CSDS 341 Final Report

Food Supply Chain Management System

Prepared by: David Casente, James Kennelly, Michael O'Brien

December 6, 2020

Background

There are many services that help feed the CWRU population. Some of these services include Rough Rider, Melt, L3, etc. The managers of this restaurant network need to send ingredients to each store to make sure they have enough stock to serve their customers. The purpose of our database is to help keep track of the stock at each restaurant to aid the supply chain management of the restaurant network. In order to do this the database records each order made at the restaurants and updates the stock by removing the ingredients used in each order.

The two main interfaces of the database are the Restaurant Page, which is used by servers at the restaurants, and the Manager Page, which is used by managers of the restaurant network. The functionalities of the Restaurant Page includes order entry, which will update the stock of the restaurant by depletion, and restock entry, which also updates the stock by increasing it. The manager page can be used to get an overview of the supply levels at each restaurant and for sales analysis.

This food supply chain management system enables the managers to keep track of a lot of the information regarding stores on their campus besides supply management. They might use this database to help keep track of what food items are popular at each store and which items are underperforming. Another use of this database could be to tell what combinations of foods and drinks are most commonly paired together.

Data Description

Data Generation

We simulated the supply chain of the restaurants at CWRU by generating the data ourselves. We used information available online to populate the restaurant information including the menu and the items available on the menu. We will generate the ingredient data through the descriptions of each item on the menu along with some speculation based on recipe knowledge. The stock data for each restaurant was created based on research into the typical amount of stock restaurants have. We generated order data for each restaurant.

The menus of the restaurant we simulated in our database are included in the folder.

Entity Schemas and Constraints

Restaurant(<u>restaurantid</u>, restaurantName, restaurantlocation)

NOT NULL constraint on all attributes

Each store on the CWRU campus has its own tuple in the Restaurant entity, excluding the buffet restaurants Leutner and Fribley where orders are not placed. In addition to the restaurant id, which uniquely identifies each restaurant, the entity also keeps track of the name and location of the restaurant.

Menu(menuid, restaurantid)

NOT NULL constraint on all attributes

FOREIGN KEY restaurantid REFERENCES Restaurant(restaurantid)

Referential Integrity Constraint: on delete or update cascade

Each restaurant has a single menu, which is represented by a tuple in the Item entity.

Item(itemid, itemName, menuid, price)

NOT NULL constraint on all attributes

FOREIGN KEY menuid REFERENCES Menu(menuid)

Referential Integrity Constraint: on delete or update cascade

A menu is made up of items that a customer can order at a restaurant, which are represented by tuples in the Item entity. In addition to the id, the Item entity keeps track of the name of each item, the price of each item, and the menu each item is associated with.

Ingredient(ingredientid, ingredientName, expirationDate)

NOT NULL constraint on all attributes

An item is made up of ingredients that are stored at each restaurant which are represented by tuples in the Ingredient table. In addition to the id, the Ingredient table also records the name and the expiration date of each ingredient.

CustOrder(orderid, restaurantid, customerName, dateOfOrder, timeOfOrder)

NOT NULL constraint on all attributes

FOREIGN KEY restaurantid REFERENCES Restaurant(restaurantid)

Referential Integrity Constraint: on delete or update cascade

A customer makes an order at a restaurant, which is represented by a tuple in the CustOrder table. In addition to the id of each order, the table also contains the restaurant the order is placed at, the customer's name, the date of the order, and the time of the order.

Relationship Schemas with Cardinality and Constraints

Inventory(ingredientid, restaurantid, stock)

NOT NULL constraint on all attributes

FOREIGN KEY ingredientid REFERENCES Ingredient(ingredientid)

Referential Integrity Constraint: on delete or update cascade

FOREIGN KEY restaurantid REFERENCES Restaurant(restaurantid)

Referential Integrity Constraint: on delete or update cascade

Mapping Cardinalities:

Ingredient(Many, Partial Participation) -- Inventory -- Restaurant(Many, Full Participation)

Each ingredient can be stored at zero to many restaurants and a Restaurant has at least one ingredient in stock. Inventory keeps track of the ingredients stored at each restaurant. Inventory also records how much of each ingredient is in stock through the stock attribute. There are scenarios where a specific ingredient could not be stored at any restaurant if the item(s) that it is in are removed from menus. In this case, we do not want to have to remove the ingredient from the database, because it could be used in future items.

IngredientsOfItems(ingredientid, itemid)

NOT NULL constraint on all attributes
FOREIGN KEY ingredientid REFERENCES Ingredient(ingredientid)
Referential Integrity Constraint: on delete or update cascade
FOREIGN KEY itemid REFERENCES Item(itemid)
Referential Integrity Constraint: on delete or update cascade

Mapping Cardinalities:

Ingredient(Many, Partial Participation) -- IngredientsOfItem -- Item(Many, Partial Participation)

IngredientsOfItems keeps track of which ingredients are used to make each item. Each ingredient can be a part of zero to many items and each item can have zero to many ingredients. For example, bottled water is an item you can purchase at a restaurant, but is not made up of any ingredients. Also a ketchup packet is an ingredient that is stored at restaurants, but is not a part of any item that is purchasable at a restaurant.

ItemsOfCustOrder(itemid, orderid, amount)

NOT NULL constraint on all attributes
FOREIGN KEY itemid REFERENCES Item(itemid)
Referential Integrity Constraint: on delete or update no action
FOREIGN KEY orderid REFERENCES CustOrder(orderid)
Referential Integrity Constraint: on delete or update no action

Mapping Cardinalities:

Item(Many, Partial Participation) -- ItemsOfCustOrder-- CustOrder(Many, Full Participation)

ItemsOfCustOrder records the items that a customer purchases in an order. In addition, the ItemsOfCustOrder also records the number of each item that a customer orders in the attribute amount. That way if a customer purchases two of the same item in one order, there are no duplicate tuples in the table. Each item can be in zero to many orders and each order has at least one item. There is a chance that an item is never ordered by a customer, but in order for a customer to make an order at a restaurant, he or she must purchase at least one item. In this specific table, upon deletion of a tuple, no action is taken to delete the tuples referenced due to the fact that cyclical cascades occur. For example, if a customer returns an order, the ingredients of the items would not be returned, but instead thrown out

Reduction to Relational Schemas

These are the Relationship Schemas that were removed from the database due to redundancy. To capture the constraints of the ER diagram, the mapping cardinalities and participation constraints are still displayed along with the logic behind the removal.

RestaurantMenu

Restaurant(One, Full Participation) -- RestaurantMenu -- Menu(One, Full Participation)

Each menu is associated with only one restaurant and each restaurant has only one menu. Everytime a restaurant's menu changes, the menu tuple will be updated rather than adding a new menu. Given the one to one relationship between menu and restaurant with full

participation from both tables, the relationship MenuItems can be merged into either entity. For stylistic reasons such as the menu table being barren, we decided to merge MenuItems into the Menu table, adding the restaurant id and foreign key constraint to Menu.

RestaurantOrders

Restaurant(One, Partial Participation) -- RestaurantOrders -- CustOrder(Many, Full Participation)

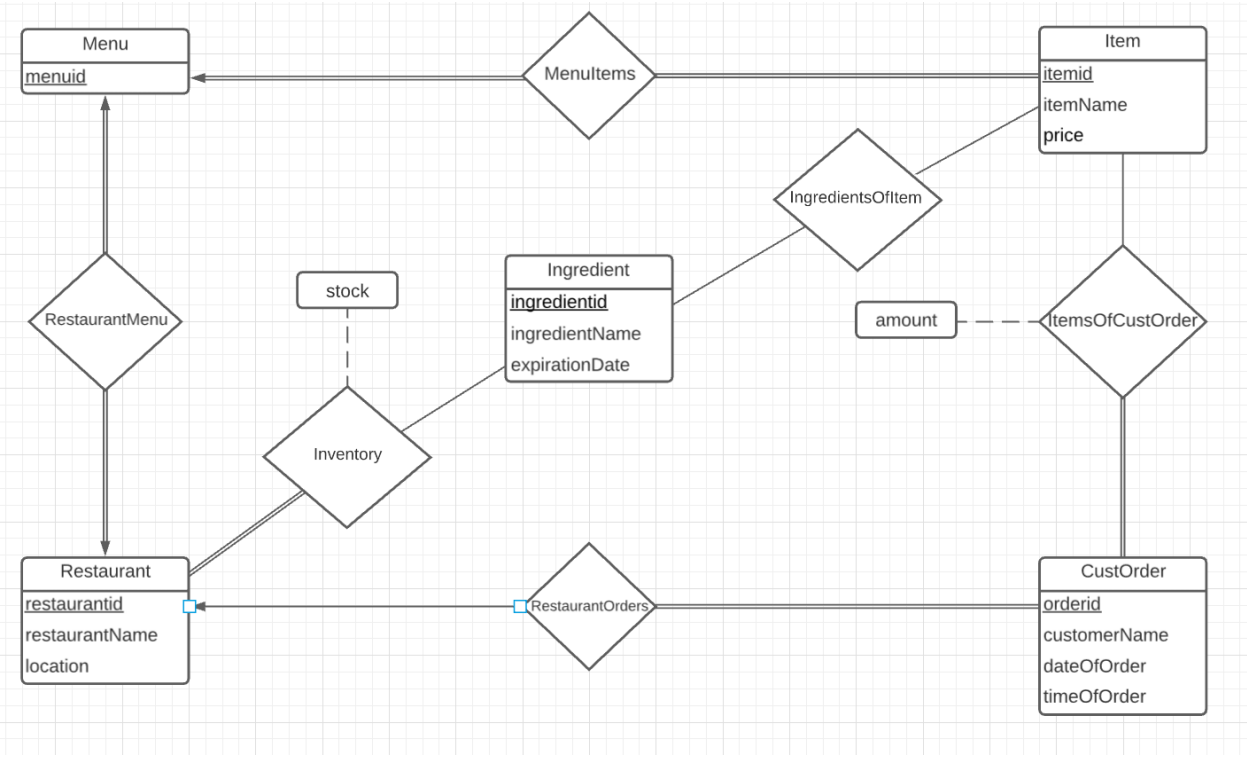
Each restaurant can have zero to many orders, while an order is placed at only one restaurant. There is a chance that no one makes an order at a restaurant. Each order cannot be placed at multiple restaurants or no restaurant. Given the many to one relationship and the full participation of CustOrder, RestaurantOrders can be merged into the CustOrder table, adding the restaurant id and foreign key constraint to CustOrder.

MenuItems

Menu(One, Full Participation) -- MenuItems -- Item(Many, Full Participation)

Each menu has at least one item and each item is associated with only one menu. We chose to have each item be associated with only one menu to help with the purpose of the database, which is keeping track of stock, so that querying for low stock items at each restaurant is easier. Given the many to one relationship and the full participation of Item, MenuItems can be merged into the Item table, with the menu id becoming an attribute of Item along with the foreign key constraint.

ER Diagram



Database Schema

Below are images of the creation of each schema in SQL. They are also in the setupRSCDB.sql file.

```
CREATE TABLE MENU
(
menuid          INT NOT NULL,
restaurantid    INT NOT NULL,
CONSTRAINT PK_MENU      PRIMARY KEY(menuid),
CONSTRAINT FK_M_REST    FOREIGN KEY( restaurantid)
REFERENCES RESTAURANT(restaurantid)
ON DELETE CASCADE ON UPDATE CASCADE
)
```

```
CREATE TABLE INGREDIENT
(
ingredientid    INT NOT NULL,
ingredientName  CHAR(50) NOT NULL,
expirationDate  DATE NOT NULL,
CONSTRAINT PK_INGREDIENT    PRIMARY KEY(ingredientid)
)
```

```
CREATE TABLE CUSTORDER
(
orderid         INT NOT NULL,
restaurantid    INT NOT NULL,
customerName    CHAR(50) NOT NULL,
dateOfOrder     DATE NOT NULL,
timeOfOrder     TIME NOT NULL,
CONSTRAINT PK_CUSTORDER PRIMARY KEY(orderid),
CONSTRAINT FK_CO_REST   FOREIGN KEY( restaurantid)
REFERENCES RESTAURANT(restaurantid)
ON DELETE CASCADE ON UPDATE CASCADE
)
```

```
CREATE TABLE INGREDIENTSOFITEM
(
ingredientid    INT NOT NULL,
itemid          INT NOT NULL,
CONSTRAINT PK_INGREDIENTSOFITEM     PRIMARY KEY(ingredientid, itemid),
CONSTRAINT FK_II_INGRED             FOREIGN KEY(ingredientid)
REFERENCES INGREDIENT(ingredientid)
ON DELETE CASCADE ON UPDATE CASCADE,
CONSTRAINT FK_II_ITEM               FOREIGN KEY( itemid)
REFERENCES ITEM(itemid)
ON DELETE CASCADE ON UPDATE CASCADE
)
```

```
CREATE TABLE RESTAURANT
(
restaurantid        INT NOT NULL,
restaurantName      CHAR(50) NOT NULL,
restaurantlocation  CHAR(50) NOT NULL,
CONSTRAINT PK_RESTAURANT        PRIMARY KEY (restaurantid)
```

```
CREATE TABLE ITEMSOFCUSTORDER
(
itemid          INT NOT NULL,
orderid         INT NOT NULL,
amount          INT NOT NULL,
CONSTRAINT PK_ITEMSOFCUSTORDER  PRIMARY KEY(itemid, orderid),
CONSTRAINT FK_IC_ITEM           FOREIGN KEY(itemid)
REFERENCES ITEM(itemid)
ON DELETE NO ACTION ON UPDATE NO ACTION,
CONSTRAINT FK_IC_CUST           FOREIGN KEY(orderid)
REFERENCES CUSTORDER(orderid)
ON DELETE NO ACTION ON UPDATE NO ACTION
)
```

```
CREATE TABLE INVENTORY
(
ingredientid    INT NOT NULL,
restaurantid    INT NOT NULL,
stock           INT NOT NULL,
CONSTRAINT PK_INVENTORY     PRIMARY KEY(ingredientid, restaurantid),
CONSTRAINT FK_I_INGRED      FOREIGN KEY(ingredientid)
REFERENCES INGREDIENT(ingredientid)
ON DELETE CASCADE ON UPDATE CASCADE,
CONSTRAINT FK_I_REST        FOREIGN KEY( restaurantid)
REFERENCES RESTAURANT(restaurantid)
ON DELETE CASCADE ON UPDATE CASCADE
```

```
CREATE TABLE ITEM
(
itemid          INT NOT NULL,
itemName        CHAR(50) NOT NULL,
menuid          INT NOT NULL,
price           FLOAT NOT NULL,
CONSTRAINT PK_ITEM      PRIMARY KEY( itemid),
CONSTRAINT FK_I_MENU    FOREIGN KEY( menuid)
REFERENCES MENU (menuid)
ON DELETE CASCADE ON UPDATE CASCADE
)
```

Satisfaction of Normal Forms

Tables in 3NF

Restaurant
Functional Dependencies:

restaurantid → restaurantName, restaurantlocation

restaurantName → restaurantlocation

Superkeys:

restaurantid, restaurantName, restaurantlocation
restaurantid, restaurantName
restaurantid

The first FD satisfies BCNF as restaurantid is a superkey of Restaurant, but the second is only in 3NF as the attribute restaurantName is only a part of a superkey of R, but not one on its own.

Item
Functional Dependencies:

itemid → itemName, menuid, price

itemName, menuid → price

Superkeys:

itemid, itemName, menuid, price
itemid, itemName, menuid
itemid

The first FD satisfies BCNF as itemid is a superkey of Item. The second FD satisfies 3NF as the attributes (itemName, menuid) do not make up a superkey on their own, but are a part of a superkey.

CustOrder
Functional Dependencies

orderid → restaurantid, customerName, dateOfOrder, timeOfOrder

restaurantid, dateOfOrder, timeOfOrder → customerName

Superkeys:

orderid, restaurantid, customerName, dateOfOrder, timeOfOrder
orderid, restaurantid, dateOfOrder, timeOfOrder
orderid

The first FD satisfies BCNF as the attribute orderid is a superkey, but the second is only in 3NF as the group of attributes (restaurantid, dateOfOrder, timeOfOrder) is only a part of a superkey.

Tables in BCNF

Menu

Functional Dependencies:

menuid → restaurantid

restaurantid → menuid

Superkeys:

menuid, restaurantid
menuid
restaurantid

Both menuid and restaurantid are superkeys on their own. Due to the one to one relationship between Menu and Restaurant, they both can uniquely identify tuples in the Menu table. So the first FD satisfies BCNF as menuid is a superkey and the second satisfies BCNF as the attribute restaurantid is a superkey.

Ingredient

Functional Dependencies

ingredientid → ingredientName, expirationDate

Superkeys:

ingredientid, ingredientName, expirationDate
ingredientid

The FD of Ingredient satisfies BCNF as ingredient id is a superkey of the Ingredient table, therefore the table is in BCNF.

Inventory

Functional Dependencies:

ingredientid, restaurantid → stock

SuperKeys:

ingredientid, restaurantid, stock
ingredientid, restaurantid

The FD of Inventory satisfies BCNF as the attributes (ingredientid, restaurantid) are a superkey of Inventory, so the table is in BCNF.

IngredientsOfItems

Functional Dependencies:

None

Superkeys:

ingredientid, itemid

There are no functional dependencies of IngredientsOfItems because there are no non-keyed attributes. A table with no FDs is in BCNF.

ItemsOfCustOrder

Functional dependencies:

itemid,orderid → amount

Superkeys:

itemid, orderid, amount

itemid, orderid

The FD is in BCNF because the attributes on the left (itemid, orderid) are a superkey of ItemsOfCustOrder, making the table satisfy BCNF.

Example Queries

Below are some example queries that are used in our database implementation to help with supply chain management and also sales analysis. They are ordered by degree of complexity.

Sample Query One

Get the names of all Items that cost more than $7 at the Den

SQL

```sql
USE FoodManagement

SELECT itemName, price
FROM ITEM, MENU, RESTAURANT
WHERE ITEM.menuid = MENU.menuid AND MENU.restaurantid = RESTAURANT.restaurantid
AND RESTAURANT.restaurantName = 'The Den'
AND ITEM.price > 7
```

Output

	itemName	price
1	French Toast Slam …	7.09
2	The Double Den Burger …	7.99
3	Bacon Cheese Burger …	7.99
4	Chipotle Bacon Cheese Bur…	7.99
5	Bacon Avocado Club Burger…	7.99
6	Breakfast Scramble Burger…	7.99
7	Moons Over My Hammy Burge…	7.99
8	Cajun Chicken Wrap …	7.29
9	Southwestern Chicken Sala…	7.59
10	Crispy Chicken Salad …	7.59

Relational Algebra

$$\Pi_{itemName,price}\left(\sigma_{restaurantName="The\ Den"\wedge price>7}\ (Item \bowtie Menu \bowtie Restaurant)\right)$$

Tuple Relational Calculus

$$\{t|(\exists s)(s \in Item(t[itemName] = s[itemName] \wedge t[price] = s[price] \wedge s[price] > 7 \\ \wedge (\exists u)(u \in Menu \wedge u[menuid] = s[menuid] \\ \wedge (\exists v)(v \in Restaurant \wedge v[restaurantid] = u[restaurantid] \wedge v[restaurantName = "The\ Den"])))\}$$

Sample Query Two

GetI the ingredients at Rough Rider Pizza that have a stock of less than 50.

SQL

```sql
USE FoodManagement

SELECT ingredientName, stock
FROM INGREDIENT, INVENTORY, RESTAURANT
WHERE INGREDIENT.ingredientid = INVENTORY.ingredientid
AND INVENTORY.restaurantid = RESTAURANT.restaurantid
AND RESTAURANT.restaurantName = 'Rough Rider Pizza'
AND INVENTORY.stock < 50
```

Output

	ingredientName	stock
1	Chicken …	42
2	Pepsi …	12
3	Coffee …	47
4	Bagel …	23

Relational Algebra

$$\Pi_{ingredientName,stock}\left(\sigma_{restaurantName="Rough\ Rider\ Pizza"\wedge\ stock<50}\ (Ingredient \bowtie Inventory \bowtie Restaurant)\right)$$

Tuple Relational Calculus

$$\{t|(\exists s)(s \in Ingredient(t[ingredientName] = s[ingredientName] \wedge (\exists u)(u \in Inventory \wedge u[ingredientid] = s[ingredientid] \wedge u[stock] = t[stock] \wedge u[stock] < 50 \wedge (\exists v)(v \in Restaurant \wedge v[restaurantid] = u[restaurantid] \wedge v[restaurantName] = "Rough\ Rider\ Pizza"))))\}$$

Sample Query Three

Find the most ordered item from The Den

SQL

```
USE FoodManagement

SELECT ITEMSOFCUSTORDER.itemid, COUNT(ITEMSOFCUSTORDER.itemid) as numTimesOrdered
INTO #temp1
FROM ITEMSOFCUSTORDER, CUSTORDER, RESTAURANT
WHERE CUSTORDER.restaurantid = RESTAURANT.restaurantid
AND CUSTORDER.orderid = ITEMSOFCUSTORDER.orderid
AND RESTAURANT.restaurantName = 'The Den'
GROUP BY ITEMSOFCUSTORDER.itemid

SELECT itemName
FROM ITEM
WHERE ITEM.itemid =
(SELECT #temp1.itemid
FROM #temp1
WHERE numTimesOrdered = (SELECT MAX(numTimesOrdered) FROM #temp1))
```

Output

	itemName
1	The Lil Den Burger …

Relational Algebra

$$temp1 \leftarrow {}_{itemid}\mathcal{G}_{count(itemid)\ as\ numTimesOrdered}\left(\sigma_{restaurantName="The\ Den"}\ (ItemsOfCustorder \bowtie Custorder \bowtie Restaurant)\right)$$

$$temp2 \leftarrow \mathcal{G}_{max(numTimesOrdered)}\ (temp1)$$

$$\Pi_{itemName}\left(\sigma_{temp2.itemid=item.id}(temp2 \bowtie item)\right)$$

Tuple Relational Calculus

- As there is no group by function in Tuple Relational Calculus, assume there exists a temporary table “temp” in our database that contains the number of times an item has been ordered(numTimesOrdered) from the Den, along with the itemid

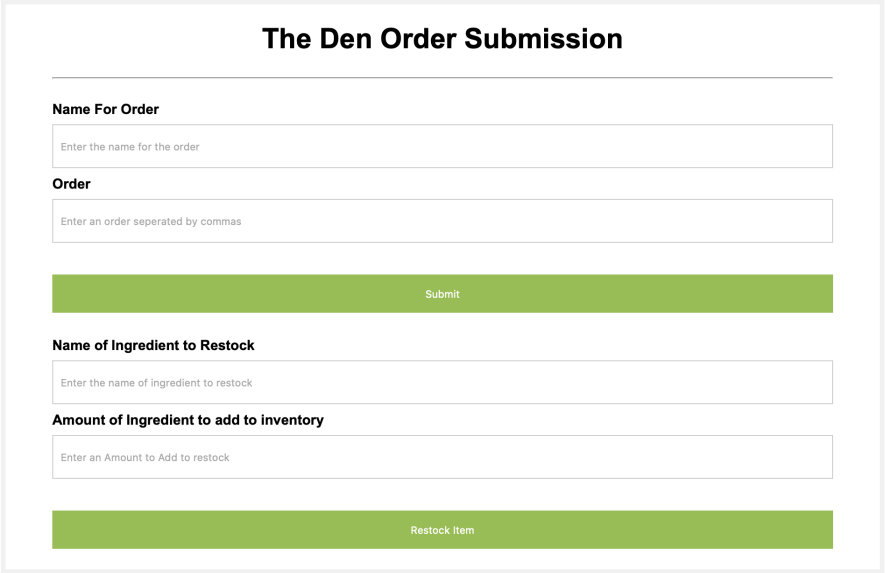
$$\{t|(\exists s)(s \in Item(t[itemName] = s[itemName] \\ \wedge (\exists u)(u \in temp \wedge u[itemid] = s[itemid] \\ \wedge \neg(\exists v)(v \in temp(v.numTimesOrdered > u.numTimesOrdered)))\}$$

Implementation

Starting at the front end, we used HTML, CSS and JavaScript to create our simple website. This website is for the use case of an employer at the Den submitting an order of a customer into our database or a manager checking to see if any restaurant is running low on stock. The user has two options when looking at our website, to submit an order or to restock the ingredients in our database. Both options update the “stock” of ingredients within our database. For the manager page, the user can select which restaurant to view the stock of We used PHP to connect the front end to the back end database. Currently our web application and our database run locally. To help with version control we used github

Web Application Code/Screenshots

Here is a screenshot of the webpage for the Den order submission form and ingredient restock page



Order Submission Functionality

Here is the Inventory table for 'The lil Den Burger' before and after the submission of the order. As you can see the stock attribute is one less for each of the ingredients within the items ordered.

	ingredientName		stock
1	Hamburger buns	…	19
2	Ground beed patty	…	19
3	Lettuce	…	19
4	Lettuce	…	19
5	Tomato	…	19
6	Pickles	…	19
7	Onions	…	19
8	American cheese	…	18
9	American cheese	…	18

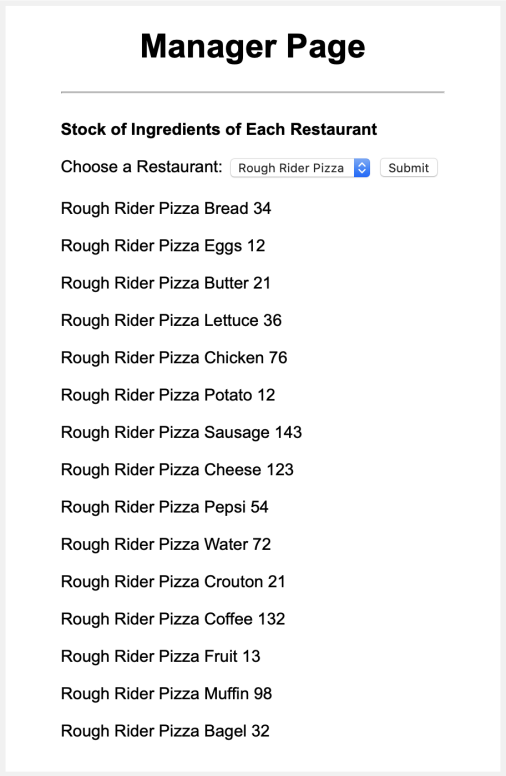
	ingredientName		stock
1	Hamburger buns	…	18
2	Ground beed patty	…	18
3	Lettuce	…	18
4	Lettuce	…	18
5	Tomato	…	18
6	Pickles	…	18
7	Onions	…	18
8	American cheese	…	17
9	American cheese	…	17

Ingredient Restock Functionality

Here is the Inventory table before and after the stock of 'Pickles' get increased by 5

	ingredientName	stock
1	Pickles …	18

	ingredientName	stock
1	Pickles …	23



Manager Page Functionality

On the right is an example output of the Manager Page looking at stock for Rough Rider, a manager can select the restaurant that they would like to see stock off and submit their request and the page will generate with

Work Breakdown

David

- Wrote SQL table creation script
- Worked on ER Diagram
- Worked on Reduction to Relational Schemas
- Worked on Functional Dependencies and Normalization

James

- Worked on ER Diagram
- Worked on Reduction to Relational Schemas
- Wrote SQL data generation script
- Worked with Mike on SQL queries
- Wrote RA and TRC equivalents of SQL queries

Michael

- Worked on ER Diagram
- Worked on Reduction to Relational Schemas
- Wrote PHP and HTML code of web application
- Worked with James on SQL queries

What We've Learned

We've learned what it takes to create a web based application with a database implementation from scratch. We learned how to take an idea and use the database design rules we learned in class to create our initial ER diagram, keeping in mind the specifications of our idea. Then from

our original design, we had to apply dependency theory and revisit database design rules to revise our ER diagram and relational schemas. We iterated this revision process as we implemented our database into SQL, learning about the nuances of topics covered in class along the way, such as a table with no functional dependencies is possible and is in BCNF. We struggled to set up our ER diagram in such a way that captured the goals of our idea and reduced redundancy. In particular, edits such as adding a relationship between ingredients and restaurants to capture the stock aspect of our implementation, and adding an amount attribute to prevent redundancy within the ItemsOfCustOrder were particularly important breakthroughs. Implementing our queries, particularly the third query, was incredibly useful in developing our SQL skills. We learned how useful creating temporary tables are. It allowed us to circumvent the issue of trying to select an attribute that was not in the GROUP BY function. Additionally we have learned how to effectively connect a database to a web application using php as the glue between the front end and the backend. The php deemed incredibly difficult to complete as we realized that the front end would need additional work to function properly.