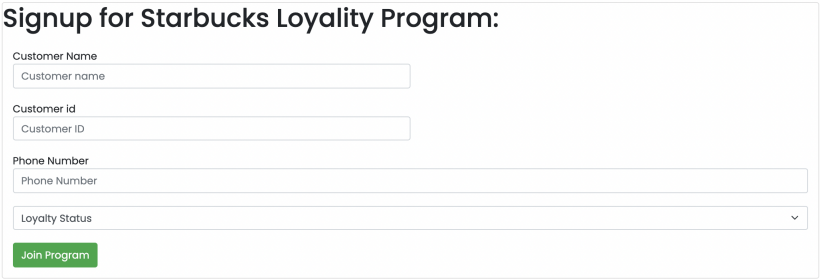
The information regarding the customers of a selected branch can be accessed here. The admin can gain more insights about the customers and make appropriate marketing.



1.4.3 Customers

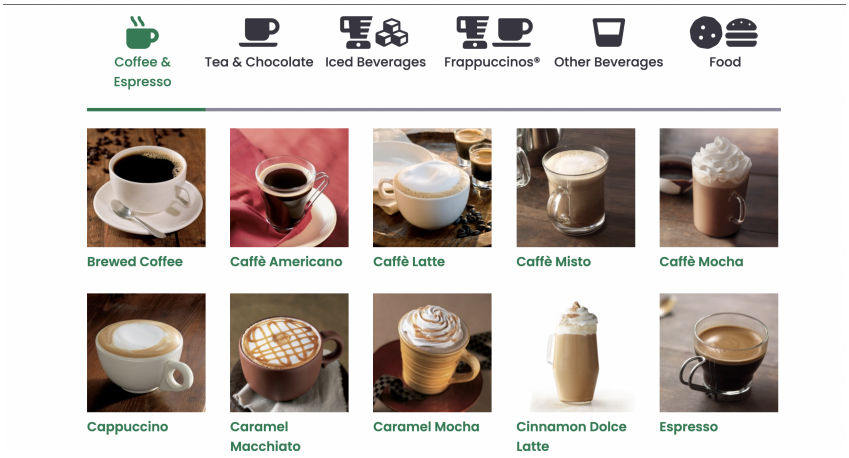
Starbucks Loyalty Program

Starbucks offers wide range of loyalty services, the customers can choose to be part of the services through this portal. The customers can also join as a regular member of Starbucks through this portal.



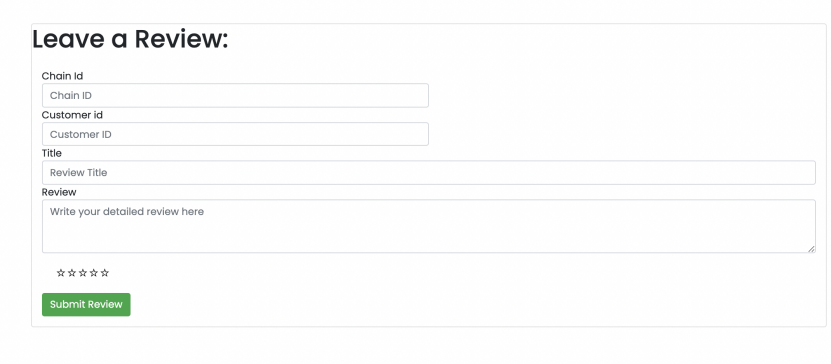
Menu

The customers will be able to view the menu for Starbucks in order to place an order.



Leave a Review

The customers can leave the review with appropriate feedback in this portal in order to improve the service of Starbucks near their location.



Chapter 2

Database Structure

2.1 ER Diagram

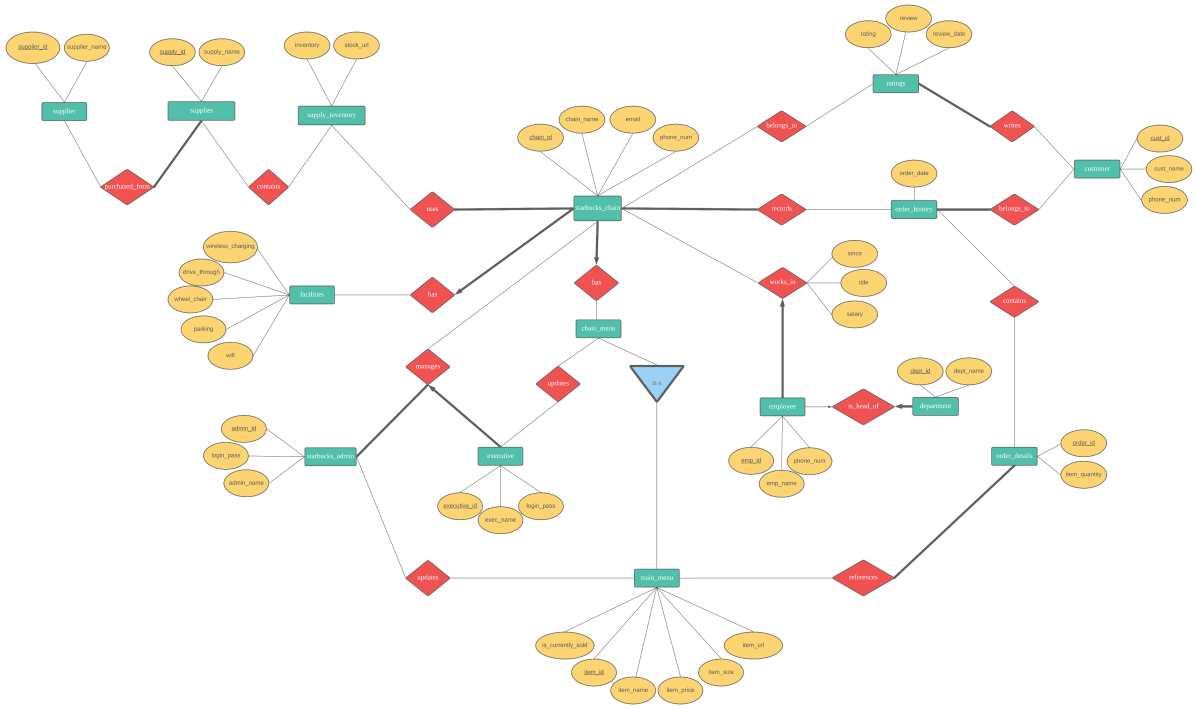


Figure 2.1: ER Diagram

2.2 Table Structure (before normalization)

2.2.1 employee

employee(emp_id, emp_name, phone_num, dept_id, dept_name, chain_id, since, title, salary)

Functional Dependencies:

- emp_id ⟶ all
- phone_num ⟶ all
- dept_id ⟶ (dept_name, chain_id)
- (dept_name, chain_id) ⟶ dept_id

Normal Form: 2NF

2.2.2 supply

supply(supplier_id, supplier_name, supply_id, supply_name, chain_id, inventory, stock_url)

Functional Dependencies:

- (chain_id, supply_id) ⟶ all
- supplier_id ⟶ supplier_name
- supply_id ⟶ (supply_name, supplier_id, stock_url)

Normal Form: 1NF

2.2.3 order

order(order_id, item_id, item_quantity, customer_id, order_date, chain_id)

Functional Dependencies:

- order_id ⟶ (customer_id, order_date, chain_id)
- (order_id, item_id) ⟶ all
- (item_id, customer_id, order_date) ⟶ item_quantity
- (customer_id, order_date) ⟶ order_id

Normal Form: 1NF

2.2.4 starbucks_admin

starbucks_admin(admin_id, admin_name, login_pass)

Functional Dependencies:

- admin_id ⟶ (admin_name, login_pass)

- (admin_name,login_pass) ⟶ admin_id

Normal Form: BCNF

2.2.5 starbucks_chain

starbucks_chain(chain_id, chain_name,phone_num, email)

Functional Dependencies:

- chain_id ⟶ (chain_name, phone_num, email)
- phone_num ⟶ (chain_id, chain_name, email)

Normal Form: BCNF

2.2.6 executive

executive(executive_id, login_pass, chain_id, executive_name)

Functional Dependencies:

- executive_id ⟶ (login_pass, chain_id, executive_name)
- chain_id ⟶ (login_pass, executive_name, executive_id)

Normal Form: BCNF

2.2.7 manages_chain

manages_chain(executive_id, chain_id)

Functional Dependencies:

- executive_id ⟶ chain_id
- chain_id ⟶ executive_id

Normal Form: BCNF

2.2.8 customer

customer(customer_id, customer_name, phone_num)

Functional Dependencies:

- customer_id ⟶ (customer_name, phone_num)

- phone_num $\longrightarrow$ (customer_id, customer_name)

Normal Form: BCNF

2.2.9 main_menu

main_menu(item_id, item_name, item_size, item_price, item_url, is_currently_sold)

Functional Dependencies:

- item_id $\longrightarrow$ (item_name, item_size, item_price, item_url, is_currently_sold)
- (item_name, item_size, item_price, item_url, is_currently_sold) $\longrightarrow$ item_id
- (item_name, item_size) $\longrightarrow$ (item_id, item_price, item_url, is_currently_sold)

Normal Form: BCNF

2.2.10 chain_menu

chain_menu(chain_id, item_id)

Functional Dependencies:

- (chain_id, item_id) $\longrightarrow$ all

Normal Form: BCNF

2.2.11 ratings

ratings(cust_id, chain_id, rating, review, review_date)

Functional Dependencies:

- (cust_id, chain_id) $\longrightarrow$ (rating, review, review_date)

Normal Form: BCNF

2.2.12 chain_facility

chain_facility(chain_id, drive_through, wheel_chair, has_parking, wireless_charging, WIFI)

Functional Dependencies:

- chain_id $\longrightarrow$ (drive_through, wheel_chair, has_parking, wireless_charging, WIFI)

Normal Form: BCNF

2.3 Normalization Process

Since we have the relations which are not in the desirable normal form, we need to decompose those relations into sub relation.

2.3.1 employee

employee(emp_id, emp_name, phone_num, dept_id, dept_name, chain_id, since, title, salary)
Functional Dependencies:

- emp_id $\longrightarrow$ all
- phone_num $\longrightarrow$ all
- dept_id $\longrightarrow$ (dept_name, chain_id)
- (dept_name, chain_id) $\longrightarrow$ dept_id

Normal Form: 2NF

Super keys: emp_id, phone_num

Prime attributes: emp_id, phone_num

Decomposing the relation using the 3rd functional dependency above.

$dept\_id^+$ = {dept_id, dept_name, chain_id}

R1{dept_id, dept_name, chain_id}

Functional Dependencies for **R1**:
dept_id $\longrightarrow$ dept_name, chain_id

R2{emp_id, emp_name, phone_num, since, title, salary, dept_id}

Functional Dependencies for **R2**:
emp_id $\longrightarrow$ all
phone_num $\longrightarrow$ all

Resulting relations:

- **R1**{dept_id, dept_name, chain_id}
- **R2**{emp_id, emp_name, phone_num, since, title, salary, dept_id}

2.3.2 supply

supply(supplier_id, supplier_name, supply_id, supply_name, chain_id, inventory, stock_url)

Functional Dependencies:

- (chain_id, supply_id) $\longrightarrow$ all
- supplier_id $\longrightarrow$ supplier_name
- supply_id $\longrightarrow$ (supply_name, supplier_id, stock_url)

Normal Form: 1NF

Super keys: (chain_id, supply_id)

Prime attributes: chain_id, supply_id

Decomposing the relation using the 2nd functional dependency above.

$supplier\_id^+$ = {supplier_name, supplier_id}

R1{supplier_name, supplier_id, supply_id}

Functional Dependencies for **R1**:

supplier_id $\longrightarrow$ supplier_name

D{supply_id, supply_name, chain_id, inventory, stock_url}

Functional Dependencies for D:

(chain_id, supply_id) $\longrightarrow$ all

supply_id $\longrightarrow$ (supply_name, stock_url) (1 NF)

Since one of the dependency is not in the desired form again so we decompose more using supply_id FD:

$supply\_id^+$ = {supply_id, supply_name, stock_url}

R2{supply_id, supply_name, stock_url}

Functional Dependencies for **R2**:

supply_id $\longrightarrow$ (supply_name, stock_url)

R3{supply_id, chain_id, inventory}

Functional Dependencies for **R3**:

supply_id, chain_id $\longrightarrow$ all

Resulting relations:

- **R1**{supplier_name, supplier_id, supply_id}
- **R2**{supply_id, supply_name, stock_url}
- **R3**{supply_id, chain_id, inventory}

2.3.3 order

order(order_id, item_id, item_quantity, customer_id, order_date, chain_id)

Functional Dependencies:

- order_id $\longrightarrow$ (customer_id, order_date, chain_id)
- (order_id, item_id) $\longrightarrow$ all
- (item_id, customer_id, order_date) $\longrightarrow$ item_quantity
- (customer_id, order_date) $\longrightarrow$ order_id

Normal Form: 1NF

Super keys: (order_id, item_id)

Prime attributes: order_id, item_id

Decomposing the relation using the 1st functional dependency above.

$order\_id^+$ = {order_id, customer_id, order_date, chain_id}

R2{order_id, customer_id, order_date, chain_id}

Functional Dependencies for R2:

- order_id $\longrightarrow$ (customer_id, order_date, chain_id)
- (customer_id, order_date) $\longrightarrow$ order_id

R1{order_id, item_id, item_quantity}

Functional Dependencies for **R1**:

- (order_id, item_id) $\longrightarrow$ item_quantity

Resulting relations:

- **R1**{order_id, item_id, item_quantity}
- **R2**{order_id, customer_id, order_date, chain_id}

As R2 is going to be used often to add order information, we decided to leave R2 in 3NF. Decomposing R2 more will make adding orders expensive, which we don't want.

2.4 Table Structure (after normalization)

2.4.1 employee

employee(emp_id, emp_name, phone_num, dept_id, dept_name, chain_id, since, title, salary)

Normal Form: 2NF

Decomposed into:

R1{dept_id, dept_name, chain_id}

Normal Form: BCNF

R2{emp_id, emp_name, phone_num, since, title, salary, dept_id}

Normal Form: BCNF

2.4.2 supply

supply(supplier_id, supplier_name, supply_id, supply_name, chain_id, inventory, stock_url)

Normal Form: 1NF

Decomposed into:

R1{supplier_name, supplier_id, supply_id}

Normal Form: BCNF

R2{supply_id, supply_name, stock_url}

Normal Form: BCNF

R3{supply_id, chain_id, inventory}

Normal Form: BCNF

2.4.3 order

order(order_id, item_id, item_quantity, customer_id, order_date, chain_id)

Normal Form: 1NF

Decomposed into:

R1{order_id, item_id, item_quantity}

Normal Form: BCNF

R2{order_id, customer_id, order_date, chain_id}

Normal Form: 3NF

2.4.4 starbucks_admin

starbucks_admin(admin_id, admin_name, login_pass)

Normal Form: BCNF

2.4.5 starbucks_chain

starbucks_chain(chain_id, chain_name,phone_num, email)

Normal Form: BCNF

2.4.6 executive

executive(executive_id, login_pass, chain_id, executive_name)

Normal Form: BCNF

2.4.7 manages_chain

manages_chain(executive_id, chain_id)

Normal Form: BCNF

2.4.8 customer

customer(customer_id, customer_name, phone_num)

Normal Form: BCNF

2.4.9 main_menu

main_menu(item_id, item_name, item_size, item_price, item_url, is_currently_sold)

Normal Form: BCNF

2.4.10 chain_menu

chain_menu(chain_id, item_id)

Normal Form: BCNF

2.4.11 ratings

ratings(cust_id, chain_id, rating, review, review_date)

Normal Form: BCNF

2.4.12 chain_facility

chain_facility(chain_id, drive_through, wheel_chair, has_parking, wireless_charging, WIFI)

Normal Form: BCNF

2.5 Impacts of Normalization

2.5.1 Lossless Join

employee(emp_id, emp_name, phone_num, dept_id, dept_name, chain_id, since, title, salary)

Resulting relations:

- **R1**{dept_id, dept_name, chain_id}
- **R2**{emp_id, emp_name, phone_num, since, title, salary, dept_id}

We started with the above employee relation which was in 2NF. After decomposition, we got relations R1 and R2, which are in BCNF. Since R1 and R2 have dept_id as the common attribute and dept_id is the primary key in R1, the join of two tables R1 and R2 will be lossless.

supply(supplier_id, supplier_name, supply_id, supply_name, chain_id, inventory, stock_url)

Resulting relations:

- **R1**{supplier_name, supplier_id, supply_id}
- **R2**{supply_id, supply_name, stock_url}
- **R3**{supply_id, chain_id, inventory}

We started with the above supply relation which was in 1NF. After decomposition, we got relations R1, R2 and R3, which are in BCNF. Since R1, R2 and R3 have supply_id as the common attribute and supply_id is the primary key in R2, the join of three tables R1, R2 and R3 will be lossless.

order(order_id, item_id, item_quantity, customer_id, order_date, chain_id)

Resulting relations:

- **R1**{order_id, item_id, item_quantity}
- **R2**{customer_id, order_date, order_id, chain_id}

We started with the above orders relation which was in 1NF. After decomposition, we got relations R1 and R2, which are in BCNF and 3NF respectively. Since R1 and R2 have order_id as the common attribute and order_id is the primary key in R2, the join of three tables R1 and R2 will be lossless.

2.5.2 Dependency Preservation

employee

employee(emp_id, emp_name, phone_num, dept_id, dept_name, chain_id, since, title, salary)

Functional Dependencies:

- emp_id $\longrightarrow$ all
- phone_num $\longrightarrow$ all
- dept_id $\longrightarrow$ (dept_name, chain_id)
- (dept_name, chain_id) $\longrightarrow$ dept_id

Resulting relations:

R1{dept_id, dept_name, chain_id}

Functional Dependencies for **R1**:

dept_id $\longrightarrow$ dept_name, chain_id

(dept_name, chain_id) $\longrightarrow$ dept_id

R2{emp_id, emp_name, phone_num, since, title, salary, dept_id}

Functional Dependencies for **R2**:

emp_id $\longrightarrow$ all

phone_num $\longrightarrow$ all

Since the union of Functional dependencies of R1 and R2 add up to give the functional dependencies in the original relation and vice versa, this decomposition is dependency preserving.

supply

supply(supplier_id, supplier_name, supply_id, supply_name, chain_id, inventory, stock_url)
Functional Dependencies:

- (chain_id, supply_id) $\longrightarrow$ all
- supplier_id $\longrightarrow$ supplier_name
- supply_id $\longrightarrow$ (supply_name, supplier_id, stock_url)

Resulting relations:

R1{supplier_name, supplier_id, supply_id}

Functional Dependencies for **R1**:

supplier_id $\longrightarrow$ supplier_name supply_id $\longrightarrow$ all

R2{supply_id, supply_name, stock_url}

Functional Dependencies for **R2**:

supply_id $\longrightarrow$ (supply_name, stock_url)

R3{supply_id, chain_id, inventory}

Functional Dependencies for **R3**:

supply_id, chain_id $\longrightarrow$ all

Since the union of Functional dependencies of R1, R2, and R3 add up to give the functional dependencies in the original relation and vice versa, this decomposition is dependency preserving.

order

order(order_id, item_id, item_quantity, customer_id, order_date, chain_id)

Functional Dependencies:

- order_id $\longrightarrow$ (customer_id, order_date, chain_id)
- (order_id, item_id) $\longrightarrow$ all
- (item_id, customer_id, order_date) $\longrightarrow$ item_quantity
- (customer_id, order_date) $\longrightarrow$ order_id

Resulting relations:

R1{order_id, item_id, item_quantity}

Functional Dependencies for **R1**:

- (order_id, item_id) $\longrightarrow$ item_quantity

R2{order_id, customer_id, order_date, chain_id}

Functional Dependencies for **R2**:

- order_id $\longrightarrow$ (customer_id, order_date, chain_id)
- (customer_id, order_date) $\longrightarrow$ order_id

Since the union of Functional dependencies of R1 and R2 add up to give the functional dependencies in the original relation and vice versa, this decomposition is dependency preserving.

2.5.3 Trade-offs vs Advantages

In the employee relation, before normalization, a query of form "Given the phone number of an employee, find the employee name and the department they work in", would not need any join and thus be fast to compute. But after decomposition into R1 and R2, we need a join to compute the same query. As a result, the transaction is slower.
In the supply relation, before normalization, a query of form "Given the supply name, find the supplier name", would not need any join and thus be fast to compute. But after decomposition into R1, R2 and R3, we need a join to compute the same query. As a result, the transaction is slower.

In the order relation, R2 is in 3NF only. If we decompose R2 further, for frequently computed queries of form "Given an order id, find the chain id", will need a join and thus be slower to compute. In order to make this frequently used query faster, we chose to stick with 3NF.

Chapter 3

Database Features

3.1 Constraints

3.1.1 Primary Keys and Foreign Keys

Some examples of key constraints used in our database are:

```
CREATE TABLE works_in (
    emp_id VARCHAR(100),
    dept_id VARCHAR(100),
    chain_id INT,
    since VARCHAR(60),
    title VARCHAR(60),
    salary REAL,
    PRIMARY KEY (emp_id),
    FOREIGN KEY (emp_id) REFERENCES employee(emp_id)
        ON DELETE CASCADE,
    FOREIGN KEY (chain_id) REFERENCES  starbucks_chain(chain_id),
    FOREIGN KEY (dept_id) REFERENCES  department(dept_id)

);
```

```
CREATE TABLE order_hist (
    order_id VARCHAR(100),
    cust_id INT,
    order_date DATE,
    chain_id INT,
    PRIMARY KEY (order_id),
    FOREIGN KEY (cust_id) REFERENCES  customer(cust_id),
    FOREIGN KEY (order_id) REFERENCES  order_hist_details(order_id),
    FOREIGN KEY (chain_id) REFERENCES  starbucks_chain(chain_id)
);
```

3.1.2 Checks

Some examples of check constraints used in our database are:

```
CREATE TABLE main_menu (
    item_id INT,
    item_name VARCHAR(60),
    item_size VARCHAR(20),
    item_price REAL,
    item_url VARCHAR(2000),
    is_currently_sold BOOLEAN NOT NULL,
    PRIMARY KEY (item_id),
    CONSTRAINT IS_VALID_PRICE CHECK (item_price>0)
);
```

```
CREATE TABLE employee (
    emp_id VARCHAR(100),
    emp_name VARCHAR(60),
    phone_num VARCHAR(110),
    PRIMARY KEY (emp_id),
    UNIQUE (phone_num)
);
```

3.1.3 Assertions

Some examples of assertions used in our database are:

```
CREATE ASSERTION isValidHire
    CHECK (datediff(curdate(), since) > 0 AND
        salary > 10000 FROM works_in)
);
```

```
CREATE ASSERTION isValidOrder
    CHECK (1 <= ALL
    (SELECT SUM(item_quantity)
    FROM order_hist_details
    GROUP BY order_id )
);
```

3.2 Triggers

Some examples of triggers used in our database are:

```
CREATE TRIGGER InsertIntoBranchMenuCheck
AFTER UPDATE ON main_menu
REFERENCING NEW ROW AS NewTuple
FOR EACH ROW
WHEN (is_currently_sold <> NULL IN
    (SELECT is_currently_sold
    FROM main_menu))
INSERT INTO chain_menu
VALUES(NewTuple);
```

```
CREATE TRIGGER orderCheck
AFTER INSERT ON order_hist_det
REFERENCING NEW ROW AS NewTuple
FOR EACH ROW
WHEN (1 <= ALL
    (SELECT SUM(item_quantity)
    FROM order_hist_details
    GROUP BY order_id ))
INSERT INTO order_hist_details
VALUES(NewTuple);
```

3.3 Views

Some examples of views used in our database are:

```
CREATE VIEW orderDetails
    AS
    SELECT order_date, item_quantity, item_quantity, customer_id, order_price
    FROM order_hist H, order_hist_details D WHERE
    H.order_id = D.order_id;
```

```
CREATE VIEW supplyInventory
    AS
    SELECT supply_name, supply_id, inventory, stock_url, supplier_name
    FROM supply_inventory I, supplies S, supplier SP WHERE
    I.supply_id = S.supply_id AND S.supplier_id = SP.supplier_id;
```

Chapter 4

Future Improvements & Conclusion

The design which we have worked on is focused more on management side and less on customer side. An improvement to the system can be the addition of a better user interface and features for customers.

Since performing database queries involves the front end communicating with back end, our choice of technology (Vanilla.js) was limiting because performing API Requests was hard using script tags only. A future improvement to this might be to work with an advanced JS version like React. Also for the back end, since we are using REST APIs, we have different API routes for different operations. In a large scale application, having too many REST APIs might not be as efficient as too many back end requests will be created. A future improvement to this is to use GraphQL, which we could have done if we had more time.

Also, due to the limited time, we could only add a limited set of CRUD operations. With more time, more functionalities for the Starbucks head/Admin could be added.

We would like to express our sincere gratitude to **Professor Arthur Lee**, who introduced us to the fundamentals of database systems and allowed us to work as a team for this group project.

4.0.1 Github Repository and Youtube Demo

The Github Repository can be found at
https://github.com/iamroshanpoudel/StarbucksChainManagementSystem.git