

The State University of New York, Korea

# Starbucks Chain Management System

Roshan Poudel, Nilson Chapagain, Crispin Chisina

Bachelors of Science in Computer Science

June 10, 2021

Professor Arthur Lee

Department of Computer Science



Stony Brook University



# **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

<b>1 Project Overview</b>	<b>1</b>
1.1 Introduction . . . . .	1
1.2 Requirements . . . . .	1
1.3 Tech Stack . . . . .	2
1.4 User Perspectives . . . . .	2
1.4.1 Chain Manager Perspective / Guide . . . . .	2
1.4.2 Starbucks Corporation head (Admin) Perspective . . . . .	6
1.4.3 Customers . . . . .	9
<b>2 Database Structure</b>	<b>11</b>
2.1 ER Diagram . . . . .	11
2.2 Table Structure (before normalization) . . . . .	11
2.2.1 employee . . . . .	11
2.2.2 supply . . . . .	12
2.2.3 order . . . . .	12
2.2.4 starbucks_admin . . . . .	12
2.2.5 starbucks_chain . . . . .	13
2.2.6 executive . . . . .	13
2.2.7 manages_chain . . . . .	13
2.2.8 customer . . . . .	13
2.2.9 main_menu . . . . .	14
2.2.10 chain_menu . . . . .	14
2.2.11 ratings . . . . .	14
2.2.12 chain_facility . . . . .	14

2.3	Normalization Process . . . . .	15
2.3.1	employee . . . . .	15
2.3.2	supply . . . . .	16
2.3.3	order . . . . .	17
2.4	Table Structure (after normalization) . . . . .	18
2.4.1	employee . . . . .	18
2.4.2	supply . . . . .	18
2.4.3	order . . . . .	18
2.4.4	starbucks_admin . . . . .	19
2.4.5	starbucks_chain . . . . .	19
2.4.6	executive . . . . .	19
2.4.7	manages_chain . . . . .	19
2.4.8	customer . . . . .	19
2.4.9	main_menu . . . . .	19
2.4.10	chain_menu . . . . .	20
2.4.11	ratings . . . . .	20
2.4.12	chain_facility . . . . .	20
2.5	Impacts of Normalization . . . . .	20
2.5.1	Lossless Join . . . . .	20
2.5.2	Dependency Preservation . . . . .	21
2.5.3	Trade-offs vs Advantages . . . . .	23
<b>3</b>	<b>Database Features . . . . .</b>	<b>25</b>
3.1	Constraints . . . . .	25
3.1.1	Primary Keys and Foreign Keys . . . . .	25
3.1.2	Checks . . . . .	26
3.1.3	Assertions . . . . .	27
3.2	Triggers . . . . .	28
3.3	Views . . . . .	29
<b>4</b>	<b>Future Improvements &amp; Conclusion . . . . .</b>	<b>30</b>
4.0.1	Github Repository and Youtube Demo . . . . .	30

# **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

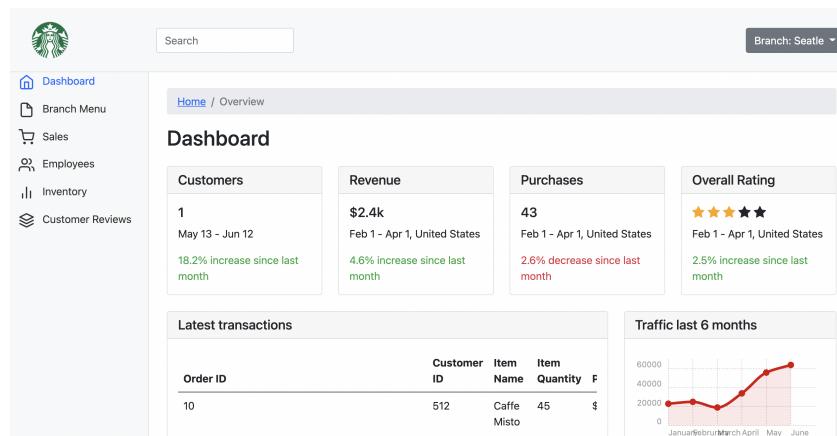
The project maintains three levels of users:

- 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.

This screenshot shows the 'Seattle branch menu' section of a software application. At the top, there's a navigation bar with a Starbucks logo, a search bar, and a dropdown menu set to 'Branch: Seattle'. On the left, a sidebar lists 'Dashboard', 'Branch Menu' (which is selected and highlighted in blue), 'Sales', 'Employees', 'Inventory', and 'Customer Reviews'. The main area is titled 'Seattle branch menu:' with a 'Toggle branch Menu' button. A table displays two items:

Item ID	Item Name	Image	Item size	Item Price	Action
12	Brewed Coffee		regular	3	<button>Remove from menu</button>
13	Caffe latte		regular	4	<button>Remove from menu</button>

This screenshot shows the 'Add to Seattle branch menu' section. At the top, it says 'Add to Seattle branch menu:' with a 'Toggle Not in branch Menu' button. Below is a table with two items:

Item ID	Item Name	Image	Item size	Item Price	Is currently sold?	Action
17	Cappuccino		regular	3.4	✓	<button>Add to branch menu</button>
18	Caramel Macchiato		regular	3.5	✓	<button>Add to branch menu</button>

## 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.

The screenshot shows the Starbucks Sales interface with the following details:

- Customer ID:** 546
- Items:**
  - Caffe latte (regular)
  - Caffe Mocha (regular)
- Unit:** (dropdown menu)
- Quantity:** (dropdown menu)
- Price:** (dropdown menu)
  - 3
  - 4
  - 2
  - 6
- Total:** \$ (displayed as \$12.00)
- Buttons:** Calculate Total, Place Order

## 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.

The screenshot shows the Starbucks Employees interface with the following details:

Current Employees on Seattle branch:					
Employee ID	Employee Name	Phone Number	Title	Salary	Action
1	john doe	010111	Barista	30000	<span style="color: red;">Fire</span>
11	Darrell Bricks	0104567887	cleaning	30000	<span style="color: red;">Fire</span>
16	Wilson McDanielis	0104567887	secretary	20000	<span style="color: red;">Fire</span>
21	John Curry	010876534	recruiter	50000	<span style="color: red;">Fire</span>
6	Desmond Dubé	0104562344	clerk	35000	<span style="color: red;">Fire</span>

**Add Employee:**

Name:   
Phone number:

Title:   
Barista

Salary:   
Department:   
Coffee House

**Buttons:** Add Employee

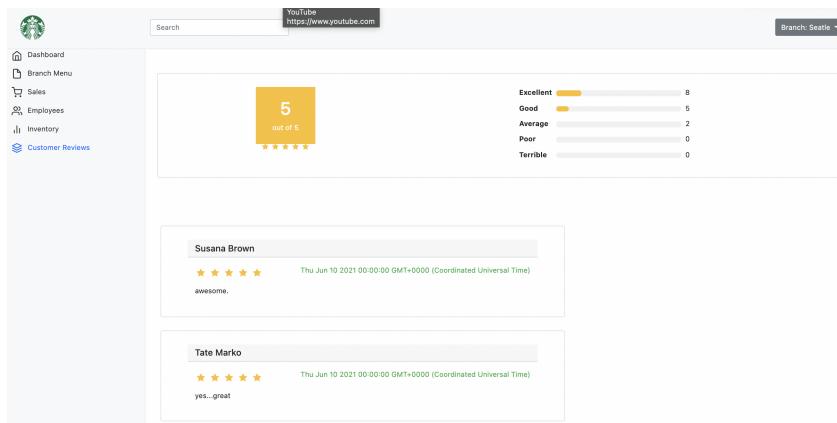
## Inventory

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.

Supply ID	Supply Name	Image	Supplier	Inventory	Action
1	Coffee beans		Black Ivory Coffee	20	<button>Restock</button>
2	Milk		Dairy Board ltd	77	<button>Restock</button>
3	Sugar		Hullets Sugars	71	<button>Restock</button>

## 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.

Manager ID	Chain Id	Name	Chain Name	Phone Number	Email	Action
1	1	Mark Smith	Seattle	0104444	seattle@starbucks.com	<button>Fire</button>
2	2	Tom Crus	Los Angeles	0103232	la@starbucks.com	<button>Fire</button>
3	3	Denzel Washington	Baltimore	0105544	baltimore@starbucks.com	<button>Fire</button>
4	4	Chris Flair	Miami	0103423	miami@starbucks.com	<button>Fire</button>
5	5	Hailey Sparkles	New York	0108989	ny@starbucks.com	<button>Fire</button>

#### 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 Id	Chain Name	Manager Name	Phone Number	Salary	Working Since	Action
1	Seattle	Saul Dodger	01048972234	\$100000	2015-01-01	<button>Fire</button>
2	Los Angeles	Nicholas Cage	0106667778	\$120000	2016-01-01	<button>Fire</button>
3	Baltimore	Linda Mazy	0109087652	\$123000	2017-01-01	<button>Fire</button>
4	Miami	John Moutang	0102337054891	\$122000	2018-01-01	<button>Fire</button>
5	New York	Luke Parker	01099923490	\$200000	2010-01-01	<button>Fire</button>

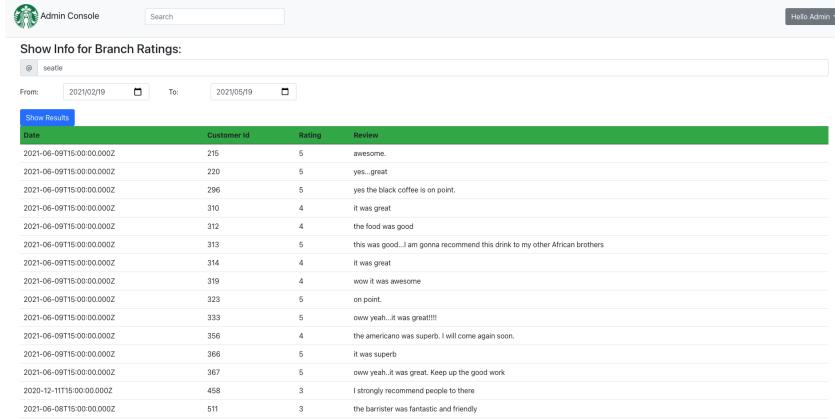
#### Chain Facilities

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

Chain Id	Chain Name	Drive Through	Has Parking	Wheelchair Accessible	WIFI	Wireless Charging
5	New York	X	✓	✓	✓	X

## Branch Ratings

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



Show Info for Branch Ratings:

From: 2021/02/19 To: 2021/05/19

**Customer Reviews for Seattle**

Date	Customer Id	Rating	Review
2021-06-09T15:00:00.000Z	215	5	awesome.
2021-06-09T15:00:00.000Z	220	5	yes...great
2021-06-09T15:00:00.000Z	296	5	yes the black coffee is on point.
2021-06-09T15:00:00.000Z	310	4	it was great
2021-06-09T15:00:00.000Z	312	4	the food was good
2021-06-09T15:00:00.000Z	313	5	this was good...I am gonna recommend this drink to my other African brothers
2021-06-09T15:00:00.000Z	314	4	it was great
2021-06-09T15:00:00.000Z	319	4	wow it was awesome
2021-06-09T15:00:00.000Z	323	5	on point.
2021-06-09T15:00:00.000Z	333	5	oww yeah...it was great!!!!
2021-06-09T15:00:00.000Z	356	4	the americano was superb. I will come again soon.
2021-06-09T15:00:00.000Z	366	5	it was superb
2021-06-09T15:00:00.000Z	367	5	oww yeah...it was great. Keep up the good work
2020-12-11T15:00:00.000Z	458	3	I strongly recommend people to there
2021-06-08T15:00:00.000Z	511	3	the barrister was fantastic and friendly

## Employees

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



Show Info for Employees:

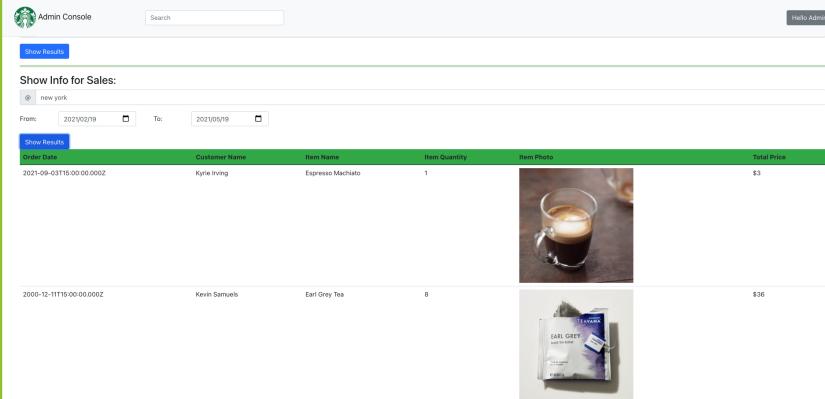
finance  
@ New York

**Employee Details**

Employee Id	Employee Name	Phone Number	Salary	Action
10	Bob Horton	0108989	23000	<button>Fire</button>

## 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.



The screenshot shows a table with two rows of sales data. The columns are Order Date, Customer Name, Item Name, Item Quantity, Item Photo, and Total Price.

Order Date	Customer Name	Item Name	Item Quantity	Item Photo	Total Price
2021-09-03T15:00:00.000Z	Kyrie Irving	Espresso Macchiato	1		\$3
2020-12-11T19:00:00.000Z	Kevin Samuels	Earl Grey Tea	8		\$36

## 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.



The screenshot shows a table with two rows of customer information. The columns are Customer Id, Customer Name, and Phone Number.

Customer Id	Customer Name	Phone Number
459	Kyrie Irving	+158989
500	Kevin Samuels	+15567788

### 1.4.3 Customers

#### Starbucks Loyalty Program

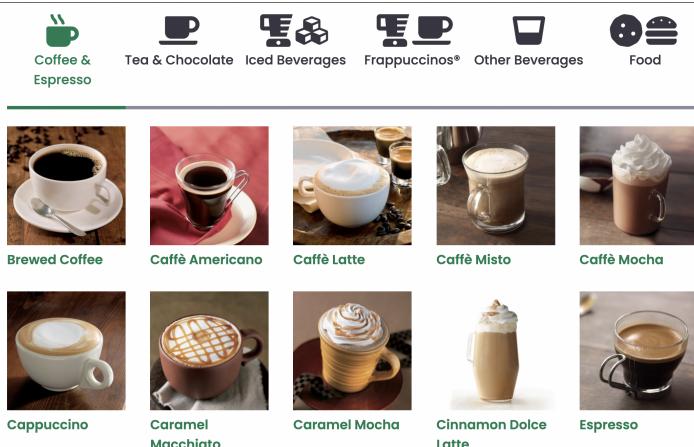
Starbucks offers wide range of loyalty services, the customers can choose to be part of the services through this portal. The customers can also join as a regular member of Starbucks through this portal.

##### Signup for Starbucks Loyalty Program:

Customer Name	<input type="text" value="Customer name"/>
Customer id	<input type="text" value="Customer ID"/>
Phone Number	<input type="text" value="Phone Number"/>
Loyalty Status	<input type="text"/>
<input type="button" value="Join Program"/>	

#### 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.

**Leave a Review:**

Chain Id

Customer id

Title

Review

Write your detailed review here

☆☆☆☆☆

**Submit Review**

# Chapter 2

# Database Structure

## 2.1 ER Diagram

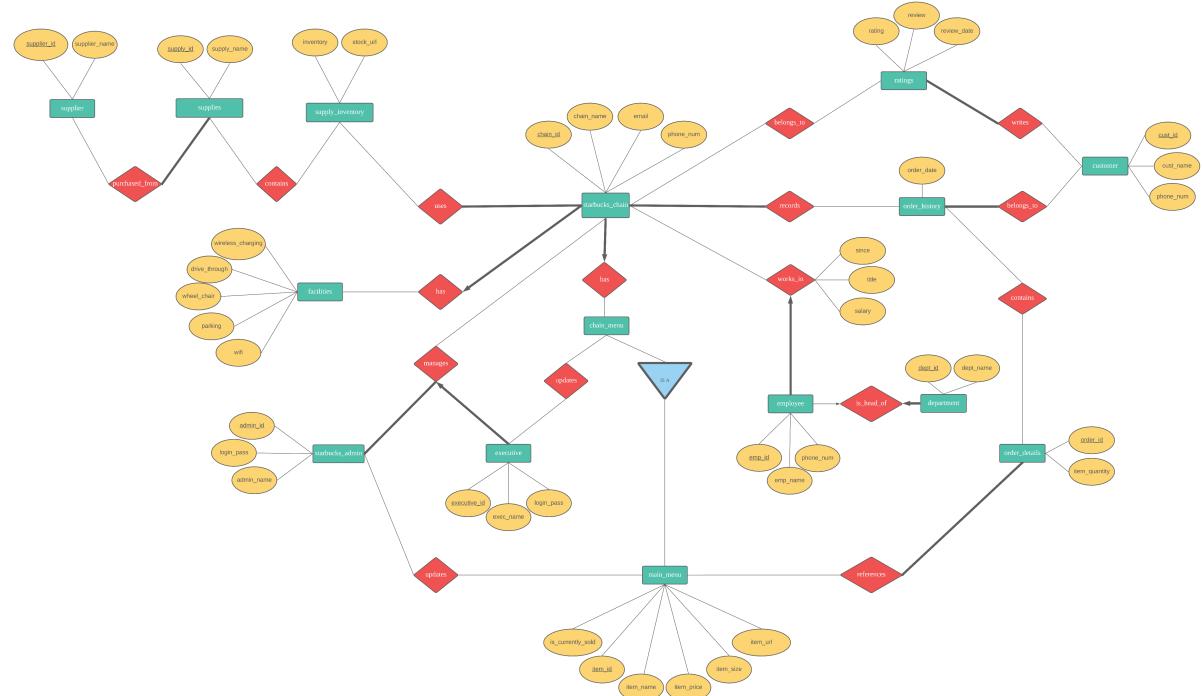


Figure 2.1: ER Diagram

## 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)

- $(\text{admin\_name}, \text{login\_pass}) \rightarrow \text{admin\_id}$

**Normal Form:** BCNF

### 2.2.5 starbucks\_chain

`starbucks_chain( chain_id, chain_name, phone_num, email)`

Functional Dependencies:

- $\text{chain\_id} \rightarrow (\text{chain\_name}, \text{phone\_num}, \text{email})$
- $\text{phone\_num} \rightarrow (\text{chain\_id}, \text{chain\_name}, \text{email})$

**Normal Form:** BCNF

### 2.2.6 executive

`executive( executive_id, login_pass, chain_id, executive_name)`

Functional Dependencies:

- $\text{executive\_id} \rightarrow (\text{login\_pass}, \text{chain\_id}, \text{executive\_name})$
- $\text{chain\_id} \rightarrow (\text{login\_pass}, \text{executive\_name}, \text{executive\_id})$

**Normal Form:** BCNF

### 2.2.7 manages\_chain

`manages_chain(executive_id, chain_id)`

Functional Dependencies:

- $\text{executive\_id} \rightarrow \text{chain\_id}$
- $\text{chain\_id} \rightarrow \text{executive\_id}$

**Normal Form:** BCNF

### 2.2.8 customer

`customer( customer_id, customer_name, phone_num)`

Functional Dependencies:

- $\text{customer\_id} \rightarrow (\text{customer\_name}, \text{phone\_num})$

- phone\_num → (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 → (item\_name, item\_size, item\_price, item\_url, is\_currently\_sold)
- (item\_name, item\_size, item\_price, item\_url, is\_currently\_sold) → item\_id
- (item\_name, item\_size) → (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) → all

**Normal Form:** BCNF

### 2.2.11 ratings

ratings( cust\_id, chain\_id, rating, review, review\_date)

Functional Dependencies:

- (cust\_id, chain\_id) → (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 → (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:

- $\text{emp\_id} \rightarrow \text{all}$
- $\text{phone\_num} \rightarrow \text{all}$
- $\text{dept\_id} \rightarrow (\text{dept\_name}, \text{chain\_id})$
- $(\text{dept\_name}, \text{chain\_id}) \rightarrow \text{dept\_id}$

**Normal Form:** 2NF

**Super keys:** `emp_id, phone_num`

**Prime attributes:** `emp_id, phone_num`

Decomposing the relation using the 3rd functional dependency above.

$$\text{dept\_id}^+ = \{\text{dept\_id}, \text{dept\_name}, \text{chain\_id}\}$$

$$\mathbf{R1}\{\text{dept\_id}, \text{dept\_name}, \text{chain\_id}\}$$

Functional Dependencies for **R1**:

$$\text{dept\_id} \rightarrow \text{dept\_name}, \text{chain\_id}$$

$$\mathbf{R2}\{\text{emp\_id}, \text{emp\_name}, \text{phone\_num}, \text{since}, \text{title}, \text{salary}, \text{dept\_id}\}$$

Functional Dependencies for **R2**:

$$\text{emp\_id} \rightarrow \text{all}$$

$$\text{phone\_num} \rightarrow \text{all}$$

**Resulting relations:**

- $\mathbf{R1}\{\text{dept\_id}, \text{dept\_name}, \text{chain\_id}\}$
- $\mathbf{R2}\{\text{emp\_id}, \text{emp\_name}, \text{phone\_num}, \text{since}, \text{title}, \text{salary}, \text{dept\_id}\}$

### 2.3.2 supply

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

Functional Dependencies:

- $(chain\_id, supply\_id) \rightarrow all$
- $supplier\_id \rightarrow supplier\_name$
- $supply\_id \rightarrow (supply\_name, supplier\_id, stock\_url)$

**Normal Form:** 1NF

**Super keys:**  $(chain\_id, supply\_id)$

**Prime attributes:**  $chain\_id, supply\_id$

Decomposing the relation using the 2nd functional dependency above.

$$supplier\_id^+ = \{supplier\_name, supplier\_id\}$$

$$\mathbf{R1}\{supplier\_name, supplier\_id, supply\_id\}$$

Functional Dependencies for **R1**:

$$supplier\_id \rightarrow supplier\_name$$

$$D\{supply\_id, supply\_name, chain\_id, inventory, stock\_url\}$$

Functional Dependencies for D:

$$(chain\_id, supply\_id) \rightarrow all$$

$$supply\_id \rightarrow (supply\_name, stock\_url) \text{ (1 NF)}$$

Since one of the dependency is not in the desired form again so we decompose more using  $supply\_id$  FD:

$$supply\_id^+ = \{supply\_id, supply\_name, stock\_url\}$$

$$\mathbf{R2}\{supply\_id, supply\_name, stock\_url\}$$

Functional Dependencies for **R2**:

$$supply\_id \rightarrow (supply\_name, stock\_url)$$

$$\mathbf{R3}\{supply\_id, chain\_id, inventory\}$$

Functional Dependencies for **R3**:

$\text{supply\_id}, \text{chain\_id} \rightarrow \text{all}$

**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

$\text{order}(\underline{\text{order\_id}}, \underline{\text{item\_id}}, \text{item\_quantity}, \text{customer\_id}, \text{order\_date}, \text{chain\_id})$

Functional Dependencies:

- $\text{order\_id} \rightarrow (\text{customer\_id}, \text{order\_date}, \text{chain\_id})$
- $(\text{order\_id}, \text{item\_id}) \rightarrow \text{all}$
- $(\text{item\_id}, \text{customer\_id}, \text{order\_date}) \rightarrow \text{item\_quantity}$
- $(\text{customer\_id}, \text{order\_date}) \rightarrow \text{order\_id}$

**Normal Form:** 1NF

**Super keys:** (order\_id, item\_id)

**Prime attributes:** order\_id, item\_id

Decomposing the relation using the 1st functional dependency above.

$\text{order\_id}^+ = \{\text{order\_id}, \text{customer\_id}, \text{order\_date}, \text{chain\_id}\}$

**R2**{order\_id, customer\_id, order\_date, chain\_id}

Functional Dependencies for R2:

- $\text{order\_id} \rightarrow (\text{customer\_id}, \text{order\_date}, \text{chain\_id})$
- $(\text{customer\_id}, \text{order\_date}) \rightarrow \text{order\_id}$

**R1**{order\_id, item\_id, item\_quantity}

Functional Dependencies for **R1**:

- $(\text{order\_id}, \text{item\_id}) \rightarrow \text{item\_quantity}$

**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:**

- $\text{emp\_id} \rightarrow \text{all}$
- $\text{phone\_num} \rightarrow \text{all}$
- $\text{dept\_id} \rightarrow (\text{dept\_name}, \text{chain\_id})$
- $(\text{dept\_name}, \text{chain\_id}) \rightarrow \text{dept\_id}$

**Resulting relations:**

**R1**{dept\_id, dept\_name, chain\_id}

Functional Dependencies for **R1**:

$\text{dept\_id} \rightarrow \text{dept\_name}, \text{chain\_id}$   
 $(\text{dept\_name}, \text{chain\_id}) \rightarrow \text{dept\_id}$

**R2**{emp\_id, emp\_name, phone\_num, since, title, salary, dept\_id}

Functional Dependencies for **R2**:

$\text{emp\_id} \rightarrow \text{all}$

$\text{phone\_num} \rightarrow \text{all}$

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:

- $(\text{chain\_id}, \text{supply\_id}) \rightarrow \text{all}$
- $\text{supplier\_id} \rightarrow \text{supplier\_name}$
- $\text{supply\_id} \rightarrow (\text{supply\_name}, \text{supplier\_id}, \text{stock\_url})$

### Resulting relations:

**R1**{supplier\_name, supplier\_id, supply\_id}

Functional Dependencies for **R1**:

$\text{supplier\_id} \rightarrow \text{supplier\_name}$   $\text{supply\_id} \rightarrow \text{all}$

**R2**{supply\_id, supply\_name, stock\_url}

Functional Dependencies for **R2**:

$\text{supply\_id} \rightarrow (\text{supply\_name}, \text{stock\_url})$

**R3**{supply\_id, chain\_id, inventory}

Functional Dependencies for **R3**:

$\text{supply\_id}, \text{chain\_id} \rightarrow \text{all}$

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:

- $\text{order\_id} \rightarrow (\text{customer\_id}, \text{order\_date}, \text{chain\_id})$
- $(\text{order\_id}, \text{item\_id}) \rightarrow \text{all}$
- $(\text{item\_id}, \text{customer\_id}, \text{order\_date}) \rightarrow \text{item\_quantity}$
- $(\text{customer\_id}, \text{order\_date}) \rightarrow \text{order\_id}$

**Resulting relations:**

**R1**{`order_id, item_id, item_quantity`}

Functional Dependencies for **R1**:

- $(\text{order\_id}, \text{item\_id}) \rightarrow \text{item\_quantity}$

**R2**{`order_id, customer_id, order_date, chain_id`}

Functional Dependencies for **R2**:

- $\text{order\_id} \rightarrow (\text{customer\_id}, \text{order\_date}, \text{chain\_id})$
- $(\text{customer\_id}, \text{order\_date}) \rightarrow \text{order\_id}$

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>