CMPS 3420

Sequoia Sandwich Company Database

Zakary Worman

Group 06

Due: 20th May, 2019

Table of Contents

Phase 1: Info Gathering and E-R Modeling

1 Data/Information Collection, Fact Finding, Conceptual Database Design

In order for us to successfully implement a database of Sequoia Sandwich Company, We must first design the database. This section will focus on the set-up and needs of the company. The methods of fact gathering will be summarized and displayed clearly in an E-R Diagram using UML style.

1.1 Fact-Finding/Data Collection

The first step to creating a database for any business is to first know what needs to be included. This will be described in the following section as well as how these ideas were discovered. It will include all the relevant data of the business and its dealings. To efficiently create a database for our business, we must create models to represent our information we will be storing into the database. This will allow us to accurately organize and manage our data in the most efficient and optimal way possible without needless fixing and reprogramming of the database itself.

This report will include multiple phases in which we will design and implement our database. There will be five phases, beginning with an introduction to the store and description of what the database with be about, and ending with the implementation of the database itself and the user interface. Each phase will include its own section, with its own description and concepts. All information from each phase will be included in its respective section.

1.1.1 Introduction to Enterprise/Organization:

The business we chose is Sequoia Sandwich Company. The company acts as a restaurant that specializes in using fresh produce to make sandwiches, soup, and desert. They also sell homemade pastries and a few preprocessed items (i.e. soda and chips). As per their website, “At The Sequoia Sandwich Company we are committed to using only premium quality ingredients and hand-crafting each sandwich. Our promise is to serve only the best food available. That means using premium deli meats and cheeses exclusively for our sandwiches.Our delicatessen products have been made the old fashioned way, with no artificial colors, flavors or fillers.”

Sequoia Sandwich Company employs workers at multiple locations, with some of them being promoted to supervisors and the rest holding their preferred position. All items from the menu are made in-house and ordered fresh from the supplier “Cisco”. Ingredients such as lettuce, tomatoes, meat, mustard, and so on are used to create menu items and deliver them to the customers with the primary goal of serving fresh food in a smaller store dynamic.

Sequoia Sandwich company started as a small time sandwich business within Bakersfield. The original owners were Gary Blackburn and Jeff Simpson. Tired of their everyday jobs, they quit, in search of a new venture in which they could become successful in life. The store specializes in being a mixture between a fast food restaurant and a dine-in restaurant. In 2002, the company opened a second store in Rosedale in Bakersfield, followed by a third in 2005 on Ming Avenue. In 2008 Sequoia Sandwich Company successfully opened its fourth store in Clovis. Recently a new store opened in San Luis Obispo with another opening planned in the near future for Buena Vista Road on the Southwest side of town.

1.1.2 Description Fact-Finding Techniques:

Zachary Kaiser is an employee at Sequoia Sandwich Company. We also conducted a quick interview with the store manager that Zachary Kaiser works for. During our fact gathering phase, the company ownership was switched to a new group of individuals. This means that there may be some new changes to how the company will run and is managed. In the case that changes are made, we will construct a new E-R Diagram to reflect the operational relationships and entities that need to be altered. However, until all these changes take place, it is impossible to determine what these changes will be, or if there will be changes at all.

The best method for finding information about the store location is asking the manager, as they oversee most if not all of a location's business. There is always more detail that can be added to a database, especially when the business is of a larger size. In the case of Sequoia, as it is a smaller business, obtaining and organizing the amount of information is a bit easier, but there must be a great amount of detail given to the database to ensure that the organization is as efficient as possible.

1.1.3 Section of Enterprise:

We will be designing our E-R Diagram for the city section of the enterprise. Meaning we will be including each restaurant and its location. These restaurants will include everything that they need to operate. We will not be doing anything above city level, meaning that store locations outside of Bakersfield will not be included, and ownership will not be included.

There are a total of three store locations in Bakersfield. There is a location in Rosedale, Southwest Bakersfield, and Downtown Bakersfield. Each store location acts independently of the others, but contain the same menu items and so on. For example, the Sequoia on Rosedale stays open until 8:00pm, but the store located in the Southwest area of Bakersfield closes at 3:00pm. Each store location is also run by its own manager and maintains its own staff and supply orders.

1.1.4 Itemized Descriptions of Entity Sets and Relationship Sets:

The customer makes an order, which is created by an employee who works for a store. A customer comes in and makes an order to an employee, who then rings the order up, sending it to the front and back employees who, depending on what the menu

items are, will cook or make the food that the customer will then receive. Employees are capable of working more than one role, but not more than one role in a day. They are capable or ringing up orders, cooking food, cleaning dishes, and supervising employees.

Menu items ordered contain the identification number, name, and price. These menu items are made from produce, which contains the type, quantity, date acquired, expiration date, and weight. The produce is ordered from a supplier, specifically “Cisco”, who then delivers the produce by order from the store. Nutritional facts of each produce item are displayed in Nutrition, which contains all contents needed for a nutritional label, including serving size, carbs, calories, and so on.

The store must create orders to send out to the supplier, who then brings in a shipment of supplies to the store. The order contains an identification number, a quantity of supplies requested, the weight, and the price. The supplier contains a name and product, while the store contains a number, location, and a manager.

1.1.5 User Groups, Data Views and Operations:

For user groups, it consists of an entire group of individuals accessing the database in a similar way and possibly through different data. Customers are a user group, as they are the group that comes in unrelated to the store and make an order to the employee. Customers are an entity that accesses the database in a way that gives information to the company regarding payment, and usage of produce and supplies.

The employees are an entire user group, and they all work inside of a specific store location. Employees access the database through the creation of orders and manage the amount of money received and given to the company. The creation of the orders will help keep track of the amount of produce.

1.2 Conceptual Database Design

This section will focus on the parts of the E-R Diagram. It first describes each entity in detail, followed by their attributes. It will also describe how they are related in ways such as cardinality and participation.

1.2.1 Entity Type Description:

Employee: The employee entity is meant to maintain a record of all people who work in at Sequoia. It contains attributes such as their name, ID, address, salary, and so on. This allows a more efficient way to access the information about an employee or employees.

- Attributes:
    - Employee_ID: Identification number of the employee
        - Domain/type: Integer
        - Value-range: [≥ 000]
        - Default value: 000
        - Null Allowed: No
        - Unique: Yes

        - Single or Multi Value: Single
        - Simple or Composite: Simple
    - Address: Location that the employee lives
        - Domain/type: Integer, Char String, Char String, Char String, Integer
        - Value-range: [> 0, > 3 chars, > 3 chars, > 3 chars, > 0]
        - Default value: [Null]
        - Null Allowed: Yes
        - Unique: No
        - Single or Multi Value: Multi
        - Simple or Composite:
    - Fname: Employee's legal first name
        - Domain/type: Char string
        - Value-range: [≥ 3 chars]
        - Default value: John
        - Null Allowed: No
        - Unique: No
        - Single or Multi Value: Single
        - Simple or Composite: Simple
    - Minit: Employee's middle initial.
        - Domain/type: Char String
        - Value-range: [≥ 3 chars]
        - Default value: James
        - Null Allowed: Yes
        - Unique: No
        - Single or Multi Value: Single Value
        - Simple or Composite: Simple
    - Lname: Employee's last name
        - Domain/type: Char string
        - Value-range: [≥ 3 chars]
        - Default value: Doe
        - Null Allowed: No
        - Unique: No
        - Single or Multi Value: Single
        - Simple or Composite: Simple
    - DOB: Date of Birth of employee
        - Domain/type: Integer
        - Value-range: [1-12, 1-31, 1900-2004]
        - Default value: [1/1/1990]
        - Null Allowed: No
        - Unique: No
        - Single or Multi Value: Single
        - Simple or Composite: Composite
    - SSN: Employee's social security number
        - Domain/type: Integer
        - Value-range: [000-999, 00-99, 0000-9999]
        - Default value: 000-00-0000
        - Null Allowed: No

        - Unique: Yes
        - Single or Multi Value: Single
        - Simple or Composite: Composite
    - Start_Date: The exact day, year, and month that the employee was hired
        - Domain/type: Integer
        - Value-range: [1-12, 1-31, 1900-2019]
        - Default value: 1/1/2019
        - Null Allowed: No
        - Unique: No
        - Single or Multi Value: Single
        - Simple or Composite: Composite
- Candidate Keys: Employee ID, SSN
- Primary Keys: Employee ID
- Strong/Weak Entity: Strong
- Fields to be indexed: Employee ID, Name, SSN

Janitor: A position that includes cleaning up by means of sweeping, mopping, doing dishes, etc

- Attributes:
    - Salary: How much the person working this position makes hourly
        - Domain/type: Integer
        - Value-Range: [≥ minimum wage]
        - Default Value: [minimum wage]
        - Null Allowed: No
        - Unique: No
        - Single or Multi Value: Single
        - Simple or Composite: Simple
- Candidate Keys: Employee ID
- Primary Keys: Employee ID
- Strong/Weak Entity: Weak
- Fields to be indexed: Employee ID

Salad Prep: A position that has the sole job of making the salads to be sold

- Attributes:
    - Salary: How much the person working this position makes hourly
        - Domain/type: Integer
        - Value-Range: [≥ minimum wage]
        - Default Value: [minimum wage]
        - Null Allowed: No
        - Unique: No
        - Single or Multi Value: Single
        - Simple or Composite: Simple
- Candidate Keys: Employee ID
- Primary Keys: Employee ID
- Strong/Weak Entity: Weak
- Fields to be indexed: Employee ID

Griller: A position that has the sole job of cooking the meats and vegetables to be used on the sandwiches

- Attributes:
    - Salary: How much the person working this position makes hourly
        - Domain/type: Integer
        - Value-Range: [≥ minimum wage]
        - Default Value: [minimum wage]
        - Null Allowed: No
        - Unique: No
        - Single or Multi Value: Single
        - Simple or Composite: Simple
- Candidate Keys: Employee ID
- Primary Keys: Employee ID
- Strong/Weak Entity: Weak
- Fields to be indexed: Employee ID

Cold Prep: A position that has the sole job of putting the sandwiches together once all the meat, vegetables, condiments, etc have been brought forth

- Attributes:
    - Salary: How much the person working this position makes hourly
        - Domain/type: Integer
        - Value-Range: [≥ minimum wage]
        - Default Value: [minimum wage]
        - Null Allowed: No
        - Unique: No
        - Single or Multi Value: Single
        - Simple or Composite: Simple
- Candidate Keys: Employee ID
- Primary Keys: Employee ID
- Strong/Weak Entity: Weak
- Fields to be indexed: Employee ID

Support: Any position that includes jobs that can be done with little time. These include cutting vegetables, refilling soda machines, cleaning tables, etc. Essentially, any role that assists the rest of the roles in performing their primary job.

- Attributes:
    - Salary: How much the person working this position makes hourly
        - Domain/type: Integer
        - Value-Range: [≥ minimum wage]
        - Default Value: [minimum wage]
        - Null Allowed: No
        - Unique: No
        - Single or Multi Value: Single
        - Simple or Composite: Simple

- Candidate Keys: Employee ID
- Primary Keys: Employee ID
- Strong/Weak Entity: Weak
- Fields to be indexed: Employee ID

Cashier: The person who takes the orders from customers

- Attributes:
    - Salary: How much the person working this position makes hourly
        - Domain/type: Integer
        - Value-Range: [≥ minimum wage]
        - Default Value: [minimum wage]
        - Null Allowed: No
        - Unique: No
        - Single or Multi Value: Single
        - Simple or Composite: Simple
- Candidate Keys: Employee ID
- Primary Keys: Employee ID
- Strong/Weak Entity: Weak
- Fields to be indexed: Employee ID

Expediter: The person in charge of taking phone orders

- Attributes:
    - Salary: How much the person working this position makes hourly
        - Domain/type: Integer
        - Value-Range: [≥ minimum wage]
        - Default Value: [minimum wage]
        - Null Allowed: No
        - Unique: No
        - Single or Multi Value: Single
        - Simple or Composite: Simple
- Candidate Keys: Employee ID
- Primary Keys: Employee ID
- Strong/Weak Entity: Weak
- Fields to be indexed: Employee ID

Supervisor: Meant to aid the Manager in keeping everything flowing smoothly and correctly in the restaurant

- Attributes:
    - Salary: How much the person working this position makes hourly
        - Domain/type: Integer
        - Value-Range: [≥ minimum wage]
        - Default Value: [minimum wage]
        - Null Allowed: No
        - Unique: No
        - Single or Multi Value: Single

        - Simple or Composite: Simple
    - Type: The current type of supervision that the employee is performing
        - Domain/type: Char String
        - Value-Range: [≥ 3 Char]
        - Default Value: [Morning]
        - Null Allowed: No
        - Unique: No
        - Single or Multi Value: Single
        - Simple or Composite: Simple
- Candidate Keys: Employee ID
- Primary Keys: Employee ID
- Strong/Weak Entity: Weak
- Fields to be indexed: Employee ID

Manager: The person largely in charge of running most of the day to day operations and keeping the store on task. They would also perform tasks such as making the produce orders from the suppliers for the store and hiring/firing employees.

- Attributes:
    - Salary: How much the person working this position makes hourly
        - Domain/type: Integer
        - Value-Range: [≥ minimum wage]
        - Default Value: [minimum wage]
        - Null Allowed: No
        - Unique: No
        - Single or Multi Value: Single
        - Simple or Composite: Simple
    - Type: The current type of management that the employee is performing
        - Domain/type: Char String
        - Value-Range: [≥ 3 Char]
        - Default Value: [General]
        - Null Allowed: No
        - Unique: No
        - Single or Multi Value: Single
        - Simple or Composite: Simple
- Candidate Keys: Employee ID
- Primary Keys: Employee ID
- Strong/Weak Entity: Weak
- Fields to be indexed: Employee ID

Store: One of the five locations that serves sandwiches.

- Attributes:
    - Number: Store number of the Sequoia location
        - Domain/type: Integer
        - Value-range: [> 1]
        - Default value: 1
        - Null Allowed: No

        - Unique: Yes
        - Single or Multi Value: Single
        - Simple or Composite: Simple
    - Location: Street address of the store
        - Domain/type: Char Strings
        - Value-range: [> 1 char]
        - Default value: [Current Store]
        - Null Allowed: No
        - Unique: Yes
        - Single or Multi Value: Multi
        - Simple or Composite: Composite
    - Manager: Manager of the store location
        - Domain/type: Char String
        - Value-range: [> 3 char]
        - Default value: [John Doe]
        - Null Allowed: No
        - Unique: Yes
        - Single or Multi Value: Multi
        - Simple or Composite: Composite
- Candidate Keys: Number, Location
- Primary Keys: Number
- Strong/Weak Entity: Strong
- Fields to be indexed: Number, Location

Clock In: One of the five locations that serves sandwiches.

- Attributes:
    - Date: The day, month, and year that the person clocked in
        - Domain/type: Integer
        - Value-range: [1-12, 1-31, 1900-2019]
        - Default value: [Today's Date]
        - Null Allowed: No
        - Unique: No
        - Single or Multi Value: Single
        - Simple or Composite: Composite
    - Time_In: The exact time that the Employee clocked in
        - Domain/type: Integer
        - Value-range: [0-24:0-59]
        - Default value: [Current Time]
        - Null Allowed: No
        - Unique: No
        - Single or Multi Value: Single
        - Simple or Composite: Composite
    - Time_Out: The exact time that the Employee Clocked out
        - Domain/type: Integer
        - Value-range: [0-24:0-59]
        - Default value: [Current Time]

        - Null Allowed: No
        - Unique: No
        - Single or Multi Value: Single
        - Simple or Composite: Composite
    - Earnings: The total earnings of the employee for this clock in period
        - Domain/type: Integer
        - Value-range: [> 0]
        - Default value: [Salary*(Time_In - Time_Out)]
        - Null Allowed: No
        - Unique: No
        - Single or Multi Value: Single
        - Simple or Composite: Single
- Candidate Keys: Date, Earnings
- Primary Keys: Employee_ID
- Strong/Weak Entity: Weak
- Fields to be indexed: Date, Time_In, Time_Out

Customer: Person who makes an order and purchases menu items at one of the store locations. Employees will work closely with customers to create orders that contain menu items. Customers are the entity that make the order, while the employee will create it. The customer entity allows access to records and details about them..

- Attributes:
    - Customer_number: Order number given to customer
        - Domain/type: Integer
        - Value-range: [0-99]
        - Default value: [1]
        - Null Allowed: No
        - Unique: Yes
        - Single or Multi Value: Single
        - Simple or Composite: Simple
    - Phone_number: Phone number of customer
        - Domain/type: Integer
        - Value-range: [(000)000-0000 to (999)999-9999]
        - Default value: [111-111-1111]
        - Null Allowed: Yes
        - Unique: Yes
        - Single or Multi Value: Single
        - Simple or Composite: Simple
    - Name: First Name and Last Initial of customer
        - Domain/type: Char Strings
        - Value-range: [> 3 chars]
        - Default value: [John Doe]
        - Null Allowed: No
        - Unique: No
        - Single or Multi Value: Single
        - Simple or Composite: Composite

- Candidate Keys: Customer_number, name
- Primary Keys: Customer_number
- Strong/Weak Entity: Strong
- Fields to be indexed: Customer_number, Phone_number, name

Order: This entity is the description of the order, including the receipt identification and the date, The order entity allows access to information involving orders made by customers. The cost of items sold are saved at the price during the time of purchase, so price alterations to menu items in the future will not change the value of the saved order.

- Attributes:
    - ID: Receipt identification number
        - Domain/type: Integer
        - Value-range: [> 0]
        - Default value: [1]
        - Null Allowed: No
        - Unique: Yes
        - Single or Multi Value: Single
        - Simple or Composite: Simple
    - Date: Date of purchase
        - Domain/type: Integer
        - Value-range: [0-31, 0-12, 1900-2019]
        - Default value: [Today's Date]
        - Null Allowed: No
        - Unique: Yes
        - Single or Multi Value: Single
        - Simple or Composite: Composite
- Candidate Keys: ID
- Primary Keys: ID
- Strong/Weak Entity: Weak
- Fields to be indexed: ID, Date

Produce: The produce entity is the description of ingredients contained by the store. Produce describes the type, quantity, price, expiration date, and acquired date of the product. This amount of produce purchased and their expiration dates are saved to keep track of incoming and outgoing resources.

- Attributes:
    - Type: Name of produce
        - Domain/type: Char String
        - Value-range: [> 3 char]
        - Default value: [name of type]
        - Null Allowed: No
        - Unique: Yes
        - Single or Multi Value: Single
        - Simple or Composite: Simple
    - Quantity: Amount of produce held
        - Domain/type: integer

        - Value-range: [≥ 0]
        - Default value: [0]
        - Null Allowed: Yes
        - Unique: No
        - Single or Multi Value: Single
        - Simple or Composite: Simple
    - Aquired_date: Date produce was acquired
        - Domain/type: Integer
        - Value-range: [0-31, 0-12, 1900-2019]
        - Default value: [Today's date]
        - Null Allowed: No
        - Unique: No
        - Single or Multi Value: Single
        - Simple or Composite: Composite
    - Expiration_date: Day produce will expire and be thrown out
        - Domain/type: Integers
        - Value-range: [0-31, 0-12, 2019]
        - Default value: [1 week after Aquired_date]
        - Null Allowed: No
        - Unique: No
        - Single or Multi Value: Single
        - Simple or Composite: Composite
    - Cost: Cost of specified produce type
        - Domain/type: Integer
        - Value-range: [> 0]
        - Default value: [1]
        - Null Allowed: No
        - Unique: Yes
        - Single or Multi Value: Single
        - Simple or Composite: Simple
- Candidate Keys: Type, Quantity
- Primary Keys: Type
- Strong/Weak Entity: Strong
- Fields to be indexed: Type, Quantity, Expiration_date

Supplier: The supplier of the inventory at Sequoia Sandwich Company store locations. The supplier delivers produce to each store location. Supplier contains the information regarding the product sold to each Sequoia location and allows for the tracking of inventory delivered.

- Attributes:
    - Name: Company Name of the supplier.
        - Domain/type: Char String
        - Value-range: [> 3 char]
        - Default value: [Company]
        - Null Allowed: No
        - Unique: No

        - Single or Multi Value: Single
        - Simple or Composite: Simple
    - Product: Name of product sold to Sequoia.
        - Domain/type: Char String
        - Value-range: [> 3 char]
        - Default value: [Product]
        - Null Allowed: No
        - Unique: No
        - Single or Multi Value: Multi
        - Simple or Composite: Simple
- Candidate Keys: Name, Product
- Primary Keys: Name
- Strong/Weak Entity: Strong
- Fields to be indexed: Name, Product

Nutrition: The nutrition is the general information regarding nutritional value of the menu items. This entity describes the many different values of nutrition on the nutrition labels of items. Nutrition allows for access to information regarding nutritional values of items.

- Attributes:
    - Serving_size: Portion of food for 1 serving size
        - Domain/type: Integer
        - Value-range: [> 0]
        - Default value: [1]
        - Null Allowed: No
        - Unique: No
        - Single or Multi Value: Single
        - Simple or Composite: Simple
    - Calories: Amount of calories per serving size
        - Domain/type: Integer
        - Value-range: [≥ 0]
        - Default value: [0]
        - Null Allowed: No
        - Unique: No
        - Single or Multi Value: Single
        - Simple or Composite: Simple
    - Fat: Substances that allow for body to store energy and keep the skin healthy.
        - Total: Total amount of fat in a serving size
            - Domain/type: Integer
            - Value-range: [≥ 0]
            - Default value: [0]
            - Null Allowed: Yes
            - Unique: No
            - Single or Multi Value: Single
            - Simple or Composite: Simple
        - Saturated: Saturated fat in a serving size

                - Domain/type: Integer
                - Value-range: [≥ 0]
                - Default value: [0]
                - Null Allowed: Yes
                - Unique: No
                - Single or Multi Value: Single
                - Simple or Composite: Simple
            - Trans: Trans fat contained in a serving size
                - Domain/type: Integer
                - Value-range: [≥ 0]
                - Default value: [0]
                - Null Allowed: Yes
                - Unique: No
                - Single or Multi Value: Single
                - Simple or Composite: Simple
    - Cholesterol: Total cholesterol in a serving size
        - Domain/type: Integer
        - Value-range: [≥ 0]
        - Default value: [0]
        - Null Allowed: Yes
        - Unique: No
        - Single or Multi Value: Single
        - Simple or Composite: Simple
    - Sodium: Amount of sodium in a serving size
        - Domain/type: Integer
        - Value-range: [≥ 0]
        - Default value: [0]
        - Null Allowed: Yes
        - Unique: No
        - Single or Multi Value: Single
        - Simple or Composite: Simple
    - Carbohydrates: Sugars, starches and fibers
        - Total: Total carbohydrates in a serving size
            - Domain/type: Integer
            - Value-range: [≥ 0]
            - Default value: [0]
            - Null Allowed: Yes
            - Unique: No
            - Single or Multi Value: Single
            - Simple or Composite: Simple
        - Dietary Fibers: Dietary fibers in a serving size
            - Domain/type: Integer
            - Value-range: [≥ 0]
            - Default value: [0]
            - Null Allowed: Yes
            - Unique: No
            - Single or Multi Value: Single

            - Simple or Composite: Simple
        - Sugars: Sugars in a serving size
            - Domain/type: Integer
            - Value-range: [≥ 0]
            - Default value: [0]
            - Null Allowed: Yes
            - Unique: No
            - Single or Multi Value: Single
            - Simple or Composite: Simple
    - Proteins: Large, complex molecules that are required for the structure, function, and regulation of the body's tissues and organs
        - Domain/type: Integer
        - Value-range: [≥ 0]
        - Default value: [0]
        - Null Allowed: Yes
        - Unique: No
        - Single or Multi Value: Single
        - Simple or Composite: Simple
    - Vitamins: Organic compounds needed in small quantities to sustain life
        - A: Vitamin A in a serving size
            - Domain/type: Integer
            - Value-range: [≥ 0]
            - Default value: [0]
            - Null Allowed: Yes
            - Unique: No
            - Single or Multi Value: Single
            - Simple or Composite: Simple
        - C: Vitamin C in a serving size
            - Domain/type: Integer
            - Value-range: [≥ 0]
            - Default value: [0]
            - Null Allowed: Yes
            - Unique: No
            - Single or Multi Value: Single
            - Simple or Composite: Simple
        - D: Vitamin D in serving size
            - Domain/type: Integer
            - Value-range: [≥ 0]
            - Default value: [0]
            - Null Allowed: Yes
            - Unique: No
            - Single or Multi Value: Single
            - Simple or Composite: Simple
        - E: Vitamin E in serving size
            - Domain/type: Integer
            - Value-range: [≥ 0]
            - Default value: [0]

                - Null Allowed: Yes
                - Unique: No
                - Single or Multi Value: Single
                - Simple or Composite: Simple
            - K: Vitamin K in serving size
                - Domain/type: Integer
                - Value-range: [≥ 0]
                - Default value: [0]
                - Null Allowed: Yes
                - Unique: No
                - Single or Multi Value: Single
                - Simple or Composite: Simple
    - Other:
        - Calcium: Builds bones and keeps them healthy. This attribute contains the amount contained in a serving size
            - Domain/type: Integer
            - Value-range: [≥ 0]
            - Default value: [0]
            - Null Allowed: Yes
            - Unique: No
            - Single or Multi Value: Single
            - Simple or Composite: Simple
        - Iron: Mineral that carries oxygen in the hemoglobin of red blood cells throughout the body so cells can produce energy. This attribute contains the amount of iron in a serving size.
            - Domain/type: Integer
            - Value-range: [≥ 0]
            - Default value: [0]
            - Null Allowed: Yes
            - Unique: No
            - Single or Multi Value: Single
            - Simple or Composite: Simple
        - Potassium: Mineral that is an electrolyte and assists in bodily functions such as blood pressure, normal water balance, muscle contractions, and so on. This attribute contains the amount of potassium in a serving size.
            - Domain/type: Integer
            - Value-range: [≥ 0]
            - Default value: [0]
            - Null Allowed: Yes
            - Unique: No
            - Single or Multi Value: Single
            - Simple or Composite: Simple
- Candidate Keys: All
- Primary Keys: Produce
- Strong/Weak Entity: Weak
- Fields to be indexed: All

Menu Items:

- Attributes:
    - ID: Identification of the menu item
        - Domain/type: Integer
        - Value-range: [≥ 000]
        - Default value: [000]
        - Null Allowed: No
        - Unique: Yes
        - Single or Multi Value: Single
        - Simple or Composite: Composite
    - Name: Name of the Menu Item
        - Domain/type: Char String
        - Value-range: [≥ 3 char]
        - Default value: [Name]
        - Null Allowed: No
        - Unique: Yes
        - Single or Multi Value: Single
        - Simple or Composite: Simple
    - Price: Price of the Menu Item
        - Domain/type: Integer
        - Value-range: [> 0]
        - Default value: [1]
        - Null Allowed: No
        - Unique: No
        - Single or Multi Value: Single
        - Simple or Composite: Simple
- Candidate Keys: ID, Name
- Primary Keys: ID
- Strong/Weak Entity: Strong
- Fields to be indexed: ID, Name, Price

Order Quantity:

- Attributes:
    - Price: Price of the quantity of menu items
        - Domain/type: Integer
        - Value-range: [> 0]
        - Default value: [1]
        - Null Allowed: No
        - Unique: No
        - Single or Multi Value: Single
        - Simple or Composite: Simple
- Candidate Keys: Menu Item (ID)
- Primary Keys: Menu Item (ID)
- Strong/Weak Entity: Weak
- Fields to be indexed: Price

1.2.2 Relationship Type Description:

Employee:

- Supervises Employee: Supervises an opening or closing shift and maintains the store while the manager is gone. Also assists customers as well as employees in answering questions. Gives a role to an employee as a supervisor who is an employee under another role.
    - Cardinality: 1...N
    - Descriptive field: None
    - Participation constraint: Total
- Works_for Store: Works for a Location and fulfils a specific role at the location
    - Cardinality: N...1
    - Descriptive field: Store Number
    - Participation constraint: Partial
- Creates Order: Employee creates order for customer
    - Cardinality: 1...1
    - Descriptive field: Date
    - Participation constraint: Total

Customer:

- Makes Order: Customer makes and order to an employee.
    - Cardinality: 1...N
    - Descriptive field: Date
    - Participation constraint: Total

Order:

- Contains Menu Item: Order contains menu items in quantity and size.
    - Cardinality: 1...N
    - Descriptive field: Produce
    - Participation constraint: Total
    - Contains Quantity: [description]
        - Cardinality: 1...N
        - Descriptive field: None
        - Participation constraint: Total

Menu Item:

- Made_From Produce: Menu items created from the inventory of the store.
    - Cardinality: 1...N
    - Descriptive field: None
    - Participation constraint: Total
    - Contains Quantity: [description]
        - Cardinality: 1..N
        - Descriptive field: None
        - Participation constraint: Total

Produce:

- Contains Nutrition: Nutritional facts contained within the produce.
    - Cardinality: 1...1
    - Descriptive field: None
    - Participation constraint: Total

Store:

- Holds Produce: Store contains produce within a location.
    - Cardinality: 1...N
    - Descriptive field: Type
    - Participation constraint: Total

Supplier:

- Sells Produce: Supplier sells produce to a store location]
    - Cardinality: 1...N
    - Descriptive field: Type
    - Participation constraint: Total
    - Contains Quantity: [description]
        - Cardinality: 1...N
        - Descriptive field: None
        - Participation constraint: Total

1.2.3 Related Entity Type:

Specialization/Generalization Relationships:

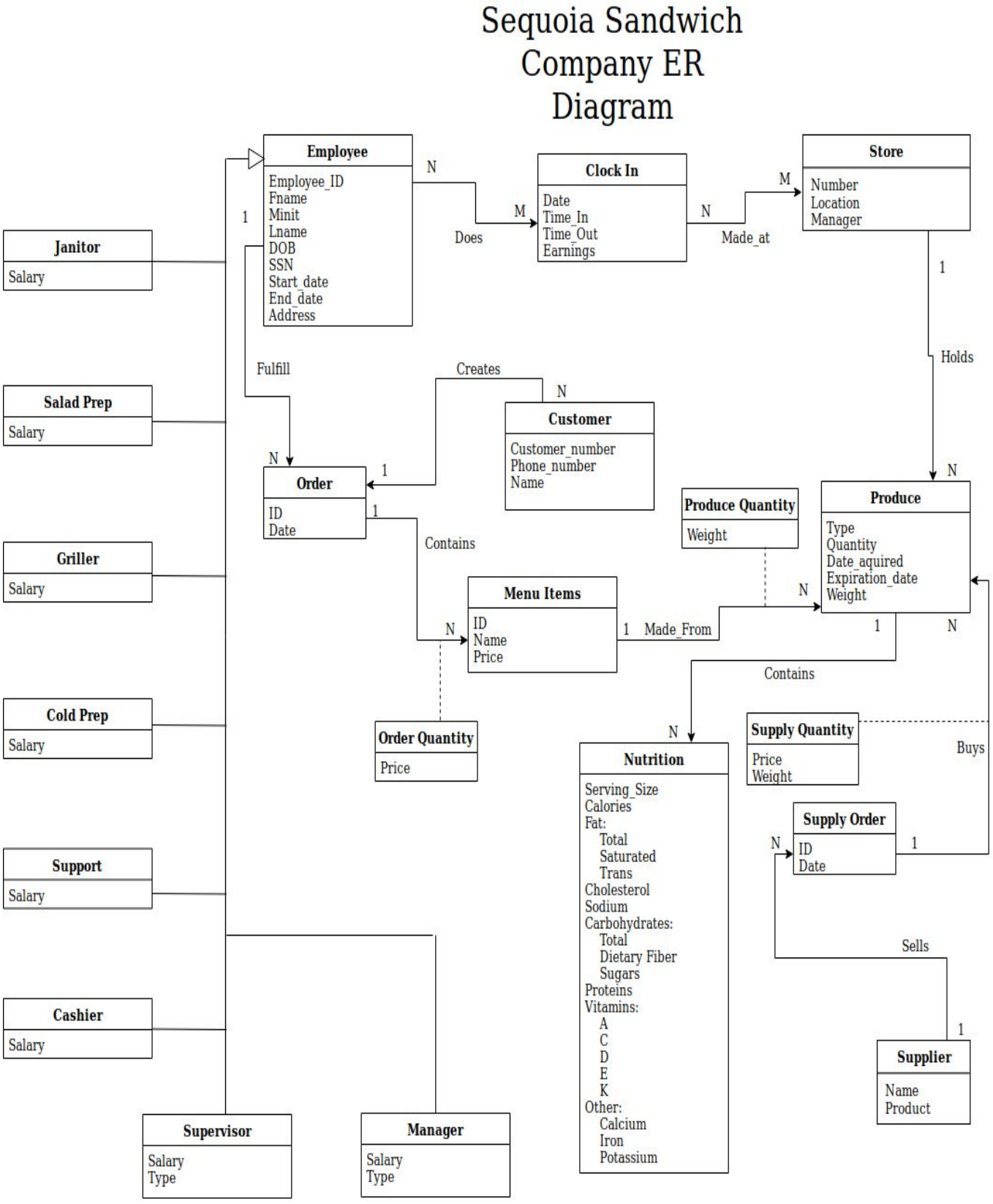
Employee as a superclass contains the subclasses: Manager, Supervisor, Cashier, Cold Prep, Salad Prep, Griller, Support, Janitor and Expeditor

Produce Types is a superclass that contains Meat, Cheese, Vegetables, Bread, and Condiments. Meat is a superclass that contains Ham, Turkey, Chicken, Beef, Salami, Pastrami, Tuna. Cheese is a superclass that contains Monterey, Pepper Jack, Cheddar, American, Provolone, Parmesan, Mozzarella, Swiss, Feta, Blue Cheese. Vegetables is a superclass that contains Lettuce, Tomatoes, Spinach, Sprouts, Pepperoccines, Bell Peppers, Jalapenos, Mushrooms, Egg Plant, Artichoke, Avocado. Bread is a superclass that contains French, Onion Dill Roll, Squaw, White, Wheat, Sourdough, Rye, Gluten Free, Croissant. Condiments is a superclass that contains Oil, Vinegar, Mayonnaise, Mustard, Pesto, Honey.

Menu Items is a superclass that contains Hot Sandwiches, Cold Sandwiches, Supreme Sandwich, Vegetarian Sandwiches, Salads, Bakery, Kid's Menu, Beverages and Sides. Hot Sandwiches is a superclass that contains BBQ Chicken, California Chicken Breast Sandwich, California Pastrami, Chicken Caesar, Chicken Fajita, Chicken Italiano, Classic Reuben, French Dip, Grilled Cajun Turkey, Hot Ham & Cheese, Hot Roast Beef, Monterey Chicken Club, Philly Cheesesteak, Southwestern Chicken, Tuna Melt, Turkey Bacon Melt, Turkey Dip, Tomato-Basil Mozzarella Panini, Eggplant-Artichoke Panini and Turkey Pesto Panini. Cold Sandwiches is a superclass that contains Albacore Tuna, BLT, Cajun Turkey, Chicken & Bacon Club, Club Deluxe, Cracked Pepper Turkey, Egg Salad, Ham & Baby Swiss, Heart Smart - Ham, Heart Smart - Turkey, Roast Beef, Roast Turkey Breast, Sequoia Sandwich, The Sicilian, Waldorf Chicken Sandwich and Lite Lunch. Supreme Sandwiches is a superclass that contains Double Hot Ham & Cheese, The All American, The Big Dipper, The Italian Stallion, The New Yorker and Turkey Club. Vegetarian Sandwiches is a superclass that contains Cold Veggie & Cheese, Deluxe Grilled Cheese, Eggplant-Artichoke Panini, Hot Vegetarian and Tomato-Basil Mozzarella Panini. Salads is a superclass that contains Antipasta Salad, Bistro Salad, Caesar Salad, California Chicken Salad, Chicken Caesar Salad, Cobb Salad, Garden Salad, Greek Salad, Mini Bistro, Mini Caesar Salad, Mini Garden Salad, Sesame Chicken Salad, Southwestern Chicken Salad, Spinach Kale and Slaw Salad, Spring Salad, Taco Salad, Tuna Salad, Waldorf Chicken Salad. Bakery is a superclass that contains Banana Pudding, Butter Cake, Carrot Cake, Frosted Fudge Brownies, Garys Famous Chocolate Cake, German Chocolate Cake, Homemade Cookies, Homemade New York Cheesecake, Lemon Berry Cake, New York Cheesecake w/strawberry sauce, Rice Krispy Treat. Kid's Menu is a superclass that contains Grilled Cheese, Turkey, Ham and Salami. Beverages is a superclass that contains Bottled Soda, Bottled Water, Hot Tea, Regular Fountain Drink, Sparkling Water, Sobe & Fuse, Specialty Sodas & Juices, Milk, Energy Drinks, Coffee, and Chocolate Milk. Sides is a superclass that contains Baked Potatoes, Bowl of Tuna, Bowl of Waldorf

Chicken Salad, Broccoli & Raisin, Broccoli-Pasta, Chili, Chili & Cornbread Special, Creamy Cole Slaw, Hot Sourdough Rolls, Macaroni, Potato Chips, Quinoa, Soup - Broth, Soup - Cream, Yukon Potato.

1.2.4 E-R Diagram:



Phase 2: Relation vs E-R Model

1 Conceptual Database and Logical Database

Section 2.1 of phase 2 will describe exactly what an E-R Model and what a Relational model is. Comparisons between the two models will describe their advantages and disadvantages when being used. 2.2.2 will describe how to change Relationship Types to Relations. 2.2.3 describes the purpose and use for Database Constraints.

1.1 E-R Model and Relational Model

1.1.1 Description of Models

History:

E-R Model - Data modeling became popular in and around the 1970s, stemming from the need to organize and structure databases and real-world business processes. A man named Peter Chen popularized the E-R model in a paper published in 1976. His paper mostly contributed to the rise of data modelling practice as it is known today, as it included words such as “entity” and “relationship”.

Relational Model - Proposed by E.F. Codd in 1969, the relational model is a method to organize data into grid-like mathematical structures known as relations. These structures contain rows and columns, and have created the basis for what we now know as “tables”. In a relational model, all data is stored in relations, and each relation must have headers and bodies.

Descriptions:

E-R Model - This model is meant to represent the objects as entities and the relationships between those entities.

Relational Model - This model uses tables and the relationship between tables to express the set.

Major Features:

E-R Model - Mapping Cardinality, Participation ratio, Keys, Entity sets, Relationship sets, and Attributes

Relational Model - Tables, Domains, Attributes, Records, and Tuples

Purpose:

E-R Model - A easier way of understanding the relationships and entities for a database. This model is meant as a starting point for conceptualizing how the functionality of the data and inputs should work.

Relational Model - This model is more difficult to understand, but closer to how its implementation would work when it is being coded. This model is somewhat of an intermediary step between the E-R Model and the coding of the database.

1.1.2 Comparison of the Two Models

The E-R Model focuses on the entities involved and their relationships, while the Relational Model focuses on Tables and the data that goes into those tables. Furthermore, the E-R Model defines the data with the entity sets, relationship sets and attributes. It is easy to tell relationships from an E-R Model. The Relational Model uses tuples, attributes and their domains to describe the data. Finally, the E-R Model shows cardinality, while the Conceptual model does not.

1.2 From Conceptual Database to Logical Database

This section will cover a range of conversions that will be used to change entity types to relations. For example: strong to weak entities.

1.2.1 Converting Entity Types to Relations

A strong entity is a key that itself is unique or consists of one or multiple attributes that are unique. For each strong (regular) entity type E, a relation R must be created that includes all of the simple attributes of E. One of the key attributes of E must be chosen as the primary key for R. If the key is composite, then the set of simple attributes that form it will form the primary key of R altogether. If multiple keys were identified for E, then the information describing the attributes that form each key is maintained in order to specify unique (secondary) keys of relation R. Knowledge of keys is also maintained for indexing purposes and other types of analyses.

Weak entities cannot exist without another primary key. They are entities that cannot be uniquely identified through attributes alone. So when mapping weak entity types W, create a relation R in which all simple attributes of W are included in R. All primary key attributes of the relation that correspond to the owner entity types must be

added as foreign keys. If there is a weak entity type E_2 that contains an owner that is also a weak entity type E_1, then E_1 should be mapped before E_2 to determine its primary key first.

1.2.2 Converting Relationship Types to Relations

1:1 Relationship Mapping:

In changing the 1-to-1 Relationship to a Relation, there are 3 possible approaches. First, we can use the Foreign Key approach. The Foreign Key approach includes using an attribute from either side of the relationship and including it in the other. For example, if we consider the Relationship between the entities Employee and Store. In this Relationship we consider the possible Foreign Keys that could be provided to the other. A quick glance over both shows that Employee has the attributes Fname, Minit, Lname, SSN, and Employee_ID that should be considered for a Foreign Key, and Store has Store_Number and Location which can be used. As for Employee, it makes more sense to use Employee_ID since it is not impossible for a multiple employees to possess the same exact Fname, Minit, and Lname, and it could pose a potential threat to employee security if each call to Employee is using an individual's SSN. Thus, it makes the most sense to use Employee_ID as the Foreign Key to refer to Employee. Looking at Store, Store_Number should be used because, while Location should be entirely unique, it is simply easier to refer to a store's number than its entire street, city, and country address every time it is called.

If the participation types between both entities is total, then it is possible to simply merge the two types. For Example, if there was only a single Store entry, then Location, Store_Number, and Manager_ID could be included in the Employee Class.

The third, and last, option to convert a 1-to-1 Relationship to a relation is to create a third entity that holds the Primary Keys of both. In our example, this would be like adding the entity Works At that only has attributes Store_Number and Employee_ID.

1:Many Relationship Mapping:

The primary approach to converting a 1 to many relationship from an E-R model to a Relational one, is to give the side that possess the N relationship and giving it the primary key of the 1 side. For example, in the database schema we have set up, each employee should be given the primary key of the store number that they work at.

The secondary method is to create another entity, and provide that entity with each of the primary keys of the two entities. This is essentially the same cross-reference method that was used as the last option for the 1 to 1 relationship.

Many:Many Relationship Mapping:

The only method for mapping a many to many relationship is to use cross-referencing/ This is, again, done by creating another entity that posses the primary keys of both related entities. Following our schema the Produce entity and Supplier entity or many to many. Thus, in order to deal with the cross reference we need to make another entity, Supply_Order to deal with this.

IsA/HasA:

Generally, to deal with superclassing and subclassing, there are four possible methods. Firstly, a new relation can be made for each subclass of a superclass, and each subclass is given the primary key of its superclass. Secondly, keep the subclasses as normal and simply add the primary key of the superclass to each subclass itself. Thirdly, get rid of all the subclasses, and create a new relation with the superclass that has attributes to define the type and hold the primary key of the superclass. Essentially shrinking the schema down. Finally, it is to give each entity a boolean value for each other entity, and should that entity be a subclass of an entity, set the boolean to true.

Relationship Types Involving Other Relationship Types - **Recursive Relationships**:

When it becomes necessary to map multiple relationships, or N-ary relationships, where both Entities can be referenced within each other, special steps must be taken. It is possible to create a new entity and give the entity each primary key. This may get overwhelming if there are a lot of recursive relationships to be considered. Another approach is to give each relationship the primary key of each other entity. This can also get overwhelming, as many entities might have a large number of other primary keys that need to be included, but can also be easier to maintain than having many more entities created to cross reference the entities in the former approach.

Relationships With More Than 2 Entity Types - **Relationship and Category Types**:

The general approach for handling a subclass that had two or more of its superclasses merged is to create a surrogate key. The surrogate key basically relate the entity to a category rather than an entity. It also allows for a user to identify all entities in a certain category.

1.2.3 Database Constraints

Database constraints are the limitations and restrictions of a database system. Most databases have many constraints, consisting of rules that pertain to the miniworld that the database represents. This allows for more efficient capturing and organizing of information gained from users. There are many types of constraints, but they are often placed into three categories: implicit constraints, explicit constraints, and business rules. Implicit constraints are the rules that are inherent in the database system. Those rules that must be put in place to create a functional database are the implicit constraints. For example, a database contains characteristics of relations, and those are often considered the inherent constraints. Explicit constraints are the constraints that can be directly expressed in the schema.

An entity constraint is a constraint that is put upon an entity, restricting or limiting its possible range of data. For example, an entity integrity constraint states that no primary key value can be NULL. This prevents the primary key from not being able to identify some tuples.

A primary key constraint is a constraint placed on a primary key preventing other primary keys from overwriting it.

A referential constraint specifies uniqueness on data item values, such that a specific entity must have a unique value for another entity. This is also known as a uniqueness constraint.

Business rules are more general and relate to meaning as well as behavior of attributes. These are normally checked within the application programs that perform database updates.

2 Converting E-R/Conceptual Database into a Relational/Logical Database

Once the E-R database has been created, we must use that to transform it into a relational or logical database. The purpose of this is to transform a “concept” into an actual design for a database. The relational database creates multiple relations to house data and information in a more organized manner. The E-R database model is the skeleton of the relational database. This allows us to take all information gained from the

E-R model and translate it to the relational model so we can more efficiently create and maintain data.

The translation from E-R to relational helps to root out problems and issues with the database involving redundancy and unnecessary inconsistencies. The idea is that the concepts taken from the E-R model are given more detail and description once translated over to the relational model. Each piece of information from the E-R model will be turned into a relation for the relational model.

2.1 Relation Schema for Local Database

Employee:

Employee_ID	Integer	000-999
Fname	String	> 0 characters
Minit	Char	> 2 characters
Lname	String	> 0 characters
Address	Integer, String, String, String, Integer	> 0, > 0 characters, > 0 characters, > 0 characters, 1001-99950
DOB	String, Integer, Integer	01-12, 00-99, 1940-2004
SSN	Integer, Integer, Integer	000-999, 00-99, 0000-9999
Start_Date	Integer, Integer, Integer	01-12, 00-99, 1940-2019
End_Date	Integer, Integer, Integer	Null | 01-12, 00-99, 1940-2019
Position	Char String	> 4 Char
Salary	Double	> 12.00
Store_Number	Integer	> 0

Candidate Key: Employee_ID, SSN

Primary Key: Employee_ID

Referential: The Employee_ID is used by other entities to refer to the Employee entity. Within the Employee entity Store_ID refers to the store at which the Employee works.

Store:

Store_Number	Integer	0-5
Location	Integer, String, String, String, Integer	> 0, > 0 characters, > 0 characters, > 0 characters, Integer
manager_ID	String	> 0 characters

Candidate Key: Store_Number, Location

Primary Key: Store_Number

Referential: The Store_Number is used to refer to specific store locations by other entities.

Customer:

Customer_Number	Integer	> 1
Phone_Number	Integer-Integer-Integer	000-999, 000-999, 0000-9999
Name	String	> 0 characters

Candidate Key: Customer_Number

Primary Key: Customer_Number

Referential: Customer_Number is used to reference which order of the day that this particular customer made

Order:

Order_ID	Integer	> 0
Date	Integer/Integer/Integer	1-12/1-31/1900-2019
Customer_ID	Integer	> 1
Employee_ID	Integer	000-999

Candidate Key: Order_ID

Primary Key: Order_ID

Referential: The Order_ID is used by other entities to refer to the Order by its unique integer value.

Menu Items:

Menu_ID	Integer	> 0
Item_Name	String	> 0 characters
Price	Double	> 0.0
Produce_ID	Integer	> 0
Order_ID	Integer	> 0

Candidate Key: Menu_ID, Item_Name

Primary Key: Menu_ID

Referential: The Menu_ID refers to an integer value for each unique Menu Item.

Produce:

Produce_Type	String	> 0 characters
Quantity	Integer	≥ 0
Date_Acquired	Integer/Integer/Integer	1-12/1-31/1900-2019
Expiration_Date	Integer/Integer/Integer	1-12/1-31/1900-2100
Weight	Double	> 0.0
Supplier_Order_ID	Integer	> 0
Produce_ID	Integer	> 0
Nutrition_ID	Integer	> 0

Candidate Key: Produce_Type, Produce_ID

Primary Key: Produce_ID

Referential: Produce_ID allows for integer reference to each unique type of produce

Supplier:

Supplier_ID	Integer	0-9
Supplier_Name	String	> 0 characters
Product	String	> 0 characters

Candidate Key: Supplier_ID, Supplier_Name

Primary Key: Supplier_ID

Referential: Supplier_ID is an integer reference to each unique supplier

Nutrition:

Nutrition_ID	Integer	> 0
Nutrition_Type	String	> 0 characters
Nutrition_Value	Integer	> 0

Candidate Key: Nutrition_ID, Nutrition_Name

Primary Key: Nutrition_ID

Referential: A unique integer that can be referenced by the produce to get the value of each nutritional type within that produce item

Order Quantity:

Order_ID	Integer	> 0
Price	Integer	> 0

Candidate Key: Can only be referenced from Order

Primary Key: Uses the Order_ID to be acquired from Order

Referential: Cannot be referenced itself, but can be retrieved through Order_ID

Produce Quantity:

Menu_ID	Integer	> 0
Menu_Item_Weight	Integer	> 0

Candidate Key: Can only be referenced from Menu

Primary Key: Uses the Menu_ID to be acquired from Menu

Referential: Cannot be referenced itself, but can be retrieved through Menu_ID

Supply Quantity:

Supplier_ID	Integer	> 0
Total_Weight	Integer	> 0
Supply_Price	Double	> 0.0

Candidate Key: Can only be referenced from Supplier

Primary Key: Uses the Supplier_ID to be acquired from Supplier

Referential: Cannot be referenced itself, but can be retrieved through Supplier_ID

Clock In:

Date	Integer, Integer, Integer	1-12, 1-31, 1900-2019
Time_In	Integer:Integer	0-23:0-59
Time_Out	Integer:Integer	0-23:0-59
Earnings	Double	> 0.0
Store_ID	Integer	> 0
Employee_ID	Integer	> 0

Candidate Key: Can be referenced from Employee_ID or Store_ID

Primary Key: Uses the Employee_ID or Store_ID

Referential: Cannot be referenced itself, but can be retrieved through Store_ID & Employee_ID

2.2 Sample Data of Relation

Employee:

Employee	497	Grant	K.	Granite	9989 Killian Way, Bakersfield CA, 93314	8/15/1990	809-38-921	10/23/2013	9/22/2014	Position	$12.00	4
Employee	77	Fred	F.	Frederick	703 Book Rd, Bakersfield CA, 93314	5/11/2003	374-46-304	11/16/1991	6/29/2008	Position	$12.00	3
Employee	513	Zachary	J.	Kaiser	3275 Page St, Bakersfield CA, 93311	7/26/1998	35-79-312	3/17/2003	8/6/1994	Position	$12.00	4
Employee	484	Zakary	E.	Worman	3382 Great Way, Bakersfield CA, 93311	8/9/1995	828-57-449	5/30/1993	9/20/2003	Position	$12.00	4
Employee	769	Peter	I.	Peterson	6968 Ming Ave, Bakersfield CA, 93314	1/28/1995	560-52-343	10/7/2011	8/12/1999	Position	$12.00	3
Employee	104	George	R.	Martin	5471 Old River Rd, Bakersfield CA, 93313	1/16/2010	663-94-525	8/28/2019	3/4/2014	Position	$12.00	3
Employee	728	Jolken	R. R.	Tolkien	6968 New River Rd, Bakersfield CA, 93311	2/20/2000	980-17-447	9/2/1998	10/23/2003	Position	$12.00	1
Employee	440	John	P.	Cailang	6536 Wherehelives Street, Bakersfield CA, 93314	3/18/1997	756-47-31	7/19/2010	2/15/2001	Position	$12.00	4
Employee	384	Natalie	K.	Dormer	3401 Rich St, Bakersfield CA, 93313	8/5/1999	739-15-66	1/25/2000	12/10/1992	Position	$12.00	2

Store:

Store	5	5810 Street Name, Bakersfield CA, 93314	552
Store	2	342 Street Name, Bakersfield CA, 93314	818
Store	2	2789 Street Name, Bakersfield CA, 93314	370
Store	3	8186 Street Name, Bakersfield CA, 93311	764
Store	2	4642 Street Name, Bakersfield CA, 93312	394
Store	1	5244 Street Name, Bakersfield CA, 93312	474
Store	3	7036 Street Name, Bakersfield CA, 93312	775
Store	1	1005 Street Name, Bakersfield CA, 93313	428
Store	3	7034 Street Name, Bakersfield CA, 93312	607
Store	2	7487 Street Name, Bakersfield CA, 93314	144

Customer:

Customer	56	(661) 145-1492	
Customer	22	(661) 028-7297	
Customer	96	(661) 537-6901	
Customer	84	(661) 626-3936	
Customer	96	(661) 301-2940	
Customer	93	(661) 712-6246	
Customer	37	(661) 856-9047	
Customer	90	(661) 158-5007	
Customer	56	(661) 623-1819	
Customer	48	(661) 173-2637	

Order:

Order	508	1/16/1947	14	622
Order	259	1/16/1975	19	352
Order	718	3/22/1925	97	485
Order	234	2/28/1907	7	694

Order	355	6/28/2019	99	563
Order	608	10/8/1938	44	686
Order	835	10/4/1972	13	521
Order	133	12/12/2003	21	790
Order	433	1/20/1957	3	548
Order	53	11/21/1913	51	819
Order	604	6/21/1999	88	408
Order	186	11/5/1916	92	283
Order	107	9/2/2018	10	672
Order	467	5/17/1910	24	095
Order	39	6/2/1995	14	467
Order	209	11/28/1922	65	430
Order	487	6/28/1913	87	573
Order	770	8/25/2008	21	316
Order	132	7/25/2003	12	025
Order	404	6/23/1958	2	669
Order	894	2/2/1985	12	563
Order	680	9/27/1960	84	931
Order	499	1/24/1933	70	266
Order	149	12/27/1930	59	304
Order	781	11/25/1961	22	591
Order	719	7/7/1926	27	726
Order	535	3/6/2015	4	719
Order	682	2/18/1941	38	230
Order	842	1/5/1933	15	686
Order	320	12/19/1924	5	175
Order	982	7/16/1923	71	129
Order	662	9/3/1944	8	030
Order	340	6/11/1920	85	765
Order	153	10/4/1906	50	920

Order	93	4/27/2000	16	737
Order	836	1/7/1933	91	017
Order	954	8/8/2014	10	947
Order	2	3/13/1919	9	083
Order	799	1/22/2002	4	216
Order	297	3/5/1995	95	042
Order	689	4/7/1924	35	745
Order	834	7/12/1926	98	341
Order	552	12/6/1950	58	791
Order	60	8/8/1960	78	146
Order	629	9/14/1978	16	793
Order	879	3/26/1996	8	981
Order	759	3/27/2004	24	287
Order	931	10/13/1945	9	377
Order	594	10/28/1921	25	980
Order	850	2/23/1901	78	376
Order	982	4/20/1933	16	134
Order	917	8/31/1921	70	350
Order	984	1/20/1938	12	134
Order	651	6/15/1969	51	585
Order	309	1/23/1943	39	577
Order	632	9/7/2009	96	370
Order	354	11/31/1929	94	637
Order	179	1/29/2019	66	016
Order	579	1/22/1902	59	609
Order	455	7/26/1932	44	194

Menu Items:

Menu Item	2	California Pastrami	$14.64	92	3
Menu Item	4	New Yorker	$14.54	33	18

Menu Item	9	Chilli	$07.02	54	2
Menu Item	1	Chicken Caesar Sandwich	$11.33	82	73
Menu Item	2	Spring Salad	$18.73	95	95
Menu Item	5	Roasted Turkey Breast	$17.04	40	49
Menu Item	7	Club Deluxe	$13.64	51	88
Menu Item	7	Turkey Club	$04.54	39	41
Menu Item	7	The Sicilian	$14.43	48	28
Menu Item	6	Italian Stallion	$17.87	42	92

Produce:

Produce	Roast Turkey	77	7/4/1999	6/12/1990	10185	614	612	222
Produce	Provolone	58	10/10/1998	2/28/2016	13033	116	693	184
Produce	Lettuce	89	6/1/2002	4/10/2007	3705	930	963	422
Produce	Yellow Mustard	82	6/5/1994	1/22/1995	10284	103	223	35
Produce	Mayonnaise	100	2/12/2014	8/7/2017	457	7	310	827
Produce	Salami	64	4/2/2014	3/10/1991	12961	956	727	177
Produce	Pepperoncinis	43	4/28/2002	12/2/2003	7986	218	677	395
Produce	Sprouts	49	3/1/2019	8/25/2000	9515	63	347	348
Produce	Broccoli	36	9/30/2003	3/2/2004	3992	849	360	588
Produce	Mushrooms	58	9/14/1998	9/20/1996	10424	143	843	571

Supplier:

Supplier	102	Sysco	Product
Supplier	102	Sysco	Product
Supplier	102	Sysco	Product
Supplier	102	Sysco	Product
Supplier	102	Sysco	Product

Supplier	102	Sysco	Product
Supplier	102	Sysco	Product
Supplier	102	Sysco	Product
Supplier	102	Sysco	Product
Supplier	102	Sysco	Product

Nutrition:

Nutrition	983	Carbohydrates	7749
Nutrition	282	Vitamin_A	6936
Nutrition	942	Total_Fat	7296
Nutrition	432	Saturated_Fat	8341
Nutrition	883	Sodium	3046
Nutrition	984	Cholesterol	97
Nutrition	363	Protein	7465
Nutrition	273	Calcium	9585
Nutrition	249	Iron	4915
Nutrition	415	Vitamin_C	5476

Order_Quantity:

Order_Quantity	39	$7774.83
Order_Quantity	83	$533.65
Order_Quantity	47	$1247.27
Order_Quantity	72	$5987.94
Order_Quantity	49	$8171.14
Order_Quantity	49	$625.48
Order_Quantity	42	$4737.39
Order_Quantity	78	$4076.96
Order_Quantity	87	$8065.16
Order_Quantity	46	$1471.18

Order_Quantity	56	$1124.53
Order_Quantity	8	$7060.2
Order_Quantity	89	$4429.1
Order_Quantity	97	$6179.56
Order_Quantity	79	$3351.99
Order_Quantity	96	$7808.1
Order_Quantity	17	$5989.88
Order_Quantity	28	$8131.46
Order_Quantity	9	$2085.49
Order_Quantity	58	$9059.28
Order_Quantity	62	$4692.10
Order_Quantity	64	$2580.61
Order_Quantity	78	$8343.76
Order_Quantity	27	$7129.78
Order_Quantity	22	$3776.18
Order_Quantity	21	$1082.15
Order_Quantity	24	$4587.7
Order_Quantity	57	$8303.77
Order_Quantity	17	$3718.62
Order_Quantity	46	$3669.32
Order_Quantity	53	$1769.48
Order_Quantity	28	$2238.88
Order_Quantity	65	$7754.85
Order_Quantity	32	$3653.63
Order_Quantity	30	$70.54
Order_Quantity	66	$5233.33
Order_Quantity	91	$8759.83
Order_Quantity	8	$9584.77
Order_Quantity	58	$3421.11
Order_Quantity	100	$9519.3

Order_Quantity	16	$6460.14
Order_Quantity	3	$8262.25
Order_Quantity	38	$2507.85
Order_Quantity	100	$2597.90
Order_Quantity	38	$8419.76
Order_Quantity	90	$1708.35
Order_Quantity	73	$7686.81
Order_Quantity	35	$3346.68
Order_Quantity	29	$9820.79
Order_Quantity	80	$9734.32
Order_Quantity	38	$2265.24
Order_Quantity	14	$8897.82
Order_Quantity	33	$5887.58
Order_Quantity	100	$8175.36
Order_Quantity	72	$7654.87
Order_Quantity	32	$8276.65
Order_Quantity	56	$6752.5
Order_Quantity	18	$891.16
Order_Quantity	51	$5307.12
Order_Quantity	85	$5515.20

Produce_Quantity:

Produce_Quantity	13	12459
Produce_Quantity	48	9519
Produce_Quantity	28	2132
Produce_Quantity	80	1844
Produce_Quantity	0	2558
Produce_Quantity	18	8594
Produce_Quantity	5	11058
Produce_Quantity	45	1924

Produce_Quantity	9	9990
Produce_Quantity	46	7738
Produce_Quantity	17	7764
Produce_Quantity	74	3576
Produce_Quantity	91	8145
Produce_Quantity	29	7278
Produce_Quantity	33	3614
Produce_Quantity	34	12222
Produce_Quantity	50	8329
Produce_Quantity	74	13119
Produce_Quantity	98	4038
Produce_Quantity	92	683
Produce_Quantity	28	11925
Produce_Quantity	75	8136
Produce_Quantity	99	4837
Produce_Quantity	5	5921
Produce_Quantity	22	10776
Produce_Quantity	17	10499
Produce_Quantity	33	12542
Produce_Quantity	75	7718
Produce_Quantity	83	4730
Produce_Quantity	19	7833
Produce_Quantity	5	11276
Produce_Quantity	74	12693
Produce_Quantity	74	8144
Produce_Quantity	15	8614
Produce_Quantity	5	6089
Produce_Quantity	73	8394
Produce_Quantity	13	6898
Produce_Quantity	46	6510

Produce_Quantity	45	3966
Produce_Quantity	59	13540
Produce_Quantity	32	7801
Produce_Quantity	1	10596
Produce_Quantity	54	1247
Produce_Quantity	63	4827
Produce_Quantity	13	12674
Produce_Quantity	38	12020
Produce_Quantity	62	8769
Produce_Quantity	83	7602
Produce_Quantity	7	11369
Produce_Quantity	72	6009
Produce_Quantity	25	13352
Produce_Quantity	66	1551
Produce_Quantity	31	9711
Produce_Quantity	1	1047
Produce_Quantity	59	9004
Produce_Quantity	68	3538
Produce_Quantity	94	2902
Produce_Quantity	33	4820
Produce_Quantity	96	3285
Produce_Quantity	10	12771

Supplier Quantity:

Supplier_Quantity	1	68	$553.59
Supplier_Quantity	1	55	$579.68
Supplier_Quantity	1	74	$491.44
Supplier_Quantity	1	58	$496.38
Supplier_Quantity	1	12	$300.4
Supplier_Quantity	1	84	$421.65

Supplier_Quantity	1	34	$639.1
Supplier_Quantity	1	56	$349.0
Supplier_Quantity	1	15	$986.42
Supplier_Quantity	1	90	$689.89
Supplier_Quantity	1	39	$240.33
Supplier_Quantity	1	5	$58.24
Supplier_Quantity	1	32	$975.82
Supplier_Quantity	1	17	$89.46
Supplier_Quantity	1	73	$203.25
Supplier_Quantity	1	10	$96.51
Supplier_Quantity	1	78	$465.79
Supplier_Quantity	1	88	$995.50
Supplier_Quantity	1	72	$728.62
Supplier_Quantity	1	25	$274.59
Supplier_Quantity	1	4	$474.84
Supplier_Quantity	1	83	$24.39
Supplier_Quantity	1	30	$38.92
Supplier_Quantity	1	99	$139.22
Supplier_Quantity	1	23	$741.12
Supplier_Quantity	1	35	$165.1
Supplier_Quantity	1	6	$39.60
Supplier_Quantity	1	0	$743.60
Supplier_Quantity	1	69	$628.42
Supplier_Quantity	1	66	$837.53
Supplier_Quantity	1	63	$279.46
Supplier_Quantity	1	73	$820.96
Supplier_Quantity	1	49	$179.61
Supplier_Quantity	1	8	$851.35
Supplier_Quantity	1	86	$228.93
Supplier_Quantity	1	62	$576.73

Supplier_Quantity	1	18	$316.44
Supplier_Quantity	1	83	$263.2
Supplier_Quantity	1	93	$185.71
Supplier_Quantity	1	25	$565.23
Supplier_Quantity	1	71	$347.52
Supplier_Quantity	1	0	$814.98
Supplier_Quantity	1	43	$420.58
Supplier_Quantity	1	37	$458.40
Supplier_Quantity	1	7	$506.32
Supplier_Quantity	1	45	$922.47
Supplier_Quantity	1	91	$219.24
Supplier_Quantity	1	26	$981.28
Supplier_Quantity	1	66	$103.15
Supplier_Quantity	1	95	$77.99
Supplier_Quantity	1	81	$416.3
Supplier_Quantity	1	54	$818.41
Supplier_Quantity	1	17	$96.7
Supplier_Quantity	1	16	$914.88
Supplier_Quantity	1	9	$776.60
Supplier_Quantity	1	18	$925.11
Supplier_Quantity	1	58	$450.29
Supplier_Quantity	1	51	$326.3
Supplier_Quantity	1	72	$189.56
Supplier_Quantity	1	64	$408.50

Clock In:

Clock_In	11/18/1906	11:41	12:21	$17.23	4	993
Clock_In	9/4/1927	5:11	1:44	$346.6	4	203

Clock_In	9/24/1965	19:41	4:43	$400.17	3	838
Clock_In	7/5/2003	16:7	6:24	$32.29	3	283
Clock_In	12/8/1929	1:7	1:27	$205.37	2	776
Clock_In	9/28/1937	12:25	6:41	$101.90	1	529
Clock_In	12/11/1930	2:41	11:29	$270.55	4	322
Clock_In	10/14/1958	10:11	1:20	$461.71	2	980
Clock_In	7/6/1990	9:52	14:7	$398.27	2	330
Clock_In	5/20/2008	11:4	17:20	$8.61	1	837
Clock_In	4/19/1963	7:58	9:56	$139.50	3	851
Clock_In	8/19/1904	18:57	9:25	$316.77	2	317
Clock_In	8/23/2019	20:4	7:51	$236.19	4	10
Clock_In	1/31/2015	3:56	22:57	$81.57	3	832
Clock_In	10/16/1992	6:30	5:41	$375.10	2	270
Clock_In	12/6/2001	15:26	6:58	$334.89	1	831
Clock_In	11/27/1974	4:11	11:55	$335.98	2	414
Clock_In	3/3/1970	14:58	4:55	$447.31	2	259
Clock_In	5/16/1967	0:3	16:3	$340.69	2	93
Clock_In	7/4/1963	22:11	4:38	$344.51	1	389
Clock_In	9/9/1963	7:16	11:15	$155.4	2	247
Clock_In	2/7/1989	3:32	7:18	$240.98	3	465

Clock_In	4/5/2001	12:15	0:26	$422.36	3	78
Clock_In	1/31/1919	8:6	15:45	$131.83	1	856
Clock_In	11/5/1998	12:26	12:38	$406.58	3	596
Clock_In	6/15/2003	17:14	11:7	$318.25	2	891
Clock_In	1/6/1947	10:30	8:15	$256.57	3	320
Clock_In	4/31/2017	15:28	8:30	$100.29	2	403
Clock_In	7/27/1975	16:43	10:2	$222.33	2	201
Clock_In	9/29/1996	10:35	20:11	$45.66	2	844
Clock_In	5/14/1941	20:4	2:45	$376.1	1	701
Clock_In	3/5/1970	9:12	17:48	$447.24	4	311
Clock_In	4/31/1996	19:36	17:58	$499.10	1	757
Clock_In	9/31/2015	15:21	7:44	$136.85	3	478
Clock_In	3/19/2017	5:38	13:50	$230.47	3	562
Clock_In	9/29/1984	21:42	3:24	$179.96	1	423
Clock_In	2/4/2006	3:9	7:6	$53.7	3	166
Clock_In	8/13/1915	16:22	14:53	$266.55	3	151
Clock_In	3/23/1928	5:42	7:4	$458.5	2	426
Clock_In	11/9/1916	10:13	22:4	$416.33	4	874
Clock_In	7/22/2005	3:5	17:54	$53.66	2	973
Clock_In	10/16/1971	1:25	21:24	$96.60	2	315

Clock_In	6/21/2003	7:42	14:35	$489.65	1	549
Clock_In	4/10/1923	18:54	18:25	$185.79	1	455
Clock_In	8/7/1959	15:26	1:54	$338.80	4	36
Clock_In	6/25/1940	15:56	8:56	$333.66	3	806
Clock_In	8/13/2018	21:15	23:38	$471.15	4	968
Clock_In	3/2/1996	10:10	3:51	$258.24	3	909
Clock_In	1/5/1982	5:45	2:20	$36.56	2	750
Clock_In	4/11/1932	3:1	14:58	$34.53	4	791
Clock_In	1/18/2006	5:41	9:33	$142.98	4	634
Clock_In	10/30/1989	18:54	11:58	$126.12	2	589
Clock_In	10/31/1957	7:0	22:13	$60.76	1	864
Clock_In	4/20/1964	10:22	7:17	$148.44	3	813
Clock_In	7/16/2015	18:26	10:38	$201.80	2	10
Clock_In	4/25/1972	2:50	9:49	$172.55	1	306
Clock_In	11/6/2001	11:21	11:3	$231.72	2	169
Clock_In	11/4/1922	11:26	17:10	$22.72	4	173
Clock_In	5/18/1905	22:28	1:59	$288.50	2	404
Clock_In	2/18/1999	19:12	7:43	$440.13	3	758

3 Sample Queries for the Database

Section 4 will consist of creations of queries to search through our database. 4.1 will show a list of entities and their attributes, which will allow for the creation of queries. The next sections will show relational algebra and calculus expressions.

3.1 Design of Queries

Employee (Employee_ID, Fname, Minit, Lname, Address, DOB, SSN, Store_Number, Start_Date, End_Date)
Janitor(Salary, Employee_ID)
Salad Prep(Salary, Employee_ID)
Griller(Salary, Employee_ID)
Cold Prep(Salary, Employee_ID)
Support(Salary, Employee_ID)
Cashier(Salary, Employee_ID)
Supervisor(Salary, Employee_ID, Type)
Manager(Salary, Employee_ID, Type)
Clock In(Employee_ID, Time_In, Time_Out, Earnings, Store_ID)
Order (Employee_ID, Customer_Number, Order_ID, Date)
Customer (Customer_Number, Phone_number, Customer_Name)
Menu Items (Order_ID, Menu_ID, Item_Name, Price, Produce_ID)
Produce (Produce_Type, Quantity, Date_Acquired, Expiration_Date, Weight, Supply_Order_ID, Produce_ID, Nutrition_ID)
Produce Quantity(Produce_ID, Menu_ID, Menu_Item_Weight)
Order Quantity (Order_ID, Order_Price)
Supply Order (Supply_Order_ID, Supplier_ID, Supply_Date, Store_Number)
Supply Quantity (Supplier_ID, Supply_Price, Total_Weight, Produce_ID)
Store (Store_Number, Location, Manager_ID)
Nutrition (Nutrition_ID, Nutrition_Type, Nutrition_Value)
Supplier (Supplier_ID, Supplier_Name, Product)

1. Name the menu items that would have less than 1000 calories
2. Name the Menu Items that would be under $10
3. List all employees who have taken an order from the same customer more than once in the same week.
4. List the most expensive produce items that were ordered in the past month.
5. List the store that profits the most.
6. List the customers who have never ordered the same item.

7. List the customers who have ordered the same order at least three times.
8. List the employees who have worked more than 40 hours in the past week or worked more than 8 hours in a single day.
9. List employees who have worked in every location.
10. List the store that has purchased the most produce in the past month.

3.2 Relational Algebra Expressions

1. Name the menu items that would have less than 1000 calories
2. Name the Menu Items that would be under $10
3. List all employees who have taken an order from the same customer more than once in the same week.
4. List the most expensive produce items that were ordered in the past month.
5. List the store that profits the most.
6. List the customers who have never ordered the same item.
7. List the customers who have ordered the same order at least three times.
8. List the employees who have worked more than 40 hours in the past week or worked more than 8 hours in a single day.
9. List employees who have worked in every location.
10. List the store that has purchased the most produce in the past month.

1. $\pi_{\text{Item\_Name}}(\sigma\ \text{n.Nutrition\_ID} \wedge \text{n.Nutrition\_Value} < 1000(\text{Nutrition}))$
2. $\pi_{\text{Item\_Name}}(\sigma_{\text{price} < 10}(\text{Menu Items}))$
3. $\sigma_{\text{o1.Employee\_ID} = \text{o2.Employee\_ID}}$ **AND** o1.Customer_Name = o2.Custmoer_Name **AND (**o1.Customer_Number != o2.Customer_Number **OR** o1.Date != o2.Date) **AND** o1.Date is within same week as o2.Date((Order * Customer) ^{o1} x (Order * Customer) ^{o2})
4. $\sigma_{\text{s1.Supply\_Price} > \text{s2.Supply\_Price}}$ **AND** s1.Supply_Date in past month **AND** s2.Supply_Date in past month((Supply Order * Supply Quantity) ^{s1} x (Supply Order * Supply Quantity) ^{s2})
5. π (σ s1.Store_Number = s2. Store_Number Λ s1.Location = s2.Location (Store)(σ sp.Order_ID = s1.Store_Number Λ s1.Store_Number > s2.Store_Number Λ s1.Store_Number > o.Order_Price) (Supply Order * Supply Quantity * Store)
6. $\pi_{\text{customer\_name}}(\sigma_{\text{c1.customer\_name} = \text{c2.customer\_name}}$ **AND** c1.Menu_ID != c2.Menu_ID((Customer * Order * Menu Items)^{c1} x (Customer * Order * Menu Items)^{c2})
7. $\pi_{\text{customer\_name}}(\sigma_{\text{c1.customer\_name} = \text{c2.customer\_name}}$ **AND** c1.Menu_ID = c2.Menu_ID **AND** c1.Menu_ID = c3.Menu_ID **AND** c1.Order_ID != c2.Order_ID **AND** c1.Order_ID != c3.Order_ID **AND** c2.Order_ID != c3.Order_ID((Customer * Order * Menu Items)^{c1} x (Customer * Order * Menu Items)^{c2} x (Customer * Order * Menu Items)^{c3})
8. $\pi_{\text{Employee\_ID}}(\sigma_{\text{Time\_Out - Time\_In} > 8}(\text{Employee} * \text{Clock In}))$
9. $\pi_{\text{Employee\_ID}}((\text{Employee} * \text{Location}) \div \text{Location})$

10. $\pi_{Store\_Number}(G_{\mathbf{sum}(cost)}((Store * Supply\ Order * Supply\ Quantity)^{s1} \times (Store * Supply\ Order * Supply\ Quantity)^{s2})$

3.3 Tuple Relational Calculus Expressions

1. Name the menu items that would have less than 1000 calories
2. Name the Menu Items that would be under $10
3. List all employees who have taken an order from the same customer more than once in the same week.
4. List the most expensive produce items that were ordered in the past month.
5. List the store that profits the most.
6. List the customers who have never ordered the same item.
7. List the customers who have ordered the same order at least three times.
8. List the employees who have worked more than 40 hours in the past week or worked more than 8 hours in a single day.
9. List employees who have worked in every location.
10. List the store that has purchased the most produce in the past month.

1. {m | Menu Items(m) ∧ (∃ n) (Nutrition(n) ∧ n.Nutrition_Value < 1000)}
2. {m | Menu Items(m) ∧ m.price < 10}
3. {e | Employee(e) ∧ (∃ o1)(∃ o2) (Order(o1) ∧ Order(o2) ∧ o1.Employee_ID = o2.Employee_ID ∧ o1.Customer_Name = o1.Customer_Name ∧ ¬(o1.Customer_Number != o2.Customer_Number ∨ o1.Date != o2.Date) ∧ o1.Date is within a week of o2.Date))}
4. {p | (Produce(p) ∧ Order(o)) (∀) p.Produce_ID ∧ (∃ o) ∧ o.Order_ Price ∧ p.Quantity >= “30” o.Date)}
5. {s | Store(s) ∧ Order Quantity(q) (∀ s) s.Store_Number ∧ s.Location ∧ s.Manager_ID) (∧ ∃ > q.Order_ID ∧ q.Order_Price)}
6. {c | (Customer(c) (∃ c1)(∃ c2) (∧ Order(o) ∧ Order Quantity(q)) (∀ c1 ∧ c2) ≠ (∃ o) o.Order_ID ∧ q.Order_Price}
7. {{c | (Customer(c) (∃ c1)(∃ c2) (∧ Order(o) ∧ Order Quantity(q)) (∀ c1 ∧ c2) = (∃ o) o.Order_ID ∧ q.Order_Price >= 3 times}
8. {e | Employee(e) ∧ (Clock in(cn) (∀ e) (cn.Empolyee_ID ∧ cn.Time_in) > 40 hours < 168 hours V (cn.Empolyee_ID ∧ cn.Time_in) > 8 hours = 24 hours}
9. {e | Employee(e) ∧ Store(s) (∀e. ∀s) (e.Empolyee_ID ∧ s.Location)}

10. {s | Store(s) (∃ so) (∃ p) (∃ q) (∃ o) (Supply Order(so) ∧ (Produce(p) ∧ Order Quantity(q) ∧ Order(o)) (s.Store_Number ∧ s.Location) > p.Produce_ID ∧ q.Order_Price > o.Date = 30 days))}

3.4 Domain Relational Calculus Expressions

1. Name the menu items that would have less than 1000 calories
2. Name the Menu Items that would be under $10
3. List all employees who have taken an order from the same customer more than once in the same week.
4. List the most expensive produce items that were ordered in the past month.
5. List the store that profits the most.
6. List the customers who have never ordered the same item.
7. List the customers who have ordered the same order at least three times.
8. List the employees who have worked more than 40 hours in the past week or worked more than 8 hours in a single day.
9. List employees who have worked in every location.
10. List the store that has purchased the most produce in the past month.

1. {<n> | Menu Item(_, _, n, _, _) ∧ (∀ id)(∀ p)((Menu Item(_, id, n, _, p) ∧ (∃ w)(∃ n)Produce(_, _, _, _, w, _, _, n))) ➔ (∃ v)(Nutrition(n, “Calories”, v) ∧ v*w > 1000))}
2. {<n> | Menu Item(_, _, n, > 10, _)}
3. {<id, f, m, l> | Employee(id, f, m, l, _, _, _, _, _, _) ∧ (∃ c)(∃ n)(Customer(c, _, n) ∧ Customer(!=c, _, n) ∧ (∃ d)(Order(=n, =id, d, > 1 week ago) ∧ Order(!=n, id, !=d, > 1 week ago)))}
4. {<n, id> | Produce(n, _, > 1 month ago, _, _, _, id, _) ∧ (∃ p1)(∃ p2)(Supply Quantity(_, p1, _, id) ∧ Supply Quantity(_, p2, _, id) ∧ p1 > p2)}
5. {<n> | Store(n, _, _) ∧ (∃ o1)(∃ o2)(Order Quantity(o1, o2) ∧ Order Quantity(!=o1, > o2))}
6. {<c, n> | Customer(c, _, n) ∧ (∀ id)(Order(_, n, id, _) ➔ Order(_, n, !=id, _))}
7. {<c, n> | Customer(c, _, n) ∧ (∃ id) (∃ d1)(∃ d2)(Order(_, n, id, d1) ∧ Order(_, n, id, d2) ∧ Order(_, n, id, !=d1 || !=d2) ∧ d1 != d2)}
8. {<id, f, m, l> | Employee(id, f, m, l, _, _, _, _) ∧ (∃ i)(∃ o)(Clock In(id, i, o, _, _) ∧ (o - i) ≥ 8) (∀ i)(∀ o)(Clock In(id, i, o, _, _) ➔ (o - i) ≥ 40)}
9. {<id, f, m, l> | (∀ s)(Employee(id, f, m, l, _, _, _, s, _, _) ➔ Store(s, _, _))}

10. {<sn> | Store(sn, _,_) (∃ q) (Produce (_, q, _, > 30 days,_, _,_, _, _) ∧ (sn = q ∧ > 30 days))}

Phase 3: Relational Database Normalization & Implementation

1 Normalization and PostgreSQL Implementation

1.1 Normalization of Relations

1.1.1 What is Normalization?

Normalization of data is the process of analyzing the given relation schemas and minimize redundancy, insertion, deletion, and update anomalies. The schemas are examined in terms of their functional dependencies as well as their primary keys. It is a sort of “filtering” or “purification” process to provide better design results. Any unsatisfactory schema that do not meet the normal form tests are decomposed into smaller relation schemas that meet the tests and then possess the desired properties. There are different types of normal forms, including first, second, and third (1NF, 2NF, and 3NF respectively). There is also the Boyce-Codd normal form.

The first normal form is part of the formal definition of a relation in the basic relational model. It has been used to disallow multivalued attributes, composite attributes, and their combinations. It states that the domain of an attribute must include only atomic values and that the value of any tuple must be a single value from the domain of that attribute. This disallows having a tuple of values, a set of values, or both as an attribute value for a single tuple. Only single or atomic values are permitted in 1NF.

The second normal form is based on “full functional dependency”. Something X ⟶ Y is a full functional dependency if removal of any attribute A from X means that the dependency does not hold any more. This means that for any attribute of A that belongs to X, (X - {A}) does not functionally determine Y. The test for functional dependency involves testing for functional dependencies whose left-hand side attributes are part of the primary key. If the primary key contains one attribute, the test is unnecessary.

The third normal form is based on “transitive dependency”. A functional dependency X ⟶ Y is transitive in dependency if there exists a set of attributes Z in R that is neither a candidate key or subset of any key of R,[10] and both X ⟶ Z and Z ⟶ Y

hold. A relational schema R can satisfy 3NF if it satisfies 2NF and no non-prime attribute of R is transitively dependent on the primary key.

The Boyce-Codd normal form was proposed as a simpler form of 3NF, but was found to actually be more strict. Every relation in BCNF is also in 3NF, but a relation in 3NF is not necessarily in BCNF. A relation schema R is BCNF if whenever a non-trivial functional dependency $X \longrightarrow A$ holds in R, then X is a superkey of R. The formal definition of BCNF differs from 3NF in that the condition of 3NF which allows A to be prime is absent from BCNF. In practice, most relation schemas in 3NF are in BCNF as well.

When normalizing the relations in a database, it is possible to create update anomalies. Update anomalies are additional problems that are created from adding and deleting relations. These anomalies can come in the form of unusual data such as empty departments with no employees. There are some possible anomalies that may appear in our database. Some examples would be creating an order with no menu items in it, creating a menu item with no ingredients in it such as condiments. An employee may not show up to work one day, and the position may not be filled, and that is also an anomaly that can possibly occur. These issues that occur can cause unnecessary data to appear, information to be incorrectly entered, and so on.

1NF Relations: Menu Items, Supplier, Nutrition, Order Quantity, Produce Quantity, Order

2NF Relations: Menu Items, Supplier, Nutrition, Order Quantity, Produce Quantity, Order

Relations to be normalized: Employee, Produce, Clock_In, Store, Customer

For Employee, possible update anomalies could be extra copies of information such as store location and updated locations or store_ID's must be changed for all employees. For produce, if one is deleted, it could possibly delete other information involving nutritional facts. For modification anomalies, if a menu item is changed in the database, it must be changed everywhere else the menu item is mentioned.

1.1.2 3rd and Boyce-Codd Normal Forms

Briefly we can describe Boyce-Codd Normal Forms (BCNF) as a smaller form of 3NF, but BCNF is more complex or stricter than 3NF. In the same time, each relation in BCNF should be found in 3NF, but the opposite is not required.

3NF: Supplier, Store, Customer, Order_Quantity, Produce_Quantity, Supply_Quantity

BCNF: Menu_ID, Order_Quantity, Produce_Quantity, Supply_Quantity,

1.2 Main Purpose of PSQL and Functionality Provided by PSQL

When we need to manage and control our database system, we need to use a language to handle that such as, Structured Query Language which called nowadays SQL. Oracle company in databases offered multiple ways to connect and organize the database easily either by using SQL, MySQL, PostgreSQL, etc. Because the reliability of PSQL, its main purpose to allow the user to customize and add functions by using our formal programming languages, such as C, C++, Python, or Java.

There are many advanced functionalities that PSQL is provided in advance than other database management systems. Such as, tables. constraints, stored procedures, user defined types, table inheritance, foreign key referential, and tablespaces is added recently in the new versions.

1.3 Description of Schema Objects Allowed in Postgres DBMS

Postgres Database Management Systems allow the following data objects; data types, functions, and operators.

The first data object, data types, has the following categories types: numeric, monetary, character, binary, date/time, boolean, enumerated, geometric, network address, bit string, text search, UUID, XML, arrays, composite, object identifier, and pseudo. It is also possible for users to create a new data type by using CREATE TYPE.

The Numeric Types are 2, 4 or 8 byte integers, 4 or 8 byte floating-point numbers, and precision decimals. Integer types are whole numbers and come in the standar data types of smallint, integer, and bigint. There are also arbitrary precision numbers that can store up to 1000 digits of precision. Obviously, these numeric types are far slower than integer types. This is essentially limited to the numeric data type. Following, and quite similar, are the floating-point types. This body consists of the real and double data types. These are faster the arbitrary precision numbers, but also less accurate and can sometimes be rounded or estimated. Finally, there are serial types. Serial types are actually only a short hand in creating unique identifier columns.

The Monetary Types are used to store currency with fixed fractional precision. In this sense, they are an extension of the Numeric Types. The only unique functionality of this type is the keyword money, which is an 8 bytes integer specifically for storing currency.

The Character Types are used to store strings or words. There are three data types from this area. They are character varying(n), which will create a string up to n characters, truncating inputs shorter than n. The call of character(n) will do almost the same thing, but will pad the remaining length after input with spaces. The keyword text can be used to create strings of unlimited length. All of these functionalities make use of the “char” type, which will store a 1-byte representation of a letter.

The Binary Data Types are sets of bytes similar to character strings; however, with the following differences. (1) They are intended to allow for storing octets of value zero and other octets that cannot be printed, and (2) the operations done to these octets are done at the byte level, while character processing is done to the representation the bytes are taking on. This is done by using the bytea key word, and it can store 1 or 4 bytes along with the binary string.

The Date/Time Types are used for exactly what their name suggests, storing dates and times in an easily accessible and alterable manner. From these, the keywords, timestamp, date, time, and interval are allowed. Timestamp can be used to store date and time with or without time zone down to the microsecond. Date will store the date without time. Time will store the time and can be saved with or without the date, and interval can be used to specify and time interval.

The Boolean Type are used to store values that are either true or false. Sometimes, a null value is used to represent unknown values. The value of true can be represented as TRUE, t, true, y, yes, on, or 1, while false is represented by false, f, false, n, no off, or 0.

The Enumerated Types are static, ordered sets of values. To use these you must create a type as an enumerated type and list the values in the type from smallest to largest. Then giving inputs a type can be ordered and displayed.

The Geometric Types are used to represent 2-Dimensional shapes. Their types include point, line, lseg, box, path (open or closed), polygon, and circle. Point is a 16 byte represented as (x, y) for points on a plane. A line is 32 bytes, and it is written as ((x1,y1),(x2,y2)). The lseg type is used in the same manner as the line, but instead of being

considered infinite, it starts and ends at each point. Path is a type that is represented closed as: ((x1,y1),...)) and open as: [(x1,y1),...)]. A polygon must be created by giving each of its points with the same syntax as a closed path, and then they will be considered connected. Finally, a circle is represented by <(x,y),r>, where (x,y) is it center point, and r is its radius.

The Network Address Types are used to store certain types of data generally applicable to internet and connection usage. These are used over character types because they intrinsically do error checking. The inet type is used to store and IPv4 or IPv6 host address and subnet. The cidr type stores only the network for IPv4 and IPv6, and not the host. Furthermore, inet will allow for nonzero bits to the right of the netmask, and cidr will not allow this. Lastly, there is the macaddr type which stores a MAC address.

The Bit String Types are strings of 1's and 0's. Most often, these are used to store bit masks. There is bit(n) and bit varying(n) with the same distinctions as in the character types.

The Text Search Types allow for searching through strings and documents. There is functionality, such as tsvector which show a sorted list of normalized words. There is also tsquery which stores lexemes that can be searched.

The UUID Types are universally unique identifiers. It holds a 128 bit value, and it is generated using and algorithm that makes it nearly impossible for the same identifier to be generated by anyone else using the same algorithm.

The XML Types is strictly used for reading and writing from an xml file. It has functionality to handle headers and all other types of encoding in an xml file.

Arrays are another data type that can be used in Postgres Database Management Systems. They work in the same way that they do in every other programing language. Where you define the type and then with braces give a number to define the length.

The Composite Types represents the structure of a row or record; it is essentially just a list of field names and their data types. They are almost like classes or structures in C++.

Object Identifier Types are used internally as primary keys for system tables. To use such types you must use the key words WITH OIDS. They include regproc, regprocedure, regoper, regoperator, regclass, regtype, regconfig, and regdictionary.

The Pseudo-Types have special purpose entries: any, anyarray, anyelement, anyenum, anynonarray, cstring, internal, language_handler, record, trigger, void, and opaque. These used to indicate whether the function accepts certain types, to trigger functions, or to identify return types.

Functions and operators are used in conjunction to give output from the data types and use data types to get these outputs. As is with the the Data Types, there are many different functions and operators: logical, comparison, mathematical, string, binary, bit string, data type, date/time, geometric, network address, sequence manipulation, array, aggregate, set returning, system information, and system administration. Furthermore, it is possible to define new functions.

The Logical Operators available in Postgres DBMS are the basic AND, OR, NOT.

The Comparison Operators available in Postgres DBMS are less than (<), greater than (>), less than or equal to (<=), greater than or equal to (>=), equal to (=), and not equal to (<> or !=). There are also IS statements to determine if something is a data type or value. Such as ISNULL or IS NULL, IS NOT NULL or NOTNULL, IS DISTINCT FROM, IS TRUE, IS NOT TRUE, IS FALSE, IS NOT FALSE, IS UNKNOWN, IS NOT UNKNOWN.

There are multiple mathematical functions available in Postgres DBMS. The functions include: *abs(x)* (takes the absolute value of a value), *cbrt(dp)* (takes the cube root of a value), *ceil(dp or numeric)* (takes the smallest integer not less than the argument), *degrees(dp)* (which converts radians to degrees), *exp(dp or numeric)* (takes the exponential value), *floor(dp or numeric)* (takes the largest integer not greater than the argument), *ln(dp or numeric)* (which takes the natural log of a value), *log(dp or numeric)* (a base 10 logarithm), *log(b numeric, x numeric)* (a logarithm to base b), *mod(y, x)* (the remainder of y/x), *pi()* (a “π” constant), *power(a dp, b dp)* (a raised power of b), *power(a numeric, b numeric)* (a raised to the power of b), *radians(dp)* (degrees to radians), *random()* (random value in the range 0.0 <= x < 1.0), *round(dp or numeric)* (round to nearest integer), *round(v numeric, s int)* (round to s decimal places), *setseed(dp)* (set seed for subsequent random() calls(value between -1.0 and 1.0, inclusive)), *sign(dp or numeric)* (sign of the argument (-1, 0, +1)), *sqrt(dp or numeric)* (square root), *trunc(dp or numeric)* (truncate toward zero), *trunc(v numeric, s int)* (truncate toward zero), *width_bucket(op numeric, b1 numeric, b2 numeric, count int)* (return the bucket to which operand would be assigned in an equidepth histogram with count buckets, in the range b1 to b2), and *width_bucket(op dp, b1 dp, b2 dp, count int)* (return the bucket to which operand would be assigned in an equidepth histogram with count buckets, in the range b1 to b2).

Trigonometric functions are also included, such as acos(x), asin(x), atan(x), atan(x,y), cos(x), cot(x), sin(x), and tan(x).

String functions and operators are included to examine and manipulate string values. Strings in this context include values of type *character, character varying, and text*. Functions include: *string* || *string* (string concatenation), *string* || *non-string* or *non-string* || *string* (string concatenation with one non-string input), *bit_length(string)* (number of bits in string), *char_length(string)* or *character_length(string)* (number of characters in string), *lower(string)* (Convert string to lowercase), *octet_length(string)* (Number of bytes in string), *overlay(string placing string from int [for int])* (location of specified substring), *substring(string [from int] [for int])* (extract substring), *substring(string from pattern)* (extract substring matching POSIX regular expression), *substring(string from pattern for escape)* (extract substring matching SQL regular expression), *trim([leading | trailing | both] [characters] from string)* (remove the longest string containing only characters (a space by default) from the start/end/both ends of the string), *upper(string)* (convert string to uppercase).

There are also Binary String Functions and Operators. They define some string functions that use keywords, rather than commas, to separate arguments. These strings include: *string* || *string* (string concatenation), *get_bit(string, offset)* (extract bit from string), *get_byte(string, offset)* (extract byte from string), *octet_length(string)* (number of bytes in binary string), *position(substring in string)* (location of specified substring), *set_bit(string, offset, newvalue)* (set bit in string), *set_byte(string, offset, newvalue)* (set byte in string), *substring(string [from int] [for int])* (extract substring), *trim([both]) bytes from string)* (remove the longest string containing only the byes in bytes from the start and end of string).

Bit String Functions and Operators are used to change bit strings. The basic operations are concatenation (||), bitwise AND (&), bitwise OR (|), bitwise XOR (#), bitwise NOT (~), bitwise shift left (<<), and bitwise shift right (>>). There are functions such as length, bit_length, octet_length, position, and substring that do exactly what their name entails.

Data Type Formatting Functions provide a powerful set of tools for converting various data types to formatted strings and for converting from formatted strings to specified data types. Functions include: *to_char(timestamp, text)* (convert timestamp to string), *to_char(interval, text)* (convert interval to string), *to_char(int, text)* (convert integer to string), *to_char(double precision, text)* (convert real/double precision to string),

to_char(numeric, text) (convert numeric to string), and other functions that convert various data types.

Date/Time Functions and Operators are especially useful for modifying dates and times. The operators + and - can be used to add or subtract days, months, years, or even intervals from the date/time data types. The * and / operator can be used to give the scalar multiplication and division of a date or time data type. The functions for this data type include getting the current date, finding the age of an input, seeing if values are finite, etc.

Enum Support Functions allow cleaner programming without hard-coding particular values of enum type. Functions from this section include: *enum_first(anyenum)* (returns first value of the input enum type), *enum_last(anyenum)* (returns the last value of the input enum type), *enum_range(anyenum)* (returns all of the values of the input enum type in an ordered array), *enum_range(anyenum, anyenum)* (returns the range between two given enum values, as an ordered array and the values must be from the same enum type. If the first parameter is null, the result will start with the first value of the enum type. If the second parameter is null, the result will end with the last value of the enum type.

Geometric Operators are all included in Postgres, and the types point, box, lseg, line, path, polygon, and circle have a large set of native support functions and operators. Operators included are: *+, -, *, /, #* (point or box of intersection), *#* (number of points in path or polygon, @-@ (length or circumference), @@ (center), *##* (closest point to first operand on second operand) <-> (Distance between), *&&* (Overlaps), << (is strictly left of?), >> (is strictly right of?), *&<* (does not extend to the right of?), *&>* (does not extend to the left of?), <<| (is strictly below?), >>| (is strictly above?), and many more.

Geometric Functions include: *area(object)* (double precision) *center(object)* (point), *diameter(circle), height(box), isclosed(path), isopen(path), length(object), npoints(path), npoints(polygon), pclose(path), popen(path), radius(circle),* and *width(box)*. Type conversion functions include: *box(circle), box(point, point), box(polygon), circle(box), circle(point, double precision), circle(polygon), lseg(box), lseg(point, point), path(polygon), point(double precision, double precision), point(box), point(circle), point(lseg), point(polygon), polygon(box), polygon(circle), polygon(npts, circle), polygon(path)*.

Network Address Functions and Operators are used to the cidr and inet types. The operators *<<, <<=, >>, and >>=* test for subnet inclusion. Operators include: *<, <=, =, >=, >, <>, <<, <<=, >>, >>= contains or equals, ~, &,* |, *+, -*. Functions include: *abbrev(inet)* (abbreviated display format as text), *abbrev(cidr)* (abbreviated display format as text),

braodcast(inet) (broadcast address for network), *family(inet)* (extract family of address), *host(inet)* (extract IP address as text), *hostmask(inet)* (construct host mask for network), *masklen(inet)* (extract netmask length), *netmask(inet),* construct netmask for network, *network(inet)* (extract network part of address), *set_masklen(inet, int)* extract network part of address, *text(inet)* (extract IP address and netmask length as text).

Sequence Manipulation Functions can be used to parse through sequences. It uses the functions, nextval, currval, lastval and setval.

Array Functions and Operators follow mostly the same syntax for operations as the bit string operations. The functions are used to concatenate arrays, append to an array, prepend, get positions of elements, set it to a string, etc.

Aggregate Functions compute a single result value from a set of input values. These types of functions include averaging, bitwise operations, count, finding every instance, max and min, standard deviation, sums, and variance.

Window Functions provide the ability to perform calculations across sets of rows that are related to the current query row. Functions include: *row_number(), rank(), dense_rank(), percent)rank(), cume_dist(), ntile(num_buckets integer), lag(value any[, offset integer [, default any]]), lead(value any [,offset integer [, default any]]), first_value(value any), last_value(value, any), nth_value(value any, nth integer).*

1.4 Relations: Relation Schema and its Contents

C_Order:

```
zak=# \d c_order
                Table "public.c_order"
    Column     |  Type   | Collation | Nullable | Default
---------------+---------+-----------+----------+---------
 employee_id   | integer |           | not null |
 customer_num  | integer |           | not null |
 menu_id       | integer |           | not null |
 order_id      | integer |           | not null |
 order_date    | date    |           | not null |
Indexes:
    "c_order_pkey" PRIMARY KEY, btree (order_id)
Foreign-key constraints:
    "c_order_customer_num_fkey" FOREIGN KEY (customer_num) REFERENCES customer(customer_num)
    "c_order_employee_id_fkey" FOREIGN KEY (employee_id) REFERENCES employee(employee_id)
    "c_order_menu_id_fkey" FOREIGN KEY (menu_id) REFERENCES menu_items(menu_id)
Referenced by:
    TABLE "order_quantity" CONSTRAINT "order_quantity_order_id_fkey" FOREIGN KEY (order_id) REFERENCES c_order(order_id)
```

```
zak=# SELECT * FROM c_order ORDER BY order_id ASC;
 employee_id | customer_num | menu_id | order_id | order_date
-------------+--------------+---------+----------+------------
          14 |            6 |     107 |        0 | 2014-11-11
          17 |            1 |      74 |        1 | 2001-12-17
          10 |           21 |       4 |        2 | 2013-06-20
          38 |           23 |       4 |        3 | 2017-09-17
          43 |            5 |      73 |        4 | 2000-09-15
           4 |            1 |      10 |        5 | 2011-06-07
          29 |           12 |      85 |        6 | 2010-05-19
          22 |           16 |      94 |        7 | 2017-04-06
          27 |           49 |     100 |        8 | 2010-02-15
          41 |            3 |      94 |        9 | 2007-06-12
          29 |           19 |      93 |       10 | 2018-01-12
          24 |           48 |      65 |       11 | 2005-01-20
           7 |           18 |      70 |       12 | 2016-10-03
           8 |           41 |       2 |       13 | 2003-11-13
          15 |           22 |      56 |       14 | 2014-03-03
          24 |           44 |      34 |       15 | 2016-10-08
          30 |            4 |      95 |       16 | 2010-05-20
           1 |           28 |     107 |       17 | 2007-07-04
          26 |           10 |      38 |       18 | 2019-08-08
          16 |           31 |     108 |       19 | 2003-12-17
          17 |           10 |      47 |       20 | 2019-02-08
          33 |           47 |      97 |       21 | 2011-06-07
          48 |            5 |      55 |       22 | 2019-10-04
          16 |           39 |      37 |       23 | 2009-06-13
          44 |            7 |       3 |       24 | 2007-11-06
          16 |           48 |      97 |       25 | 2009-03-14
          21 |           33 |      68 |       26 | 2006-09-09
           8 |           15 |      48 |       27 | 2005-03-19
          21 |           21 |      53 |       28 | 2001-05-14
          38 |            2 |      43 |       29 | 2000-03-10
          46 |           10 |      27 |       30 | 2015-09-08
          42 |           16 |      32 |       31 | 2018-08-15
          13 |            7 |      28 |       32 | 2002-02-11
          47 |           38 |     100 |       33 | 2015-08-16
          13 |           13 |     104 |       34 | 2014-06-16
          15 |           41 |      88 |       35 | 2015-08-18
          17 |           14 |      11 |       36 | 2010-12-08
          18 |           36 |      82 |       37 | 2013-07-19
          12 |           17 |      89 |       38 | 2013-01-10
          12 |           19 |      15 |       39 | 2006-05-20
          34 |           35 |      53 |       40 | 2000-03-09
          21 |           22 |      57 |       41 | 2004-11-18
          23 |           12 |     105 |       42 | 2011-09-03
          38 |           35 |       5 |       43 | 2017-05-11
          11 |           42 |     103 |       44 | 2017-05-01
          32 |           18 |      53 |       45 | 2005-01-12
          30 |           40 |       2 |       46 | 2009-11-15
          19 |           33 |      56 |       47 | 2006-10-05
          39 |            2 |      91 |       48 | 2010-01-13
           8 |           24 |      59 |       49 | 2007-08-14
(50 rows)
```

Clock_In:

```
zak-# \d clock_in
                        Table "public.clock_in"
    Column    |            Type             | Collation | Nullable | Default
--------------+-----------------------------+-----------+----------+---------
 employee_id  | integer                     |           | not null |
 time_in      | timestamp without time zone |           | not null |
 time_out     | timestamp without time zone |           | not null |
 earnings     | real                        |           | not null |
 store_number | integer                     |           |          | 1
Check constraints:
    "ck_times" CHECK (time_in < time_out)
    "clock_in_earnings_check" CHECK (earnings > 0::double precision)
Foreign-key constraints:
    "clock_in_employee_id_fkey" FOREIGN KEY (employee_id) REFERENCES employee(employee_id)
    "clock_in_store_number_fkey" FOREIGN KEY (store_number) REFERENCES store(store_num)
```

```
zak=# SELECT * FROM clock_in ORDER BY time_in DESC;
 employee_id |       time_in       |      time_out       | earnings | store_number
-------------+---------------------+---------------------+----------+--------------
          35 | 2019-08-18 04:16:36 | 2019-08-18 05:18:11 |     2.66 |            0
          48 | 2019-08-10 09:09:36 | 2019-08-10 22:14:42 |    32.35 |            1
          15 | 2019-08-02 17:04:25 | 2019-08-02 22:34:36 |    32.29 |            0
          32 | 2018-12-05 12:13:45 | 2018-12-05 19:18:56 |     4.29 |            2
          41 | 2018-08-20 06:54:41 | 2018-08-20 16:59:42 |    45.18 |            0
          45 | 2018-07-04 09:54:54 | 2018-07-04 14:11:41 |     95.8 |            1
          47 | 2017-06-22 05:00:56 | 2017-06-22 23:57:25 |    55.65 |            1
          39 | 2017-03-24 17:22:46 | 2017-03-24 23:16:25 |    28.14 |            1
          17 | 2017-03-21 22:16:43 | 2017-03-21 23:57:22 |    15.53 |            2
          39 | 2017-03-15 19:34:50 | 2017-03-15 20:11:01 |    47.48 |            1
          46 | 2017-01-15 13:56:25 | 2017-01-15 19:22:32 |    72.13 |            0
           2 | 2016-11-15 19:39:32 | 2016-11-15 22:38:41 |    77.17 |            2
          42 | 2016-11-04 06:08:02 | 2016-11-04 11:48:39 |      8.1 |            1
          29 | 2016-10-09 05:15:37 | 2016-10-09 14:51:31 |    54.06 |            2
          22 | 2016-06-17 15:55:00 | 2016-06-17 17:00:38 |    43.82 |            0
          35 | 2016-04-18 05:37:16 | 2016-04-18 17:42:52 |    58.43 |            0
          13 | 2016-03-23 00:09:03 | 2016-03-23 21:30:24 |    33.73 |            1
           9 | 2015-11-11 22:30:25 | 2015-11-11 23:41:41 |    53.77 |            2
           3 | 2015-07-25 05:15:43 | 2015-07-25 06:58:50 |      9.6 |            0
           3 | 2014-10-10 02:20:50 | 2014-10-10 11:22:41 |    49.84 |            0
          31 | 2014-04-09 20:17:12 | 2014-04-09 23:56:04 |     4.95 |            2
          30 | 2014-02-18 00:08:20 | 2014-02-18 09:53:54 |    58.97 |            1
          26 | 2013-06-21 00:14:40 | 2013-06-21 05:07:50 |    98.46 |            1
          23 | 2012-09-06 21:50:18 | 2012-09-06 23:12:21 |    45.24 |            2
          45 | 2012-08-21 00:31:45 | 2012-08-21 06:47:21 |    84.46 |            1
           5 | 2012-02-15 22:50:35 | 2012-02-15 23:11:55 |    17.84 |            2
          39 | 2012-01-07 20:04:18 | 2012-01-07 21:49:29 |    41.56 |            2
           9 | 2011-06-25 17:21:24 | 2011-06-25 23:53:55 |    90.75 |            2
          36 | 2011-03-23 13:36:53 | 2011-03-23 23:29:26 |    21.49 |            2
          25 | 2011-03-17 12:24:15 | 2011-03-17 17:48:49 |    29.95 |            2
           3 | 2010-06-01 11:48:41 | 2010-06-01 22:49:23 |    87.31 |            2
          20 | 2009-03-09 08:40:06 | 2009-03-09 11:44:07 |     8.39 |            2
          48 | 2008-10-21 04:31:18 | 2008-10-21 08:19:33 |        7 |            2
          35 | 2008-02-05 01:57:46 | 2008-02-05 13:10:25 |    60.62 |            1
          16 | 2007-02-24 11:55:42 | 2007-02-24 22:50:29 |    12.64 |            0
           8 | 2006-05-12 12:54:00 | 2006-05-12 20:50:40 |     60.8 |            2
           3 | 2006-01-24 09:05:58 | 2006-01-24 16:17:35 |     9.45 |            2
           1 | 2005-12-21 13:22:18 | 2005-12-21 14:39:06 |    15.38 |            1
          37 | 2005-11-18 11:36:23 | 2005-11-18 19:50:36 |    72.51 |            2
          35 | 2005-01-09 10:42:53 | 2005-01-09 21:52:44 |    48.43 |            0
          42 | 2004-09-13 03:09:16 | 2004-09-13 13:15:37 |    71.63 |            0
          34 | 2004-06-14 08:58:07 | 2004-06-14 19:55:36 |    63.02 |            2
          49 | 2004-03-10 10:16:31 | 2004-03-10 19:22:44 |    10.32 |            1
          24 | 2003-03-21 05:48:11 | 2003-03-21 07:24:03 |     70.9 |            1
           6 | 2003-03-05 21:48:07 | 2003-03-05 23:29:22 |    79.44 |            0
          16 | 2003-03-04 21:16:07 | 2003-03-04 22:15:35 |    16.42 |            1
           9 | 2003-01-09 19:53:22 | 2003-01-09 20:11:32 |      1.9 |            2
          41 | 2002-04-19 11:45:57 | 2002-04-19 21:24:51 |    17.69 |            2
          14 | 2001-01-12 16:13:56 | 2001-01-12 21:26:52 |    32.27 |            2
          39 | 2000-06-17 11:36:16 | 2000-06-17 23:32:45 |    81.32 |            2
(50 rows)
```

Customer:

```
zak=# \d customer
                    Table "public.customer"
    Column    |         Type          | Collation | Nullable | Default
--------------+-----------------------+-----------+----------+---------
 customer_num | integer               |           | not null |
 phone_number | character varying(50) |           |          |
 name         | character varying(50) |           | not null |
Indexes:
    "customer_pkey" PRIMARY KEY, btree (customer_num)
Referenced by:
    TABLE "c_order" CONSTRAINT "c_order_customer_num_fkey" FOREIGN KEY (customer_num) REFERENCES customer(customer_num)
```

```
zak=# SELECT * FROM customer ORDER BY name DESC;
 customer_num | phone_number  |         name
--------------+---------------+----------------------
           39 | (283)786-1856 | Val Iwasa
            0 | (873)934-0950 | Theressa Drury
           23 | (116)707-8348 | Stanwood Lancaster
           15 | (525)424-4081 | Shannon Masih
           46 | (524)536-0625 | Sawyer Aylett
           26 | (553)220-6689 | Rosamond Divaev
           22 | (647)028-9482 | Rhianon Mcghee
           28 | (308)505-0736 | Randi Vass
           34 | (583)220-6505 | Quinton Ryjkin
            2 | (740)521-4646 | Petronille Than
           20 | (986)799-9459 | Petronille Moghadam
           27 | (964)137-7493 | Petronia Ventura
           48 | (321)733-8484 | Paulina Dantsig
            4 | (213)180-2332 | Pamella Tahan
            1 | (329)275-2915 | Noelle Jefferies
           32 | (203)310-9122 | Natala Gibson
           10 | (698)854-9661 | Miran Mansour
           42 | (955)840-5698 | Marje Blanco
           24 | (514)681-5054 | Mario Jefferies
            5 | (141)043-6347 | Mariette Zogby
            7 | (253)809-5095 | Kissee Zogby
           16 | (294)730-0495 | Kincaid Zheravin
           47 | (800)024-2813 | Kimmi Kobi
           43 | (380)028-8621 | Julio Ivanov
           45 | (346)817-6157 | Jock Arrigucci
           13 | (237)733-6103 | Jerrie Rothery
           31 | (408)090-5765 | Jennee Ganim
            6 | (655)550-9576 | Jeana Axon
            9 | (614)839-8582 | Iris Fakhoury
           41 | (140)654-2475 | Hortense Lachapelle
           35 | (735)614-5092 | Hewett Demichev
           49 | (021)082-5159 | Gunvor Svoboda
            8 | (931)596-7464 | Griselda Richards
           19 | (854)308-0224 | Gerrard Liu
           14 | (006)951-8846 | Gail Munaev
           17 | (554)140-3969 | Forster Jaffray
           29 | (222)005-4390 | Filip Wagstaff
           12 | (413)031-3493 | Eugenie Golovatsky
           33 | (280)515-3517 | Ester Toichkin
           36 | (960)251-2913 | Enrika Salzwedel
           11 | (260)647-6031 | Elizabeth Nasikovsky
           40 | (964)979-5220 | Deedee Hawtin
           21 | (963)040-2936 | Darth Etherton
           25 | (985)733-4912 | Chase Ashida
           30 | (110)551-1365 | Carlos Nakazawa
           38 | (262)258-0080 | Bryna Bagmut
            3 | (912)394-2744 | Bryan Marino
           44 | (973)798-5419 | Andriette Astley
           37 | (862)515-4136 | Amber Buonarroti
           18 | (465)553-4329 | Amalee Novosiltsev
(50 rows)
```

Employee:

```
zak-# \d employee
                    Table "public.employee"
    Column     |         Type          | Collation | Nullable | Default
---------------+-----------------------+-----------+----------+---------
 employee_id   | integer               |           | not null |
 fname         | character varying(20) |           | not null |
 minit         | character varying(1)  |           |          |
 lname         | character varying(20) |           | not null |
 address       | character varying(50) |           | not null |
 dob           | date                  |           | not null |
 ssn           | character varying(11) |           | not null |
 store_number  | integer               |           |          | 1
 start_date    | date                  |           | not null |
 end_date      | date                  |           |          |
 position      | character varying(20) |           |          |
 salary        | real                  |           |          | 12.00
Indexes:
    "employee_pkey" PRIMARY KEY, btree (employee_id)
Check constraints:
    "ck_employee_dates" CHECK (start_date < end_date OR end_date IS NULL)
    "employee_position_check" CHECK ("position"::text = ANY (ARRAY['janitor'::character varying, 'salad prep'::character varying, 'grille
r'::character varying, 'cold prep'::character varying, 'support'::character varying, 'cashier'::character varying, 'supervisor'::characte
r varying, 'manager'::character varying]::text[]))
    "employee_salary_check" CHECK (salary >= 12.00::double precision)
Foreign-key constraints:
    "employee_store_number_fkey" FOREIGN KEY (store_number) REFERENCES store(store_num)
Referenced by:
    TABLE "c_order" CONSTRAINT "c_order_employee_id_fkey" FOREIGN KEY (employee_id) REFERENCES employee(employee_id)
    TABLE "clock_in" CONSTRAINT "clock_in_employee_id_fkey" FOREIGN KEY (employee_id) REFERENCES employee(employee_id)
```

File Edit View Search Terminal Help

employee_id	fname	minit	lname	address	dob	ssn	store_number	start_date	end_date	position	salary
49	Guillema	A	Horowitz	3517 S CHESAPEAKE AVE 90016	1967-05-20	558-32-4021	1	2009-02-10	2016-02-14	cold prep	15.83
48	Joyan	S	Oppenheimer	11603 W HAYNES ST 91606	1927-08-13	082-90-7408	1	2011-05-20	2018-12-08	support	15.18
47	Jami	U	Turubanov	2644 N CAHUENGA BLVD EAST 90068	1923-01-16	038-27-1406	1	2001-03-20		griller	18.26
46	Yehudit	P	Soho	2230 W 14TH ST 90006	1968-04-11	732-59-0972	1	2002-02-18		supervisor	15.48
45	Virgie	D	Tubinov	2161 E 101ST ST 90002	1928-11-08	995-90-9290	0	2009-01-01		cold prep	16.93
44	Kania	F	Tyman	111 1/2 S FLORES ST 90048	1954-09-09	283-60-6397	1	2009-12-06		griller	19.34
43	Valencia	Y	Gridnev	3038 S 5TH AVE 90018	1915-04-08	043-13-4548	0	2005-07-13	2016-10-07	cashier	15.72
42	Melisa	T	Neave	22275 W MACFARLANE DR 91364	1981-08-19	510-54-7713	2	2015-10-17		janitor	13.39
41	Camilla	P	Gaber	127 E AVENUE 45 90031	1904-06-08	528-53-0376	2	2003-02-02		salad prep	13.96
40	Bobbette	H	Vu	1247 W 18TH ST 90731	1960-12-20	907-87-9851	2	2012-02-07	2019-03-04	manager	15.63
39	Samuel	A	Wrench	727 N CRESCENT HEIGHTS BLVD 90046	1989-02-20	817-96-6464	0	2003-01-19	2016-01-06	salad prep	19.31
38	Noni	X	Garner	854 E 118TH ST 90059	1924-08-04	148-86-8337	0	2000-02-02	2018-03-07	cold prep	12.21
37	Murdock	O	Ilsley	3980 E GLENALBYN DR 90065	1961-05-14	431-92-8321	2	2013-06-14	2019-05-07	cashier	14.54
36	Cass	Z	Tomas	10776 W TABOR ST 90034	1946-05-08	168-99-2593	0	2009-04-08	2019-12-01	griller	14.62
35	Mike	G	Kerridge	23155 W OXNARD ST 91367	1945-03-05	996-40-4033	0	2009-10-07		cold prep	19.5
34	Kylie	O	Solberg	2931 S SEPULVEDA BLVD 90064	1958-07-01	399-34-8368	1	2006-04-06	2018-01-10	manager	18.07
33	Rianon	G	Félix	4800 W OAKWOOD AVE 90004	1916-05-01	942-54-3290	2	2008-01-15		supervisor	12.65
32	Kellen	P	Giannakos	5840 N ETIWANDA AVE 1-4 91356	1919-12-07	432-18-6372	2	2010-02-02		supervisor	12.8
31	Gilbert	W	Samuel	666 W SLAUSON AVE 90044	1969-04-06	946-90-7133	2	2014-10-06	2016-03-11	cashier	17.15
30	Rubin	C	Antar	257 1/2 W 80TH ST 90003	1968-06-03	432-89-0516	0	2004-11-07		supervisor	18.19
29	Margery	M	Deeb	11362 W HOMEDALE ST 90049	2001-03-02	045-94-1380	1	2009-01-17		griller	16.98
28	Lindy	W	Fistal	4031 S LA SALLE AVE 90062	1915-04-17	529-86-7064	1	2008-08-14	2016-02-04	griller	19.48
27	Estrella	G	Dagher	10707 W SATICOY ST 91352	1975-02-18	740-71-6761	0	2007-04-06		manager	18.47
26	Vinnie	V	Mansour	405 W BAMBOO LANE 90012	1971-05-03	483-62-3669	2	2001-10-01		janitor	12.63
25	Stevana	A	Wen	11400 N WILBUR AVE 91326	1928-04-13	846-19-4199	1	2013-09-12	2018-09-10	salad prep	13.44
24	Kary	V	Gosden	17173 W MCKEEVER ST 91344	1903-03-02	363-40-5277	1	2000-10-02		cold prep	12.65
23	Yanaton	Z	Cormac	716 W 3RD ST 90731	1988-06-03	706-85-4548	1	2010-09-02		supervisor	19.72
22	Russell	B	Pears	4816 S 5TH AVE 90043	1969-05-05	147-14-3729	0	2003-08-14	2017-06-11	salad prep	14.78
21	Marion	L	Imoo	13345 W GARBER ST 91331	1950-06-05	272-89-5912	0	2013-04-17		cashier	12.65
20	Tiana	Z	Abelman	8369 N WILBUR AVE 91324	1908-09-09	807-21-9422	2	2002-07-09		manager	19.82
19	Lane	T	Hajjar	753 N NORTH SPRING ST 90012	1913-07-11	804-25-7412	2	2005-02-13	2018-07-08	salad prep	13.52
18	Nani	O	Kalimullin	14684 W RAYEN ST 1-30 91402	1922-12-03	106-51-7902	1	2007-03-11		salad prep	15.03
17	Olaf	L	Schitsyn	11919 W SALTAIR TER 90049	1936-02-07	490-70-1051	2	2007-03-03	2016-04-02	cold prep	14.58
16	Farand	Y	Psik	10947 W PEACH GROVE ST 91601	1965-07-02	246-59-7902	2	2010-10-15	2017-07-13	cold prep	19.62
15	Iormina	O	Muirchertach	1830 S MENLO AVE 90006	1907-10-15	873-93-2111	0	2003-06-04		cashier	16.88
14	Randolf	P	Tubelsky	2146 N STONEWOOD CT 90732	1947-07-06	613-53-0048	2	2010-05-11		cashier	15.74
13	Dawn	E	Grey	1680 S LONGWOOD AVE 90019	1994-02-10	918-87-3922	1	2015-09-10		supervisor	19.72
12	Carlena	F	Fabrikant	2211 S MAIN ST 90007	1944-06-18	730-57-6251	2	2000-12-06		supervisor	12.9
11	Jotham	Q	Holl	8611 W WENTWORTH ST 91040	1966-12-20	764-11-9331	0	2004-02-13	2019-03-03	salad prep	13.69
10	Bren	H	Balabas	20246 W JUBILEE WAY 91326	1962-11-03	322-17-4039	2	2013-03-06	2016-06-13	griller	19.86
9	Trixy	M	Zhiganov	13619 W GAIN ST 91331	1976-06-15	476-48-7077	0	2010-12-12	2017-06-10	support	18.89
8	Amata	H	Whitfield	10548 N HILLHAVEN AVE 91042	1978-01-04	735-22-1635	0	2002-01-07		griller	18.18
7	Morlee	U	Ibarra	4725 N LANKERSHIM BLVD 91602	1957-04-02	191-29-9637	0	2001-04-08		janitor	13.64
6	Nelly	Y	Tyman	3054 E FAIRMOUNT ST 90063	1914-02-12	129-70-4835	1	2001-02-08		manager	18.75
5	Dwayne	W	Mitsuya	7546 S WESTLAWN AVE 90045	1998-01-15	270-58-2652	0	2013-01-09		supervisor	15.24
4	Andonis	E	Pickering	422 N LAKE ST 90026	1954-05-01	886-13-2931	0	2006-12-04		cashier	16.16
3	Thom	I	Baiseltov	13535 W MOORPARK ST 91423	1922-04-15	654-96-0193	2	2011-04-12		support	12.35
2	Karine	R	Bumgarner	3655 S ST ANDREWS PL 90018	1958-06-14	931-17-4534	2	2014-02-20	2019-10-18	cold prep	12.54
1	Malvina	M	Datsenko	20551 W HART ST 91306	1919-10-08	701-08-6457	1	2004-04-12	2016-04-03	salad prep	13.27
0	Bert	F	Virgo	6556 N COSTELLO AVE 91401	1902-05-14	796-18-8886	2	2000-05-04	2016-09-18	salad prep	14.86

(50 rows)

Menu_Items:

```
zak=# \d menu_items
                    Table "public.menu_items"
   Column   |          Type          | Collation | Nullable | Default
------------+------------------------+-----------+----------+---------
 menu_id    | integer                |           | not null |
 name       | character varying(100) |           | not null |
 price      | real                   |           | not null |
 produce_id | integer                |           | not null |
Indexes:
    "menu_items_pkey" PRIMARY KEY, btree (menu_id)
Check constraints:
    "menu_items_price_check" CHECK (price > 0.00::double precision)
Foreign-key constraints:
    "menu_items_produce_id_fkey" FOREIGN KEY (produce_id) REFERENCES produce(produce_id)
Referenced by:
    TABLE "c_order" CONSTRAINT "c_order_menu_id_fkey" FOREIGN KEY (menu_id) REFERENCES menu_items(menu_id)
    TABLE "order_quantity" CONSTRAINT "order_quantity_menu_id_fkey" FOREIGN KEY (menu_id) REFERENCES menu_items(menu_id)
    TABLE "produce_quantity" CONSTRAINT "produce_quantity_menu_id_fkey" FOREIGN KEY (menu_id) REFERENCES menu_items(menu_id)
```

File Edit View Search Terminal Help

menu_id	name	price	produce_id
23	Chicken & Bacon Club	2.21	39
72	Homemade Cookies	2.59	39
103	Macaroni	2.84	9
52	Cobb Salad	3.11	25
50	California Chicken Salad	3.15	15
20	Albacore Tuna	3.27	11
28	Heart Smart - Ham	3.32	27
85	Sparkling Water	3.49	3
4	Chicken Fajita	3.65	26
33	The Sicilian	3.71	2
21	BLT	3.78	5
8	Grilled Cajun Turkey	3.88	27
37	The All American	4.26	23
18	Eggplant-Artichoke Panini	4.38	37
87	Fuse	5.61	36
38	The Big Dipper	5.65	29
107	Soup - Cream	5.69	34
77	Grilled Cheese	5.75	32
55	Mini Bistro	5.82	35
98	Broccoli-Pasta	6.08	37
32	Sequoia Sandwich	6.12	28
27	Ham & Baby Swiss	6.13	42
69	Frosted Fudge Brownies	6.29	11
30	Roast Beef	6.31	36
6	Classic Reuben	6.7	21
59	Southwestern Chicken Salad	7.02	22
3	Chicken Caesar	7.23	25
78	Turkey	7.32	27
96	Bowl of Waldorf Chicken Salad	7.33	15
34	Waldorf Chicken Sandwich	7.41	34
80	Salami	7.46	7
15	Turkey Bacon Melt	7.67	31
36	Double Hot Ham & Cheese	7.94	33
14	Tuna Melt	8.17	40
2	California Pastrami	8.3	12
102	Hot Sourdough Rolls	8.46	37
17	Tomato-Basil Mozzarella Panini	8.51	30
68	Carrot Cake	8.56	41
101	Creamy Cole Slaw	9.12	11
39	The Italian Stallion	9.16	1
24	Club Deluxe	9.3	21
57	Mini Garden Salad	9.42	6
91	Energy Drinks	9.89	40
26	Egg Salad	10.04	16
45	Hot Vegetarian	10.06	23
25	Cracked Pepper Turkey	10.07	33
90	Milk	10.86	1
93	Chocolate Milk	10.93	41
35	Lite Lunch	11.24	36
56	Mini Caesar Salad	11.28	41
1	California Chicken Breast Sandwich	11.66	39
41	Turkey Club	11.7	38
51	Chicken Caesar Salad	11.71	31
62	Spring Salad	12.18	35
10	Hot Roast Beef	12.4	16
44	Eggplant-Artichoke Panini	12.87	30
71	German Chocolate Cake	13.06	12
70	Garys Famous Chocolate Cake	13.48	16
43	Deluxe Grilled Cheese	13.58	15
108	Yukon Potato	14.23	26
74	Lemon Berry Cake	15.04	41
92	Coffee	15.48	9
54	Greek Salad	15.84	22
99	Chili	15.87	9
40	The New Yorker	16.3	40
19	Turkey Pesto Panini	16.31	27

:

Nutrition:

```
zak=# \d nutrition
                    Table "public.nutrition"
    Column    |         Type          | Collation | Nullable | Default
--------------+-----------------------+-----------+----------+---------
 nutrition_id | integer               |           | not null |
 n_type       | character varying(50) |           |          |
Indexes:
    "nutrition_pkey" PRIMARY KEY, btree (nutrition_id)
Referenced by:
    TABLE "produce_nutrition" CONSTRAINT "produce_nutrition_nutrition_id_fkey" FOREIGN KEY (nutrition_id) REFERENCES nutrition(nutrition_id)
```

```
zak=# SELECT * FROM nutrition ORDER BY n_type DESC;
 nutrition_id |       n_type
--------------+--------------------
           11 | vitamins
            4 | trans fat
            2 | total fat
            7 | total carbohydrates
            9 | sugars
            3 | sturated fat
            6 | sodium
            0 | serving size
           10 | protein
           14 | potassium
           13 | iron
            8 | dietary fiber
            5 | cholesterol
            1 | calories
           12 | calcium
(15 rows)
```

Order_Quantity:

```
zak=# \d order_quantity
            Table "public.order_quantity"
  Column  |  Type   | Collation | Nullable | Default
----------+---------+-----------+----------+---------
 order_id | integer |           | not null |
 menu_id  | integer |           | not null |
 price    | real    |           |          |
Check constraints:
    "order_quantity_price_check" CHECK (price >= 0.00::double precision)
Foreign-key constraints:
    "order_quantity_menu_id_fkey" FOREIGN KEY (menu_id) REFERENCES menu_items(menu_id)
    "order_quantity_order_id_fkey" FOREIGN KEY (order_id) REFERENCES c_order(order_id)
```

```
zak=# SELECT * FROM order_quantity ORDER BY price DESC;
 order_id | menu_id | price
----------+---------+--------
        1 |      36 | 209.67
       33 |       7 | 195.74
       43 |      51 | 188.33
       29 |      16 | 188.16
       41 |     105 | 186.85
       49 |      12 | 172.48
       21 |      74 | 170.25
       24 |      67 | 168.93
       11 |     101 | 162.93
       40 |      89 |  153.7
       39 |      60 | 152.41
       30 |      76 | 139.24
       33 |     108 | 134.83
       26 |      68 | 129.18
        8 |     101 | 127.75
       41 |      20 | 126.43
       28 |      61 | 125.98
       49 |       7 | 125.79
       40 |      56 |  121.4
       39 |      57 | 102.35
       40 |      16 |  94.78
       38 |      25 |  86.67
       38 |      77 |  82.66
       11 |      50 |  80.55
       10 |      66 |  78.52
        8 |      32 |  77.62
       26 |      12 |  74.13
       31 |      83 |  73.45
       42 |       8 |  71.33
        6 |      47 |  70.27
       16 |      12 |  68.75
        1 |      98 |  56.75
       23 |      47 |  55.19
       10 |      50 |  47.76
        9 |      89 |  45.18
       15 |      67 |  45.02
       35 |      65 |  36.93
       47 |      98 |  34.68
       41 |      55 |  34.53
       45 |      94 |  31.88
       12 |       7 |  28.52
       14 |      37 |  25.46
       17 |      20 |  24.14
       20 |       8 |  17.63
       34 |      92 |   15.8
       27 |      32 |  15.33
       28 |      37 |  14.86
       47 |      50 |  13.81
        6 |      66 |   6.23
       21 |      10 |   3.28
(50 rows)
```

Produce:

```
zak=# \d produce
                    Table "public.produce"
   Column    |         Type          | Collation | Nullable | Default
-------------+-----------------------+-----------+----------+---------
 prod_type   | character varying(20) |           | not null |
 quantity    | integer               |           |          |
 acquired    | date                  |           |          |
 expiration  | date                  |           |          |
 weight      | real                  |           |          |
 produce_id  | integer               |           | not null |
Indexes:
    "produce_pkey" PRIMARY KEY, btree (produce_id)
Check constraints:
    "produce_check" CHECK (expiration > acquired OR acquired IS NULL)
    "produce_quantity_check" CHECK (quantity >= 0)
    "produce_weight_check" CHECK (weight >= 0.00::double precision)
Referenced by:
    TABLE "menu_items" CONSTRAINT "menu_items_produce_id_fkey" FOREIGN KEY (produce_id) REFERENCES produce(produce_id)
    TABLE "produce_nutrition" CONSTRAINT "produce_nutrition_produce_id_fkey" FOREIGN KEY (produce_id) REFERENCES produce(produce_id)
    TABLE "produce_quantity" CONSTRAINT "produce_quantity_produce_id_fkey" FOREIGN KEY (produce_id) REFERENCES produce(produce_id)
    TABLE "supply_order" CONSTRAINT "supply_order_produce_id_fkey" FOREIGN KEY (produce_id) REFERENCES produce(produce_id)
```

```
zak=# SELECT * FROM produce ORDER BY weight DESC;
    prod_type     | quantity |  acquired  | expiration | weight | produce_id
------------------+----------+------------+------------+--------+------------
 Chicken          |       25 | 2019-06-01 | 2019-10-20 |  28.98 |          2
 Artichoke        |        5 | 2019-01-10 | 2019-12-18 |  28.78 |         26
 Wheat            |       20 | 2019-05-13 | 2019-11-02 |  27.01 |         32
 Salami           |       10 | 2019-04-19 | 2019-09-11 |  26.14 |          4
 Egg Plant        |       46 | 2019-03-08 | 2019-12-12 |  25.82 |         25
 Swiss            |       40 | 2019-01-02 | 2019-10-14 |  24.87 |         14
 Lettuce          |       43 | 2019-06-11 | 2019-12-07 |  24.42 |         17
 White            |       45 | 2019-04-02 | 2019-11-10 |  23.41 |         31
 Ham              |       14 | 2019-01-05 | 2019-10-08 |  23.27 |          0
 Pastrami         |        5 | 2019-03-04 | 2019-07-10 |  22.71 |          5
 Tomatoes         |       41 | 2019-06-06 | 2019-07-09 |  20.94 |         18
 Spinach          |        7 | 2019-03-17 | 2019-07-17 |  20.91 |         19
 Avocado          |       14 | 2019-04-03 | 2019-08-07 |  20.37 |         27
 Feta             |       17 | 2019-04-20 | 2019-08-03 |  20.24 |         15
 Oil              |       43 | 2019-02-07 | 2019-11-07 |  20.04 |         37
 Jalapenos        |       23 | 2019-02-01 | 2019-12-12 |  18.94 |         23
 Monterey         |       30 | 2019-05-12 | 2019-07-12 |  18.82 |          7
 Tuna             |        0 | 2019-05-03 | 2019-09-12 |  18.78 |          6
 American         |       27 | 2019-03-18 | 2019-10-09 |  18.55 |         10
 Pepperoccines    |        0 | 2019-05-12 | 2019-10-19 |   18.4 |         21
 Croissant        |       10 | 2019-03-20 | 2019-11-16 |  18.35 |         36
 Pesto            |       44 | 2019-03-10 | 2019-10-01 |  18.21 |         41
 Blue Cheese      |       39 | 2019-06-11 | 2019-07-08 |  17.84 |         16
 Beef             |       37 | 2019-02-19 | 2019-07-16 |  16.89 |          3
 Turkey           |       38 | 2019-04-12 | 2019-07-12 |  16.33 |          1
 Cheddar          |       19 | 2019-05-07 | 2019-12-09 |  16.13 |          9
 French           |       33 | 2019-04-13 | 2019-08-08 |  16.02 |         28
 Mayonnaise       |       47 | 2019-02-12 | 2019-10-20 |  14.84 |         39
 Bell Peppers     |        0 | 2019-05-08 | 2019-07-05 |  13.42 |         22
 Gluten Free      |        7 | 2019-04-06 | 2019-08-02 |  10.14 |         35
 Honey            |       44 | 2019-06-04 | 2019-07-11 |  10.01 |         42
 Mushrooms        |        9 | 2019-01-18 | 2019-07-09 |   8.98 |         24
 Rye              |        4 | 2019-02-06 | 2019-12-09 |   8.32 |         34
 Pepper Jack      |       42 | 2019-04-19 | 2019-07-11 |   7.19 |          8
 Vinegar          |       29 | 2019-03-18 | 2019-10-09 |   7.07 |         38
 Sprouts          |       46 | 2019-02-11 | 2019-10-20 |      7 |         20
 Provolone        |       41 | 2019-06-12 | 2019-07-18 |   6.54 |         11
 Mustard          |       24 | 2019-01-09 | 2019-07-14 |   6.11 |         40
 Sourdough        |       45 | 2019-03-14 | 2019-10-13 |    5.3 |         33
 Parmesan         |       26 | 2019-04-04 | 2019-12-13 |   4.29 |         12
 Mozzarella       |        8 | 2019-06-07 | 2019-09-14 |   3.68 |         13
 Onion Dill Roll  |       14 | 2019-03-03 | 2019-10-16 |   3.13 |         29
 Squaw            |        9 | 2019-05-12 | 2019-08-15 |   1.79 |         30
(43 rows)
```

Produce_Nutrition:

```
zak=# \d produce_nutrition
            Table "public.produce_nutrition"
    Column     |  Type   | Collation | Nullable | Default
---------------+---------+-----------+----------+---------
 produce_id    | integer |           | not null |
 nutrition_id  | integer |           | not null |
 value         | integer |           |          |
Foreign-key constraints:
    "produce_nutrition_nutrition_id_fkey" FOREIGN KEY (nutrition_id) REFERENCES nutrition(nutrition_id)
    "produce_nutrition_produce_id_fkey" FOREIGN KEY (produce_id) REFERENCES produce(produce_id)
```

```
zak=# SELECT * FROM produce_nutrition ORDER BY produce_id DESC;
 produce_id | nutrition_id | value
------------+--------------+-------
         42 |            1 |   344
         37 |           11 |   192
         33 |            6 |   310
         32 |           14 |   149
         32 |            6 |   261
         31 |            3 |   238
         31 |            2 |   282
         31 |           12 |   182
         30 |           13 |   342
         29 |            0 |   128
         28 |            7 |   245
         28 |           11 |   344
         27 |           14 |   103
         26 |            8 |   116
         26 |            9 |   452
         25 |           13 |    35
         25 |            4 |   495
         22 |            9 |   129
         21 |            7 |   208
         21 |           10 |   335
         19 |            9 |    59
         17 |           10 |    58
         17 |            2 |   113
         16 |           12 |   420
         15 |            1 |   442
         14 |            4 |   483
         13 |            7 |   111
         12 |            9 |   365
         11 |           12 |    17
         11 |           14 |   134
         10 |            8 |   442
          9 |           14 |   157
          9 |            2 |    84
          9 |           10 |   122
          7 |            7 |   458
          6 |            4 |   117
          5 |            0 |   468
          5 |           11 |   170
          5 |            4 |   195
          3 |            2 |   417
          3 |           14 |   479
          3 |           13 |    60
          2 |            1 |   185
          2 |            5 |   187
          1 |            9 |   341
          1 |           12 |   165
          1 |            3 |   130
          1 |            2 |   302
          1 |           13 |   263
          0 |            4 |   500
(50 rows)
```

Produce_Quantity:

```
zak=# \d produce_quantity
                Table "public.produce_quantity"
     Column     |  Type   | Collation | Nullable | Default
----------------+---------+-----------+----------+---------
 produce_id     | integer |           | not null |
 menu_id        | integer |           | not null |
 weight_in_item | real    |           |          |
Foreign-key constraints:
    "produce_quantity_menu_id_fkey" FOREIGN KEY (menu_id) REFERENCES menu_items(menu_id)
    "produce_quantity_produce_id_fkey" FOREIGN KEY (produce_id) REFERENCES produce(produce_id)
```

```
zak=# SELECT * FROM produce_quantity ORDER BY produce_id DESC;
 produce_id | menu_id | weight_in_item
------------+---------+----------------
         42 |      43 |           1.85
         42 |      56 |           1.73
         41 |      11 |           1.55
         39 |      49 |           3.13
         39 |      43 |           1.75
         38 |      16 |           4.14
         38 |      78 |           1.72
         38 |      23 |           2.35
         37 |       4 |           4.72
         36 |      55 |           2.56
         36 |      73 |           4.97
         36 |      77 |            2.7
         36 |      52 |           1.84
         35 |      23 |           1.28
         33 |      61 |           2.97
         31 |      15 |           3.41
         28 |      40 |           3.85
         27 |      13 |           2.55
         26 |       2 |           1.12
         24 |      52 |           3.52
         23 |      21 |           2.62
         23 |      68 |           3.86
         22 |      20 |           2.46
         22 |      82 |           4.61
         21 |      38 |           3.73
         19 |      65 |           4.05
         18 |      42 |           4.14
         17 |       3 |            4.3
         15 |      58 |           2.56
         14 |       7 |           3.04
         13 |      82 |           3.84
         12 |       7 |           1.58
         11 |      49 |           1.53
         10 |      17 |           4.45
          9 |       3 |           3.83
          8 |      38 |           1.51
          8 |      56 |           1.92
          7 |     104 |           1.21
          7 |      16 |           3.04
          7 |      53 |           1.42
          5 |      77 |           1.18
          5 |      67 |           4.93
          4 |      61 |           1.44
          3 |      33 |           1.95
          3 |      93 |           4.08
          3 |      87 |           3.39
          1 |      44 |           2.77
          0 |      28 |           2.41
          0 |      73 |           4.38
          0 |      83 |           2.55
(50 rows)
```

Store:

```
zak=# \d store
                    Table "public.store"
  Column   |          Type          | Collation | Nullable | Default
-----------+------------------------+-----------+----------+---------
 store_num | integer                |           | not null |
 location  | character varying(100) |           |          |
Indexes:
    "store_pkey" PRIMARY KEY, btree (store_num)
Referenced by:
    TABLE "clock_in" CONSTRAINT "clock_in_store_number_fkey" FOREIGN KEY (store_number) REFERENCES store(store_num)
    TABLE "employee" CONSTRAINT "employee_store_number_fkey" FOREIGN KEY (store_number) REFERENCES store(store_num)
    TABLE "supply_order" CONSTRAINT "supply_order_store_num_fkey" FOREIGN KEY (store_num) REFERENCES store(store_num)
```

```
zak=# SELECT * FROM store ORDER BY location DESC;
 store_num |                          location
-----------+-------------------------------------------------------------
         1 | 9500 Ming Ave. Bakersfield, CA, 93311
         2 | 9160 Rosedale Hwy. Suite #100 Bakersfield, CA, 93312
         0 | 1231 18th St. Bakersfield, CA, 93301
(3 rows)
```

Supplier:

```
zak=# \d supplier
                   Table "public.supplier"
   Column    |          Type          | Collation | Nullable | Default
-------------+------------------------+-----------+----------+---------
 supplier_id | integer                |           | not null |
 name        | character varying(50)  |           | not null |
Indexes:
    "supplier_pkey" PRIMARY KEY, btree (supplier_id)
Referenced by:
    TABLE "supply_order" CONSTRAINT "supply_order_supplier_id_fkey" FOREIGN KEY (supplier_id) REFERENCES supplier(supplier_id)
```

```
zak=# SELECT * FROM supplier ORDER BY name DESC;
 supplier_id |              name
-------------+--------------------------------
           1 | AWESOME QUALITY SUPPLIER CO.
(1 row)
```

Supply_Order:

```
zak=# \d supply_order
              Table "public.supply_order"
   Column    |  Type   | Collation | Nullable | Default
-------------+---------+-----------+----------+---------
 supplier_id | integer |           | not null |
 price       | real    |           |          |
 produce_id  | integer |           | not null |
 order_date  | date    |           | not null |
 store_num   | integer |           | not null |
 weight      | real    |           |          |
Check constraints:
    "supply_order_price_check" CHECK (price > 0.00::double precision)
    "supply_order_weight_check" CHECK (weight > 0.00::double precision)
Foreign-key constraints:
    "supply_order_produce_id_fkey" FOREIGN KEY (produce_id) REFERENCES produce(produce_id)
    "supply_order_store_num_fkey" FOREIGN KEY (store_num) REFERENCES store(store_num)
    "supply_order_supplier_id_fkey" FOREIGN KEY (supplier_id) REFERENCES supplier(supplier_id)
```

```
zak=# SELECT * FROM supply_order ORDER BY store_num DESC;
 supplier_id | price  | produce_id | order_date | store_num | weight
-------------+--------+------------+------------+-----------+--------
           1 | 297.33 |         18 | 2014-01-08 |         2 | 447.98
           1 | 435.62 |         12 | 2001-03-01 |         2 | 423.19
           1 | 862.91 |         39 | 2014-11-17 |         2 | 171.73
           1 | 951.47 |         24 | 2013-09-16 |         2 | 229.74
           1 | 835.71 |         33 | 2011-06-04 |         2 | 400.47
           1 | 461.34 |         18 | 2017-04-14 |         2 | 364.12
           1 | 295.25 |          2 | 2012-11-12 |         2 | 461.75
           1 | 453.67 |         27 | 2000-05-06 |         2 | 487.61
           1 | 248.92 |          5 | 2018-06-20 |         2 |  109.1
           1 | 531.48 |          5 | 2015-02-02 |         2 | 327.94
           1 | 865.52 |         18 | 2013-03-02 |         2 | 115.65
           1 | 895.06 |         39 | 2013-10-14 |         2 | 166.08
           1 | 804.95 |         24 | 2003-06-13 |         2 | 156.26
           1 | 100.26 |         25 | 2007-07-03 |         2 | 476.72
           1 | 606.46 |         29 | 2005-06-04 |         2 | 123.19
           1 | 914.24 |         11 | 2010-08-16 |         2 | 492.53
           1 | 523.35 |          0 | 2004-12-20 |         2 | 218.78
           1 | 510.82 |         23 | 2006-02-17 |         2 | 254.78
           1 | 140.43 |         16 | 2005-11-06 |         1 | 212.44
           1 | 703.56 |         24 | 2001-09-09 |         1 | 228.63
           1 | 776.46 |         12 | 2019-01-10 |         1 | 134.06
           1 | 929.46 |         31 | 2012-08-19 |         1 | 364.93
           1 | 816.94 |         37 | 2007-07-14 |         1 | 340.54
           1 | 596.65 |         25 | 2005-09-02 |         1 | 216.17
           1 | 428.59 |         16 | 2001-08-14 |         1 | 376.77
           1 | 782.73 |         39 | 2014-01-06 |         1 | 418.12
           1 | 305.48 |         41 | 2011-04-05 |         1 | 462.73
           1 | 111.58 |         14 | 2003-11-02 |         1 | 467.86
           1 | 853.38 |         36 | 2007-07-04 |         1 | 238.74
           1 | 834.54 |         24 | 2008-11-19 |         1 | 178.06
           1 | 775.04 |         22 | 2016-11-18 |         1 | 185.25
           1 | 706.01 |         31 | 2009-10-14 |         1 | 337.62
           1 | 481.96 |         26 | 2001-11-01 |         0 | 341.63
           1 | 536.49 |         25 | 2010-12-03 |         0 | 403.18
           1 | 724.52 |          7 | 2019-08-13 |         0 | 190.53
           1 | 451.83 |         14 | 2012-12-11 |         0 | 400.43
           1 | 226.09 |         37 | 2013-11-17 |         0 | 257.14
           1 | 547.42 |          5 | 2015-05-10 |         0 | 109.21
           1 | 623.72 |          6 | 2005-12-06 |         0 | 432.73
           1 | 167.14 |         39 | 2007-02-01 |         0 | 383.14
           1 | 131.59 |          3 | 2002-09-04 |         0 | 205.05
           1 | 157.75 |         12 | 2001-12-08 |         0 |  105.7
           1 | 763.27 |          2 | 2016-11-14 |         0 | 235.62
           1 |  357.5 |         38 | 2002-03-10 |         0 | 326.64
           1 | 911.95 |         26 | 2004-11-02 |         0 | 147.16
           1 | 914.89 |         33 | 2019-07-04 |         0 | 169.41
           1 |  728.1 |         23 | 2000-12-17 |         0 | 436.62
           1 | 742.21 |          5 | 2017-03-09 |         0 | 236.91
           1 | 389.41 |         38 | 2006-07-06 |         0 | 100.57
           1 | 568.43 |         33 | 2003-04-04 |         0 | 351.62
(50 rows)
```

1.5 Queries

1. Name the menu items that would have less than 1000 calories

   SELECT * FROM menu_item NATURAL JOIN produce NATURAL JOIN produce_nutrition NATURAL JOIN nutrition

   WHERE n_type =’calories’ AND value < 1000;

```
zak=# SELECT * FROM menu_items NATURAL JOIN produce NATURAL JOIN produce_nutrition NATURAL JOIN nutrition WHERE
zak-# n_type = 'calories' AND value < 1000;
 nutrition_id | produce_id | menu_id |            name              | price | prod_type | quantity |  acquired  | expiration | weight | value |  n_type
--------------+------------+---------+------------------------------+-------+-----------+----------+------------+------------+--------+-------+----------
            1 |          2 |       7 | French Dip                   | 22.44 | Chicken   |       25 | 2019-06-01 | 2019-10-20 |  28.98 |   185 | calories
            1 |         42 |      22 | Cajun Turkey                 |  17.8 | Honey     |       44 | 2019-06-04 | 2019-07-11 |  10.01 |   344 | calories
            1 |         42 |      27 | Ham & Baby Swiss             |  6.13 | Honey     |       44 | 2019-06-04 | 2019-07-11 |  10.01 |   344 | calories
            1 |          2 |      33 | The Sicilian                 |  3.71 | Chicken   |       25 | 2019-06-01 | 2019-10-20 |  28.98 |   185 | calories
            1 |          2 |      42 | Cold Veggie & Cheese         | 23.87 | Chicken   |       25 | 2019-06-01 | 2019-10-20 |  28.98 |   185 | calories
            1 |         15 |      43 | Deluxe Grilled Cheese        | 13.58 | Feta      |       17 | 2019-04-20 | 2019-08-03 |  20.24 |   442 | calories
            1 |         15 |      47 | Antipasta Salad              | 20.17 | Feta      |       17 | 2019-04-20 | 2019-08-03 |  20.24 |   442 | calories
            1 |         15 |      50 | California Chicken Salad     |  3.15 | Feta      |       17 | 2019-04-20 | 2019-08-03 |  20.24 |   442 | calories
            1 |         42 |      60 | Spinach Kale                 |  29.3 | Honey     |       44 | 2019-06-04 | 2019-07-11 |  10.01 |   344 | calories
            1 |         15 |      96 | Bowl of Waldorf Chicken Salad |  7.33 | Feta      |       17 | 2019-04-20 | 2019-08-03 |  20.24 |   442 | calories
(10 rows)
```

2. Name the Menu Items that would be under $10

   SELECT * FROM menu_items

   WHERE price < 10

   ORDER BY price;

```
zak=# SELECT * FROM menu_items WHERE price < 10 ORDER BY price;
 menu_id |             name              | price | produce_id
---------+-------------------------------+-------+------------
      23 | Chicken & Bacon Club          |  2.21 |         39
      72 | Homemade Cookies              |  2.59 |         39
     103 | Macaroni                      |  2.84 |          9
      52 | Cobb Salad                    |  3.11 |         25
      50 | California Chicken Salad      |  3.15 |         15
      20 | Albacore Tuna                 |  3.27 |         11
      28 | Heart Smart - Ham             |  3.32 |         27
      85 | Sparkling Water               |  3.49 |          3
       4 | Chicken Fajita                |  3.65 |         26
      33 | The Sicilian                  |  3.71 |          2
      21 | BLT                           |  3.78 |          5
       8 | Grilled Cajun Turkey          |  3.88 |         27
      37 | The All American              |  4.26 |         23
      18 | Eggplant-Artichoke Panini     |  4.38 |         37
      87 | Fuse                          |  5.61 |         36
      38 | The Big Dipper                |  5.65 |         29
     107 | Soup - Cream                  |  5.69 |         34
      77 | Grilled Cheese                |  5.75 |         32
      55 | Mini Bistro                   |  5.82 |         35
      98 | Broccoli-Pasta                |  6.08 |         37
      32 | Sequoia Sandwich              |  6.12 |         28
      27 | Ham & Baby Swiss              |  6.13 |         42
      69 | Frosted Fudge Brownies        |  6.29 |         11
      30 | Roast Beef                    |  6.31 |         36
       6 | Classic Reuben                |   6.7 |         21
      59 | Southwestern Chicken Salad    |  7.02 |         22
       3 | Chicken Caesar                |  7.23 |         25
      78 | Turkey                        |  7.32 |         27
      96 | Bowl of Waldorf Chicken Salad |  7.33 |         15
      34 | Waldorf Chicken Sandwich      |  7.41 |         34
      80 | Salami                        |  7.46 |          7
      15 | Turkey Bacon Melt             |  7.67 |         31
      36 | Double Hot Ham & Cheese       |  7.94 |         33
      14 | Tuna Melt                     |  8.17 |         40
       2 | California Pastrami           |   8.3 |         12
     102 | Hot Sourdough Rolls           |  8.46 |         37
      17 | Tomato-Basil Mozzarella Panini |  8.51 |         30
      68 | Carrot Cake                   |  8.56 |         41
     101 | Creamy Cole Slaw              |  9.12 |         11
      39 | The Italian Stallion          |  9.16 |          1
      24 | Club Deluxe                   |   9.3 |         21
      57 | Mini Garden Salad             |  9.42 |          6
      91 | Energy Drinks                 |  9.89 |         40
(43 rows)
```

3. List all employees who have taken an order from the same customer more than once in the same week.

```
SELECT * FROM Employee

WHERE Customer_number = Customer_number, Employee_ID =
Employee_ID

ORDER BY Date;
```

4. List the most expensive produce items that were ordered in the past month.

```
SELECT * FROM Produce, Menu_Items, Orders

WHERE price > 10;
```

Results:

//

//

5. List the store that profits the most.

```
SELECT * FROM Store, Supply_quantity

WHERE Store_Number = Supply_Price;
```

Results:

//

//

6. List the customers who have never ordered the same item.

```
SELECT * FROM (customer NATURAL JOIN c_order) c1, (customer
NATURAL JOIN c_order) c2
WHERE c1.name = c2.name AND c1.menu_id != c2.menu_id
ORDER BY c1.customer_num;
```

Results:

customer_num	phone_number	name	employee_id	menu_id	order_id	order_date
1	(329)275-2915	Noelle Jefferies	4	10	5	2011-06-07
1	(329)275-2915	Noelle Jefferies	17	74	1	2001-12-17
2	(740)521-4646	Petronille Than	38	43	29	2000-03-10
2	(740)521-4646	Petronille Than	39	91	48	2010-01-13
5	(141)043-6347	Mariette Zogby	43	73	4	2000-09-15
5	(141)043-6347	Mariette Zogby	48	55	22	2019-10-04
7	(253)809-5095	Kissee Zogby	13	28	32	2002-02-11
7	(253)809-5095	Kissee Zogby	44	3	24	2007-11-06
10	(698)854-9661	Miran Mansour	46	27	30	2015-09-08
10	(698)854-9661	Miran Mansour	46	27	30	2015-09-08
10	(698)854-9661	Miran Mansour	26	38	18	2019-08-08
10	(698)854-9661	Miran Mansour	26	38	18	2019-08-08
10	(698)854-9661	Miran Mansour	17	47	20	2019-02-08
10	(698)854-9661	Miran Mansour	17	47	20	2019-02-08
12	(413)031-3493	Eugenie Golovatsky	29	85	6	2010-05-19
12	(413)031-3493	Eugenie Golovatsky	23	105	42	2011-09-03
16	(294)730-0495	Kincaid Zheravin	42	32	31	2018-08-15
16	(294)730-0495	Kincaid Zheravin	22	94	7	2017-04-06
18	(465)553-4329	Amalee Novosiltsev	32	53	45	2005-01-12
18	(465)553-4329	Amalee Novosiltsev	7	70	12	2016-10-03
19	(854)308-0224	Gerrard Liu	29	93	10	2018-01-12
19	(854)308-0224	Gerrard Liu	12	15	39	2006-05-20
21	(963)040-2936	Darth Etherton	10	4	2	2013-06-20
21	(963)040-2936	Darth Etherton	21	53	28	2001-05-14
22	(647)028-9482	Rhianon Mcghee	21	57	41	2004-11-18
22	(647)028-9482	Rhianon Mcghee	15	56	14	2014-03-03
33	(280)515-3517	Ester Toichkin	21	68	26	2006-09-09
33	(280)515-3517	Ester Toichkin	19	56	47	2006-10-05
35	(735)614-5092	Hewett Demichev	34	53	40	2000-03-09
35	(735)614-5092	Hewett Demichev	38	5	43	2017-05-11
41	(140)654-2475	Hortense Lachapelle	8	2	13	2003-11-13
41	(140)654-2475	Hortense Lachapelle	15	88	35	2015-08-18
48	(321)733-8484	Paulina Dantsig	16	97	25	2009-03-14
48	(321)733-8484	Paulina Dantsig	24	65	11	2005-01-20

(34 rows)

7. List the customers who have ordered the same order at least two times.

```
zak=# SELECT * FROM (customer NATURAL JOIN c_order) c1, (customer NATURAL JOIN c_order) c2
WHERE c1.name = c2.name AND c1.order_date != c2.order_date
ORDER BY c1.customer_num;
 customer_num | phone_number  |        name         | employee_id | menu_id | order_id | order_date |
--------------+---------------+---------------------+-------------+---------+----------+------------+-
            1 | (329)275-2915 | Noelle Jefferies    |           4 |      10 |        5 | 2011-06-07 |
            1 | (329)275-2915 | Noelle Jefferies    |          17 |      74 |        1 | 2001-12-17 |
            2 | (740)521-4646 | Petronille Than     |          38 |      43 |       29 | 2000-03-10 |
            2 | (740)521-4646 | Petronille Than     |          39 |      91 |       48 | 2010-01-13 |
            5 | (141)043-6347 | Mariette Zogby      |          43 |      73 |        4 | 2000-09-15 |
            5 | (141)043-6347 | Mariette Zogby      |          48 |      55 |       22 | 2019-10-04 |
            7 | (253)809-5095 | Kissee Zogby        |          13 |      28 |       32 | 2002-02-11 |
            7 | (253)809-5095 | Kissee Zogby        |          44 |       3 |       24 | 2007-11-06 |
           10 | (698)854-9661 | Miran Mansour       |          46 |      27 |       30 | 2015-09-08 |
           10 | (698)854-9661 | Miran Mansour       |          46 |      27 |       30 | 2015-09-08 |
           10 | (698)854-9661 | Miran Mansour       |          26 |      38 |       18 | 2019-08-08 |
           10 | (698)854-9661 | Miran Mansour       |          26 |      38 |       18 | 2019-08-08 |
           10 | (698)854-9661 | Miran Mansour       |          17 |      47 |       20 | 2019-02-08 |
           10 | (698)854-9661 | Miran Mansour       |          17 |      47 |       20 | 2019-02-08 |
           12 | (413)031-3493 | Eugenie Golovatsky  |          29 |      85 |        6 | 2010-05-19 |
           12 | (413)031-3493 | Eugenie Golovatsky  |          23 |     105 |       42 | 2011-09-03 |
           16 | (294)730-0495 | Kincaid Zheravin    |          42 |      32 |       31 | 2018-08-15 |
           16 | (294)730-0495 | Kincaid Zheravin    |          22 |      94 |        7 | 2017-04-06 |
           18 | (465)553-4329 | Amalee Novosiltsev  |          32 |      53 |       45 | 2005-01-12 |
           18 | (465)553-4329 | Amalee Novosiltsev  |           7 |      70 |       12 | 2016-10-03 |
           19 | (854)308-0224 | Gerrard Liu         |          29 |      93 |       10 | 2018-01-12 |
           19 | (854)308-0224 | Gerrard Liu         |          12 |      15 |       39 | 2006-05-20 |
           21 | (963)040-2936 | Darth Etherton      |          10 |       4 |        2 | 2013-06-20 |
           21 | (963)040-2936 | Darth Etherton      |          21 |      53 |       28 | 2001-05-14 |
           22 | (647)028-9482 | Rhianon Mcghee      |          21 |      57 |       41 | 2004-11-18 |
           22 | (647)028-9482 | Rhianon Mcghee      |          15 |      56 |       14 | 2014-03-03 |
           33 | (280)515-3517 | Ester Toichkin      |          21 |      68 |       26 | 2006-09-09 |
           33 | (280)515-3517 | Ester Toichkin      |          19 |      56 |       47 | 2006-10-05 |
           35 | (735)614-5092 | Hewett Demichev     |          34 |      53 |       40 | 2000-03-09 |
           35 | (735)614-5092 | Hewett Demichev     |          38 |       5 |       43 | 2017-05-11 |
           41 | (140)654-2475 | Hortense Lachapelle |           8 |       2 |       13 | 2003-11-13 |
           41 | (140)654-2475 | Hortense Lachapelle |          15 |      88 |       35 | 2015-08-18 |
           48 | (321)733-8484 | Paulina Dantsig     |          16 |      97 |       25 | 2009-03-14 |
           48 | (321)733-8484 | Paulina Dantsig     |          24 |      65 |       11 | 2005-01-20 |
```

8. List the employees who have worked more than 40 hours in the past week or worked more than 8 hours in a single day.

```
zak=# SELECT * FROM employee NATURAL JOIN clock_in WHERE extract(hour from time_out) - extract(hour from time_in) > 8;
 employee_id | store_number |  fname   | minit |   lname    |              address              |    dob     |     ssn     | start_date |  end_date  |  position  | salary |       time_in       |      time_out       | earnings
-------------+--------------+----------+-------+------------+-----------------------------------+------------+-------------+------------+------------+------------+--------+---------------------+---------------------+----------
          41 |            2 | Camilla  | P     | Gaber      | 127 E AVENUE 45 90031             | 1904-06-08 | 528-53-0376 | 2003-02-02 |            | salad prep |  13.96 | 2002-04-19 11:45:57 | 2002-04-19 21:24:51 |    17.69
          49 |            1 | Guillema | A     | Horowitz   | 3517 S CHESAPEAKE AVE 90016       | 1967-05-20 | 558-32-4021 | 2009-02-10 | 2016-02-14 | cold prep  |  15.83 | 2004-03-10 10:16:31 | 2004-03-10 19:22:44 |    10.32
          35 |            0 | Mike     | G     | Kerridge   | 23155 W OXNARD ST 91367           | 1945-03-05 | 996-40-4033 | 2009-10-07 |            | cold prep  |   19.5 | 2005-01-09 10:42:53 | 2005-01-09 21:52:44 |    48.43
          48 |            1 | Joyan    | S     | Oppenheimer | 11603 W HAYNES ST 91606          | 1927-08-13 | 082-90-7408 | 2011-05-20 | 2018-12-08 | support    |  15.18 | 2019-08-10 09:09:36 | 2019-08-10 22:14:42 |    32.35
           3 |            2 | Thom     | I     | Baiseitov  | 13535 W MOORPARK ST 91423         | 1922-04-15 | 654-96-0193 | 2011-04-12 |            | support    |  12.35 | 2010-06-01 11:48:41 | 2010-06-01 22:49:23 |    87.31
          13 |            1 | Dawn     | E     | Grey       | 1680 S LONGWOOD AVE 90019         | 1994-02-10 | 918-87-3922 | 2015-09-10 |            | supervisor |  19.72 | 2016-03-23 00:09:03 | 2016-03-23 21:30:24 |    33.73
          47 |            1 | Jami     | U     | Turubanov  | 2644 N CAHUENGA BLVD EAST 90068   | 1923-01-16 | 038-27-1406 | 2001-03-20 |            | griller    |  18.26 | 2017-06-22 05:00:56 | 2017-06-22 23:57:25 |    55.65
          35 |            0 | Mike     | G     | Kerridge   | 23155 W OXNARD ST 91367           | 1945-03-05 | 996-40-4033 | 2009-10-07 |            | cold prep  |   19.5 | 2016-04-18 05:37:16 | 2016-04-18 17:42:52 |    58.43
(8 rows)
```

9. List employees who have worked in every location.

```
SELECT * FROM Employee, Store

WHERE address = location;
```

Results:

//

//

10. List the store that has purchased the most produce in the past month.

```
SELECT * FROM Produce, Menu_Items, Orders, Store
WHERE Quantity > 500
ORDER BY Quantity ASC;
```

```
Results:
//
//
```

1.6 Data Loader

In order for the data-loader to work, it required an account on the Delphi server. Furthermore, it was written with Oracle in mind. So in lue of these troublesome facts, I wrote a new ‘data-loader’ program using python to fill the insert SQL files to be used on the postgresql client. This was done to mimic quite similarly to the Hotel database postgresql example posted on the web page (https://www.cs.csub.edu/~hwang/CS342/)

File Edit View Search Terminal Help

```
def make_store():
    location = ['1231 18th St. Bakersfield, CA, 93301', '9500 Ming Ave. Bakersfield, CA, 93311',
            '9160 Rosedale Hwy. Suite #100 Bakersfield, CA, 93312']
    sqlFile = "ins_store.sql"
    sqlData = open(sqlFile, 'w')
    for count in range(3):
        #store_num
        sqlData.write('INSERT INTO store VALUES(' + str(count) + ',')
        #location
        sqlData.write('\'' + location[count] + '\');\n')
    sqlData.write('commit;')
    sqlData.close()
make_store()

def make_employee():
    sqlFile = "ins_employee.sql"
    sqlData = open(sqlFile, 'w')
    for count in range(50):
        #employee_id
        sqlData.write('INSERT INTO employee VALUES(' + str(count) + ',')
        #name
        f = random.randint(0,len(fname))
        l = random.randint(0,len(lname))
        sqlData.write('\'' + fname[f] + '\',\'' + random.choice(string.ascii_uppercase) + '\',\'' + lname[l] + '\',')
        #address
        a = random.randint(1,len(address))
        sqlData.write("'" + address[a] + "',")
        #dob
        sqlData.write('to_date(\'' + str(random.randint(1,12))  + '/' + str(random.randint(1,20)) + '/'
                + str(random.randint(1900, 2001)) + '\', ' + '\'mm/dd/yyyy\'),')
        #ssn
        sqlData.write("'" + str(random.randint(0,9)) + str(random.randint(0,9)) + str(random.randint(0,9)) + '-'
                + str(random.randint(0,9)) + str(random.randint(0,9)) + '-' + str(random.randint(0,9))
                + str(random.randint(0,9)) + str(random.randint(0,9)) + str(random.randint(0,9)) + "',")
        #store_num
        sqlData.write(str(random.randint(0,2)))
        #start_date
        sqlData.write(',to_date(\'' + str(random.randint(1,12))  + '/' + str(random.randint(1,20)) + '/'
                + str(random.randint(2000, 2015)) + '\', ' + '\'mm/dd/yyyy\'),')
        #end_date
        if(random.randint(0,1) == 0):
            sqlData.write('to_date(\'' + str(random.randint(1,12))  + '/' + str(random.randint(1,20)) + '/'
                    + str(random.randint(2016, 2019)) + '\', ' + '\'mm/dd/yyyy\'),')
        else:
            sqlData.write('NULL,')
        #position
        position = ['janitor', 'salad prep', 'griller', 'cold prep', 'support', 'cashier', 'supervisor', 'manager']
        sqlData.write('\'' + random.choice(position) + '\',')
        #salary
        sal = '%.2f' % random.uniform(12.00, 20.00)
        sqlData.write(sal + ');\n')
    sqlData.write('commit;')
    sqlData.close()
make_employee()

def make_clockin():
    sqlFile = "ins_clock_in.sql"
    sqlData = open(sqlFile, 'w')
    for count in range(50):
        #employee_id
        sqlData.write('INSERT INTO clock_in VALUES(' + str(random.randint(0,49)) + ',')
        #time_in
        year = random.randint(2000,2019)
        month = random.randint(1,12)
        day = random.randint(1,25)
        hour = random.randint(0,22)
        minute = random.randint(0,59)
```

```
File  Edit  View  Search  Terminal  Help
        second = random.randint(0,59)
        sqlData.write('TIMESTAMP \'' + str(year) + '-' + str(month) + '-' + str(day) + ' ' + str(hour)
                      + ':' + str(minute) + ':' + str(second) +'\',')
        #time_out
        hour2 = random.randint(hour+1,23)
        minute2 = random.randint(0,59)
        second2 = random.randint(0,59)
        sqlData.write('TIMESTAMP \'' + str(year) + '-' + str(month) + '-' + str(day) + ' ' + str(hour2)
                      + ':' + str(minute2) + ':' + str(second2) +'\',')
        #earnings
        money = '%.2f' % random.uniform(1,100)
        sqlData.write(money + ',')
        #store_number
        sqlData.write(str(random.randint(0,2)) + ');\n')
    sqlData.write('commit;')
    sqlData.close()
make_clockin();

def make_produce():
    produce = ['Ham', 'Turkey', 'Chicken', 'Beef', 'Salami', 'Pastrami', 'Tuna', 'Monterey', 'Pepper Jack', 'Cheddar', 'American',
               'Provolone', 'Parmesan', 'Mozzarella', 'Swiss', 'Feta', 'Blue Cheese', 'Lettuce', 'Tomatoes', 'Spinach', 'Sprouts',
               'Pepperoccines', 'Bell Peppers', 'Jalapenos', 'Mushrooms', 'Egg Plant', 'Artichoke', 'Avocado', 'French',
               'Onion Dill Roll', 'Squaw', 'White', 'Wheat', 'Sourdough', 'Rye', 'Gluten Free', 'Croissant', 'Oil', 'Vinegar',
               'Mayonnaise', 'Mustard', 'Pesto', 'Honey']
    sqlFile = "ins_produce.sql"
    sqlData = open(sqlFile, 'w')
    for count in range(len(produce)):
        #prod_type
        sqlData.write('INSERT INTO produce VALUES(\'' + produce[count] + '\',')
        #quanity
        sqlData.write(str(random.randint(0,50)) +',')
        #aquired
        sqlData.write('to_date(\'' + str(random.randint(1,6))  + '/' + str(random.randint(1,20)) + '/'
                + '2019\', ' + '\'mm/dd/yyyy\'),')
        #expiration
        sqlData.write('to_date(\'' + str(random.randint(7,12))  + '/' + str(random.randint(1,20)) + '/'
                + '2019\', ' + '\'mm/dd/yyyy\'),')
        #weight
        weight = '%.2f' % random.uniform(0,30)
        sqlData.write(weight + ',')
        #produce_id
        sqlData.write(str(count) + ');\n')
    sqlData.write('commit;')
    sqlData.close()
make_produce()

def make_menuitems():
    items = ['BBQ Chicken', 'California Chicken Breast Sandwich', 'California Pastrami', 'Chicken Caesar', 'Chicken Fajita',
             'Chicken Italiano', 'Classic Reuben', 'French Dip', 'Grilled Cajun Turkey', 'Hot Ham & Cheese', 'Hot Roast Beef',
             'Monterey Chicken Club', 'Philly Cheesesteak', 'Southwestern Chicken', 'Tuna Melt', 'Turkey Bacon Melt', 'Turkey Dip',
             'Tomato-Basil Mozzarella Panini', 'Eggplant-Artichoke Panini', 'Turkey Pesto Panini', 'Albacore Tuna', 'BLT', 'Cajun Turkey',
             'Chicken & Bacon Club', 'Club Deluxe', 'Cracked Pepper Turkey', 'Egg Salad', 'Ham & Baby Swiss', 'Heart Smart - Ham',
             'Heart Smart - Turkey', 'Roast Beef', 'Roast Turkey Breast', 'Sequoia Sandwich', 'The Sicilian', 'Waldorf Chicken Sandwich',
             'Lite Lunch', 'Double Hot Ham & Cheese', 'The All American', 'The Big Dipper', 'The Italian Stallion', 'The New Yorker',
             'Turkey Club', 'Cold Veggie & Cheese', 'Deluxe Grilled Cheese', 'Eggplant-Artichoke Panini', 'Hot Vegetarian',
             'Tomato-Basil Mozzarella Panini', 'Antipasta Salad', 'Bistro Salad', 'Caesar Salad', 'California Chicken Salad',
             'Chicken Caesar Salad', 'Cobb Salad', 'Garden Salad', 'Greek Salad', 'Mini Bistro', 'Mini Caesar Salad', 'Mini Garden Salad',
             'Sesame Chicken Salad', 'Southwestern Chicken Salad', 'Spinach Kale', 'Slaw Salad', 'Spring Salad', 'Taco Salad',
             'Tuna Salad', 'Waldorf Chicken Salad', 'Banana Pudding', 'Butter Cake', 'Carrot Cake', 'Frosted Fudge Brownies',
             'Garys Famous Chocolate Cake', 'German Chocolate Cake', 'Homemade Cookies', 'Homemade New York Cheesecake', 'Lemon Berry Cake',
             'New York Cheesecake w/strawberry sauce', 'Rice Krispy Treat', 'Grilled Cheese', 'Turkey', 'Ham', 'Salami', 'Bottled Soda',
             'Bottled Water', 'Hot Tea', 'Regular Fountain Drink', 'Sparkling Water', 'Sobe', 'Fuse', 'Specialty Sodas', 'Juices', 'Milk',
             'Energy Drinks', 'Coffee', 'Chocolate Milk', 'Baked Potatoes', 'Bowl of Tuna', 'Bowl of Waldorf Chicken Salad',
             'Broccoli & Raisin', 'Broccoli-Pasta', 'Chili', 'Chili & Cornbread Special', 'Creamy Cole Slaw', 'Hot Sourdough Rolls',
             'Macaroni', 'Potato Chips', 'Quinoa', 'Soup - Broth', 'Soup - Cream', 'Yukon Potato']
    sqlFile = "ins_menu.sql"
    sqlData = open(sqlFile, 'w')
    for count in range(len(items)):
```

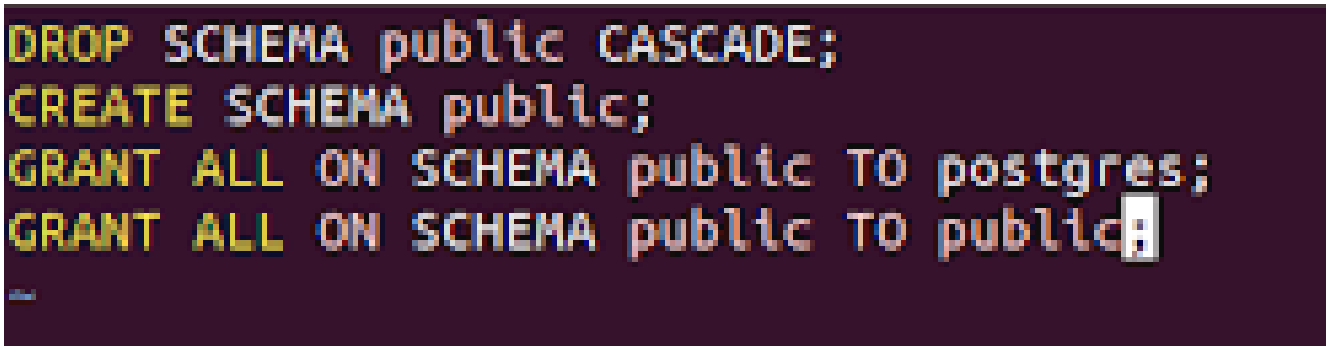
```
File  Edit  View  Search  Terminal  Help
        #menu_id
        sqlData.write('INSERT INTO menu_items VALUES(' + str(count) + ',')
        #name
        sqlData.write("'" + items[count] + "',")
        #price
        price = '%.2f' % random.uniform(2,30)
        sqlData.write(price + ',')
        #produce_id
        sqlData.write(str(random.randint(0,42)) + ');\n')
    sqlData.write('commit;')
    sqlData.close()
make_menuitems()

def make_order():
    sqlFile = "ins_order.sql"
    sqlData = open(sqlFile, 'w')
    for count in range(50):
        #employee_id
        sqlData.write('INSERT INTO c_order VALUES(' + str(random.randint(0,49)) + ',')
        #customer_num
        sqlData.write(str(random.randint(0,49)) + ',')
        #menu_id
        sqlData.write(str(random.randint(0,108)) + ',')
        #order_id
        sqlData.write(str(count) + ',')
        #date
        sqlData.write('to_date(\'' + str(random.randint(1,12))  + '/' + str(random.randint(1,20)) + '/'
                + str(random.randint(2000, 2019)) + '\', ' + '\'mm/dd/yyyy\'));\n')
    sqlData.write('commit;')
    sqlData.close()
make_order()

def make_prodquan():
    sqlFile = "ins_prodquan.sql"
    sqlData = open(sqlFile, 'w')
    for count in range(50):
        #produce_id
        sqlData.write('INSERT INTO produce_quantity VALUES(' + str(random.randint(0,42)) + ',')
        #menu_id
        sqlData.write(str(random.randint(0,108)) + ',')
        #weight
        weight = '%.2f' % random.uniform(1,5)
        sqlData.write(weight + ');\n')
    sqlData.write('commit;')
    sqlData.close()
make_prodquan()

def make_orderquan():
    sqlFile = "ins_orderquan.sql"
    sqlData = open(sqlFile, 'w')
    for count in range(50):
        #order_id
        sqlData.write('INSERT INTO order_quantity VALUES(' + str(random.randint(0,49)) + ',')
        #menu_id
        sqlData.write(str(random.randint(0,108)) + ',')
        #price
        price = '%.2f' % random.uniform(1,220)
        sqlData.write(price + ');\n')
    sqlData.write('commit;')
    sqlData.close()
make_orderquan()

def make_supquan():
    sqlFile = "ins_supquan.sql"
    sqlData = open(sqlFile, 'w')
    for count in range(50):
        #supplier_id
        sqlData.write('INSERT INTO supply_order VALUES(1,')
```

```
File  Edit  View  Search  Terminal  Help
make_order()

def make_prodquan():
    sqlFile = "ins_prodquan.sql"
    sqlData = open(sqlFile, 'w')
    for count in range(50):
        #produce_id
        sqlData.write('INSERT INTO produce_quantity VALUES(' + str(random.randint(0,42)) + ',')
        #menu_id
        sqlData.write(str(random.randint(0,108)) + ',')
        #weight
        weight = '%.2f' % random.uniform(1,5)
        sqlData.write(weight + ');\n')
    sqlData.write('commit;')
    sqlData.close()
make_prodquan()

def make_orderquan():
    sqlFile = "ins_orderquan.sql"
    sqlData = open(sqlFile, 'w')
    for count in range(50):
        #order_id
        sqlData.write('INSERT INTO order_quantity VALUES(' + str(random.randint(0,49)) + ',')
        #menu_id
        sqlData.write(str(random.randint(0,108)) + ',')
        #price
        price = '%.2f' % random.uniform(1,220)
        sqlData.write(price + ');\n')
    sqlData.write('commit;')
    sqlData.close()
make_orderquan()

def make_supquan():
    sqlFile = "ins_supquan.sql"
    sqlData = open(sqlFile, 'w')
    for count in range(50):
        #supplier_id
        sqlData.write('INSERT INTO supply_order VALUES(1,')
        #price
        price = '%.2f' % random.uniform(100, 1000)
        sqlData.write(price + ',')
        #produce_id
        sqlData.write(str(random.randint(0,42)) + ',')
        #order_date
        sqlData.write('to_date(\'' + str(random.randint(1,12))  + '/' + str(random.randint(1,20)) + '/'
                + str(random.randint(2000, 2019)) + '\', ' + '\'mm/dd/yyyy\'),')
        #store_num
        sqlData.write(str(random.randint(0,2)) + ',')
        #weight
        weight = '%.2f' % random.uniform(100,500)
        sqlData.write(weight + ');\n')
    sqlData.write('commit;')
    sqlData.close()
make_supquan()

def make_prodnut():
    sqlFile = "ins_prodnut.sql"
    sqlData = open(sqlFile, 'w')
    for count in range(50):
        #produce_id
        sqlData.write('INSERT INTO produce_nutrition VALUES(' + str(random.randint(0,42)) + ',')
        #nutrition_id
        sqlData.write(str(random.randint(0,14)) + ',')
        #value
        sqlData.write(str(random.randint(0,500)) + ');\n')
    sqlData.write('commit;')
    sqlData.close()
make_prodnut()
```

I used a corpus for the first and last names, as well as for the addresses.

To show you what one of the files it creates would look like, and to demonstrate that I have grasped the concepts displayed in class, I will also show you one of the output files created from this function.

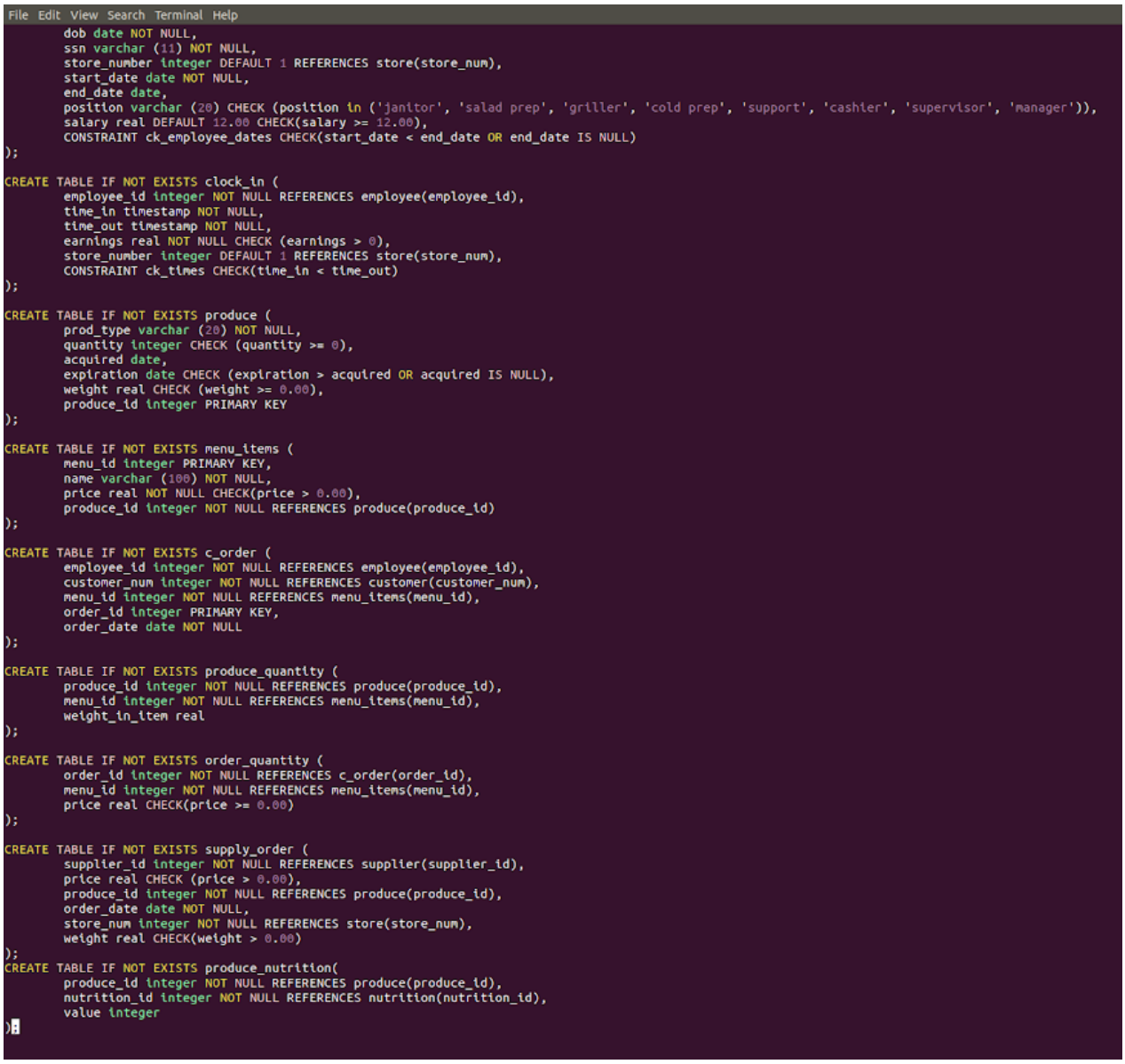
To further demonstrate this, I have also made a cleansing function for the database; however, it seemed faster to clear the entire SCHEMA and recreate it rather than purge and drop every single table. The function is simple and looks like this.



then , by calling \i sequoia.sql, I have a full database of empty tables.

The sequoia file had to be made in a specific order to ensure Foreign Keys were properly established.

```
File  Edit  View  Search  Terminal  Help
--Zakary Worman

CREATE TABLE IF NOT EXISTS customer (
        customer_num integer PRIMARY KEY,
        phone_number varchar (50),
        name varchar (50) NOT NULL
);

CREATE TABLE IF NOT EXISTS supplier (
        supplier_id integer PRIMARY KEY,
        name varchar (50) NOT NULL
);

CREATE TABLE IF NOT EXISTS nutrition (
        nutrition_id integer PRIMARY KEY,
        n_type varchar (50)
);

CREATE TABLE IF NOT EXISTS store (
        store_num integer PRIMARY KEY,
        location varchar (100)
);

CREATE TABLE IF NOT EXISTS employee (
        employee_id integer PRIMARY KEY,
        fname varchar (20) NOT NULL,
        minit varchar (1),
        lname varchar (20) NOT NULL,
        address varchar (50) NOT NULL,
        dob date NOT NULL,
        ssn varchar (11) NOT NULL,
        store_number integer DEFAULT 1 REFERENCES store(store_num),
        start_date date NOT NULL,
        end_date date,
        position varchar (20) CHECK (position in ('janitor', 'salad prep', 'griller', 'cold prep', 'support', 'cashier', 'supervisor', 'manager')),
        salary real DEFAULT 12.00 CHECK(salary >= 12.00),
        CONSTRAINT ck_employee_dates CHECK(start_date < end_date OR end_date IS NULL)
);

CREATE TABLE IF NOT EXISTS clock_in (
        employee_id integer NOT NULL REFERENCES employee(employee_id),
        time_in timestamp NOT NULL,
        time_out timestamp NOT NULL,
        earnings real NOT NULL CHECK (earnings > 0),
        store_number integer DEFAULT 1 REFERENCES store(store_num),
        CONSTRAINT ck_times CHECK(time_in < time_out)
);

CREATE TABLE IF NOT EXISTS produce (
        prod_type varchar (20) NOT NULL,
        quantity integer CHECK (quantity >= 0),
        acquired date,
        expiration date CHECK (expiration > acquired OR acquired IS NULL),
        weight real CHECK (weight >= 0.00),
        produce_id integer PRIMARY KEY
);

CREATE TABLE IF NOT EXISTS menu_items (
        menu_id integer PRIMARY KEY,
        name varchar (100) NOT NULL,
        price real NOT NULL CHECK(price > 0.00),
        produce_id integer NOT NULL REFERENCES produce(produce_id)
);

CREATE TABLE IF NOT EXISTS c_order (
        employee_id integer NOT NULL REFERENCES employee(employee_id),
        customer_num integer NOT NULL REFERENCES customer(customer_num),
        menu_id integer NOT NULL REFERENCES menu_items(menu_id),
"sequoia.sql" 97L, 3039C
```



Phase 4: DBMS Procedural Language, Stored Procedures and Triggers

1 Using Functions, Procedures and Triggers & Other DBMS

1.1 Postgres PL/pgSQL

What is PL/pgSQL?

PL/pgSQL is the base languaged used by PostgreSQL which is an open source object-relational database system. It is used to encompass the SQL language and to improve upon it. It was developed at Berkeley over 30 years ago.

What is PostgreSQL's program structure, control statements and cursors?

PostgreSQL uses a block structure to develop is functionality. There is a declare section followed by a begin section with statements an an end. This is very similar to ADA or other embedded system structures.

It also has a number of control structures similar to most other languages. These include RETURN to return a value from a function. From RETURN, it is possible to RETURN NEXT, RETURN QUERY, RETURN QUERY EXECUTE to amend the return value. It also possesses conditionals. These include IF/THEN/ELSE/ELSIF and CASE/WHEN/THEN/ELSE/END CASE for functionality similar to if then statements and switches. There are also loops. These are started with the LOOP statement and ended with END LOOP or EXIT. There are also CONTINUE statements to skip to the next element. WHILE loops can be used to do an instruction while a statement is true. FOR loops can be used to iterate a number of times. It's also possible to do FOREACH to loop through arrays. Trapping errors can be handled with the EXCEPTION statement followed by definitions for handling. The WHEN statement indicates the situation followed by a THEN statement to indicate the course of action.

A cursor is used when a person wants to read a query one row at a time. To create one, declare its name and the query it is used for. It is then opened with the OPEN statement. To use the cursor a FETCH statement is used. A MOVE statement repositions that cursor. CLOSE is used to close the cursor. UPDATE and DELETE can be used to change or remove a cursor.

What are stored procedures/functions?

Procedures and functions are very similar in PostgreSQL, the only difference being that procedures do not return a value. Also, a function is called as part of a query, and a procedure is called using the CALL statement.

The syntax for creating a procedure is:

```
CREATE [ OR REPLACE ] PROCEDURE
    name ( [ argmode ] [ argname ] argtype [ { DEFAULT | = } default_expr ] [, ...] ] )
  { LANGUAGE lang_name
    | TRANSFORM { FOR TYPE type_name } [, ...]
    | [ EXTERNAL ] SECURITY INVOKER | [ EXTERNAL ] SECURITY DEFINER
```

| SET **configuration_parameter** { TO **value** | = **value** | FROM CURRENT }
| AS **'definition'**
| AS **'obj_file', 'link_symbol'**
} ...

Where the bolded names are defined as follows:

Name: the name of the procedure

Argmode: default as IN, but can me changed to INOUT or VARIADIC

Argname: the name of the argument

Argtype: the data type(s) that will be used in the procedure.

Default_expr: defines the default expression

Lang_name: the name of the language that the procedure will use

Type_name: defines the types of transforms that can be used

Configure_parameter: setup to allow for value changing

Value: the type of value to be changed to and from

Definition: defines the procedure and can depend on the language

Obj_file: name of the shared library file containing the procedure

Link_symbol: is the procedure's name in the defined language

The syntax for creating a function is:

```
CREATE [ OR REPLACE ] FUNCTION
    name ( [ argmode ] [ argname ] argtype [ { DEFAULT | = } default_expr ] [, ...] ] )
    [ RETURNS rettype
        | RETURNS TABLE ( column_name column_type [, ...] ) ]
  { LANGUAGE lang_name
    | TRANSFORM { FOR TYPE type_name } [, ...]
    | WINDOW
    | IMMUTABLE | STABLE | VOLATILE | [ NOT ] LEAKPROOF
    | CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT | STRICT
    | [ EXTERNAL ] SECURITY INVOKER | [ EXTERNAL ] SECURITY DEFINER
    | PARALLEL { UNSAFE | RESTRICTED | SAFE }
    | COST execution_cost
    | ROWS result_rows
    | SET configuration_parameter { TO value | = value | FROM CURRENT }
    | AS 'definition'
    | AS 'obj_file', 'link_symbol'
```

} ...

Where the bolded names are defined as follows:

Name: the name of the function

Argmode: default as IN, but can me changed to INOUT or VARIADIC

Argname: the name of the argument

Argtype: the data type(s) that will be used in the procedure.

Default_expr: defines the default expression

Rettype: defines the return data type

Column_name: name of the return column

Column_type: data type of an output column

Lang_name: the name of the language that the procedure will use

Type_name: defines the types of transforms that can be used

Execution_cost: gives the estimated execution cost usual in terms of cpu power

Result_rows: gives estimated number of rows for return and defaults to 1000

Configure_parameter: setup to allow for value changing

Value: the type of value to be changed to and from

Definition: defines the procedure and can depend on the language

Obj_file: name of the shared library file containing the procedure

Link_symbol: is the procedure's name in the defined language

What is a trigger?

Triggers are functions that occur automatically when a specified event occurs.

The syntax for creating a trigger is:

CREATE [CONSTRAINT] TRIGGER **name** {BEFORE | AFTER | INSTEAD OF } { **event** [or ...]}
  ON **table_name**
  [FROM **referenced_table_name**]
  [NOT DEFERRABLE | [DEFERRABLE] [INITIALLY IMMEDIATE | INITIALLY DEFERRED]]
  [REFERENCING { { OLD | NEW } TABLE [AS] **tansition_relation_name** } [...]]
  [FOR [EACH] { ROW | STATEMENT }]

[WHEN (**condition**)]
EXECUTE { FUNCTION | PROCEDURE } **function_name** (**arguments**)

Event: can be INSERT, UPDATE, DELETE, TRUNCATE

Name: the name of the trigger

Table_name: name of the table, view, or foreign table the trigger is for

Referenced_table_name: can be given the name of tables referenced by the table the trigger is for

Transition_relation_name: specifies whether the trigger should be triggered for every row, or just once

Condition: determines when the trigger will be fired

Function_name: the function declared by the user that takes no argument and returns a trigger type

Arguments: arguments passed to the function, but will be passed as a literal string

Benefit of using procedures/functions over calling queries?

1. A created procedure or function will already have been tested and will work.
2. It will always be consistent because the same function/procedure is being called and people will not be able to forget or mistype anything.
3. It can often be faster and more efficient to create a function/procedure to call rather than write the query every single time.
4. It allows people who do not know any of the tables or relationships to understand and be able to pull information.
5. It allows a programmer to secure information by giving the information pulled from the functio/procedure, but keeping the tables secret (think with employee SSN).

1.2 Postgres PL/pgSQL Subprograms

```
CREATE OR REPLACE FUNCTION bot_average(n integer)
RETURNS real AS $$
DECLARE
        num real;
BEGIN
        SELECT AVG(price) INTO num FROM (SELECT price FROM menu_items ORDER BY
                price ASC LIMIT n) AS p;
        RETURN num;
END; $$
LANGUAGE plpgsql;
```

```
CREATE OR REPLACE FUNCTION top_average(n integer)
RETURNS real AS $$
DECLARE
        num real;
BEGIN
        SELECT AVG(price) INTO num FROM (SELECT price FROM menu_items ORDER BY
                price DESC LIMIT n) AS p;
        RETURN num;
END; $$
LANGUAGE plpgsql;
```

These function are created to return the bottom or top half of the table's average respectively.

```
zak=# SELECT bot_average(1);
 bot_average
-------------
        2.21
(1 row)

zak=# SELECT bot_average(2);
 bot_average
-------------
         2.4
(1 row)

zak=# SELECT bot_average(3);
 bot_average
-------------
     2.54667
(1 row)

zak=# SELECT bot_average(4);
 bot_average
-------------
      2.6875
(1 row)

zak=# SELECT bot_average(5);
 bot_average
-------------
        2.78
(1 row)

zak=# SELECT bot_average(10);
 bot_average
-------------
       3.134
(1 row)

zak=# SELECT bot_average(50);
 bot_average
-------------
      6.7308
(1 row)

zak=# SELECT bot_average(100);
 bot_average
-------------
     12.9508
(1 row)
```

```
zak=# SELECT top_average(5);
 top_average
-------------
      29.212
(1 row)

zak=# SELECT top_average(1);
 top_average
-------------
       29.56
(1 row)

zak=# SELECT top_average(2);
 top_average
-------------
       29.44
(1 row)

zak=# SELECT top_average(3);
 top_average
-------------
     29.3933
(1 row)

zak=# SELECT top_average(4);
 top_average
-------------
     29.3375
(1 row)

zak=# SELECT top_average(5);
 top_average
-------------
      29.212
(1 row)

zak=# SELECT top_average(10);
 top_average
-------------
      28.583
(1 row)

zak=# SELECT top_average(50);
 top_average
-------------
     22.0888
(1 row)

zak=# SELECT top_average(100);
 top_average
-------------
     15.2599
(1 row)

zak=#
```

```
CREATE OR REPLACE PROCEDURE insert_customer(integer, varchar(50), varchar(50))
LANGUAGE plpgsql
AS $$
BEGIN
        INSERT INTO customer
        VALUES($1, $2, $3);
        COMMIT;
END;
$$;
```

This inserts values into the customer table.

```
          43 | (380)028-8621 | Julio Ivanov
          44 | (973)798-5419 | Andriette Astley
          45 | (346)817-6157 | Jock Arrigucci
          46 | (524)536-0625 | Sawyer Aylett
          47 | (800)024-2813 | Kimmi Kobi
          48 | (321)733-8484 | Paulina Dantsig
          49 | (021)082-5159 | Gunvor Svoboda
           1 | (999)999-999  | Noelle Jefferies
(50 rows)

zak=# CALL insert_customer(100, '(559)246-1367', 'Zakary Worman');
CALL
zak=# SELECT * FROM customer;
 customer_num | phone_number  |         name
--------------+---------------+----------------------
           0 | (873)934-0950 | Theressa Drury
           2 | (740)521-4646 | Petronille Than
           3 | (912)394-2744 | Bryan Marino
           4 | (213)180-2332 | Pamella Tahan
           5 | (141)043-6347 | Mariette Zogby
           6 | (655)550-9576 | Jeana Axon
           7 | (253)809-5095 | Kissee Zogby
           8 | (931)596-7464 | Griselda Richards
           9 | (614)839-8582 | Iris Fakhoury
          10 | (698)854-9661 | Miran Mansour
          11 | (260)647-6031 | Elizabeth Nasikovsky
          12 | (413)031-3493 | Eugenie Golovatsky
          13 | (237)733-6103 | Jerrie Rothery
          14 | (006)951-8846 | Gail Munaev
          15 | (525)424-4081 | Shannon Masih
          16 | (294)730-0495 | Kincaid Zheravin
          17 | (554)140-3969 | Forster Jaffray
          18 | (465)553-4329 | Amalee Novosiltsev
          19 | (854)308-0224 | Gerrard Liu
          20 | (986)799-9459 | Petronille Moghadam
          21 | (963)040-2936 | Darth Etherton
          22 | (647)028-9482 | Rhianon Mcghee
          23 | (116)707-8348 | Stanwood Lancaster
          24 | (514)681-5054 | Mario Jefferies
          25 | (985)733-4912 | Chase Ashida
          26 | (553)220-6689 | Rosamond Divaev
          27 | (964)137-7493 | Petronia Ventura
          28 | (308)505-0736 | Randi Vass
          29 | (222)005-4390 | Filip Wagstaff
          30 | (110)551-1365 | Carlos Nakazawa
          31 | (408)090-5765 | Jennee Ganim
          32 | (203)310-9122 | Natala Gibson
          33 | (280)515-3517 | Ester Toichkin
          34 | (583)220-6505 | Quinton Ryjkin
          35 | (735)614-5092 | Hewett Demichev
          36 | (960)251-2913 | Enrika Salzwedel
          37 | (862)515-4136 | Amber Buonarroti
          38 | (262)258-0080 | Bryna Bagmut
          39 | (283)786-1856 | Val Iwasa
          40 | (964)979-5220 | Deedee Hawtin
          41 | (140)654-2475 | Hortense Lachapelle
          42 | (955)840-5698 | Marje Blanco
          43 | (380)028-8621 | Julio Ivanov
          44 | (973)798-5419 | Andriette Astley
          45 | (346)817-6157 | Jock Arrigucci
          46 | (524)536-0625 | Sawyer Aylett
          47 | (800)024-2813 | Kimmi Kobi
          48 | (321)733-8484 | Paulina Dantsig
          49 | (021)082-5159 | Gunvor Svoboda
           1 | (999)999-999  | Noelle Jefferies
         100 | (559)246-1367 | Zakary Worman
(51 rows)
```

```
CREATE OR REPLACE PROCEDURE delete_customer(integer, varchar(50), varchar(50))
LANGUAGE plpgsql
AS $$
BEGIN
        DELETE FROM customer
        WHERE customer_num = $1 OR phone_number = $2 OR name = $3;
        COMMIT;
END;
$$;
```

These deletes a customer with the specified attributes.

```
          40 | (964)979-5220 | Deedee Hawtin
          41 | (140)654-2475 | Hortense Lachapelle
          42 | (955)840-5698 | Marje Blanco
          43 | (380)028-8621 | Julio Ivanov
          44 | (973)798-5419 | Andriette Astley
          45 | (346)817-6157 | Jock Arrigucci
          46 | (524)536-0625 | Sawyer Aylett
          47 | (800)024-2813 | Kimmi Kobi
          48 | (321)733-8484 | Paulina Dantsig
          21 | (963)040-2936 | Darth Vader
         100 | (559)246-1367 | Zakary Worman
(49 rows)

zak=# CALL delete_customer(100, '(559)246-1367', 'Zakary Worman');
CALL
zak=# SELECT * FROM customer;
 customer_num | phone_number  |         name
--------------+---------------+----------------------
           0 | (873)934-0950 | Theressa Drury
           2 | (740)521-4646 | Petronille Than
           3 | (912)394-2744 | Bryan Marino
           4 | (213)180-2332 | Pamella Tahan
           5 | (141)043-6347 | Mariette Zogby
           6 | (655)550-9576 | Jeana Axon
           7 | (253)809-5095 | Kissee Zogby
           8 | (931)596-7464 | Griselda Richards
           9 | (614)839-8582 | Iris Fakhoury
          10 | (698)854-9661 | Miran Mansour
          11 | (260)647-6031 | Elizabeth Nasikovsky
          12 | (413)031-3493 | Eugenie Golovatsky
          13 | (237)733-6103 | Jerrie Rothery
          14 | (006)951-8846 | Gail Munaev
          15 | (525)424-4081 | Shannon Masih
          16 | (294)730-0495 | Kincaid Zheravin
          17 | (554)140-3969 | Forster Jaffray
          18 | (465)553-4329 | Amalee Novosiltsev
          19 | (854)308-0224 | Gerrard Liu
          20 | (986)799-9459 | Petronille Moghadam
          22 | (647)028-9482 | Rhianon Mcghee
          23 | (116)707-8348 | Stanwood Lancaster
          24 | (514)681-5054 | Mario Jefferies
          25 | (985)733-4912 | Chase Ashida
          26 | (553)220-6689 | Rosamond Divaev
          27 | (964)137-7493 | Petronia Ventura
          28 | (308)505-0736 | Randi Vass
          29 | (222)005-4390 | Filip Wagstaff
          30 | (110)551-1365 | Carlos Nakazawa
          31 | (408)090-5765 | Jennee Ganim
          32 | (203)310-9122 | Natala Gibson
          33 | (280)515-3517 | Ester Toichkin
          34 | (583)220-6505 | Quinton Ryjkin
          35 | (735)614-5092 | Hewett Demichev
          36 | (960)251-2913 | Enrika Salzwedel
          37 | (862)515-4136 | Amber Buonarroti
          38 | (262)258-0080 | Bryna Bagmut
          39 | (283)786-1856 | Val Iwasa
          40 | (964)979-5220 | Deedee Hawtin
          41 | (140)654-2475 | Hortense Lachapelle
          42 | (955)840-5698 | Marje Blanco
          43 | (380)028-8621 | Julio Ivanov
          44 | (973)798-5419 | Andriette Astley
          45 | (346)817-6157 | Jock Arrigucci
          46 | (524)536-0625 | Sawyer Aylett
          47 | (800)024-2813 | Kimmi Kobi
          48 | (321)733-8484 | Paulina Dantsig
          21 | (963)040-2936 | Darth Vader
(48 rows)
```

```
CREATE OR REPLACE FUNCTION f_customer_update()
RETURNS trigger AS $BODY$
BEGIN
        IF NEW.customer_num != OLD.customer_num THEN
        UPDATE c_order set customer_num = NEW.customer_num
        WHERE customer_num = OLD.customer_num;
        END IF;

        RETURN NEW;
END;
$BODY$ LANGUAGE plpgsql;

CREATE TRIGGER customer_update
        BEFORE UPDATE
        ON customer
        FOR EACH ROW
        EXECUTE PROCEDURE f_customer_update();
```

This updates all keys that reference customer_num.

```
          43 | (380)028-8621 | Julio Ivanov
          44 | (973)798-5419 | Andriette Astley
          45 | (346)817-6157 | Jock Arrigucci
          46 | (524)536-0625 | Sawyer Aylett
          47 | (800)024-2813 | Kimmi Kobi
          48 | (321)733-8484 | Paulina Dantsig
          49 | (021)082-5159 | Gunvor Svoboda
(50 rows)

zak=# UPDATE customer
SET customer_num = 100
WHERE customer_num = 1;
UPDATE 1
zak=# SELECT * FROM customer;
 customer_num | phone_number  |         name
--------------+---------------+----------------------
           0 | (873)934-0950 | Theressa Drury
           2 | (740)521-4646 | Petronille Than
           3 | (912)394-2744 | Bryan Marino
           4 | (213)180-2332 | Pamella Tahan
           5 | (141)043-6347 | Mariette Zogby
           6 | (655)550-9576 | Jeana Axon
           7 | (253)809-5095 | Kissee Zogby
           8 | (931)596-7464 | Griselda Richards
           9 | (614)839-8582 | Iris Fakhoury
          10 | (698)854-9661 | Miran Mansour
          11 | (260)647-6031 | Elizabeth Nasikovsky
          12 | (413)031-3493 | Eugenie Golovatsky
          13 | (237)733-6103 | Jerrie Rothery
          14 | (006)951-8846 | Gail Munaev
          15 | (525)424-4081 | Shannon Masih
          16 | (294)730-0495 | Kincaid Zheravin
          17 | (554)140-3969 | Forster Jaffray
          18 | (465)553-4329 | Amalee Novosiltsev
          19 | (854)308-0224 | Gerrard Liu
          20 | (986)799-9459 | Petronille Moghadam
          21 | (963)040-2936 | Darth Etherton
          22 | (647)028-9482 | Rhianon Mcghee
          23 | (116)707-8348 | Stanwood Lancaster
          24 | (514)681-5054 | Mario Jefferies
          25 | (985)733-4912 | Chase Ashida
          26 | (553)220-6689 | Rosamond Divaev
          27 | (964)137-7493 | Petronia Ventura
          28 | (308)505-0736 | Randi Vass
          29 | (222)005-4390 | Filip Wagstaff
          30 | (110)551-1365 | Carlos Nakazawa
          31 | (408)090-5765 | Jennee Ganim
          32 | (203)310-9122 | Natala Gibson
          33 | (280)515-3517 | Ester Tolchkin
          34 | (583)220-6505 | Quinton Ryjkin
          35 | (735)614-5092 | Hewett Demichev
          36 | (960)251-2913 | Enrika Salzwedel
          37 | (862)515-4136 | Amber Buonarroti
          38 | (262)258-0080 | Bryna Bagmut
          39 | (283)786-1856 | Val Iwasa
          40 | (964)979-5220 | Deedee Hawtin
          41 | (140)654-2475 | Hortense Lachapelle
          42 | (955)840-5698 | Marje Blanco
          43 | (380)028-8621 | Julio Ivanov
          44 | (973)798-5419 | Andriette Astley
          45 | (346)817-6157 | Jock Arrigucci
          46 | (524)536-0625 | Sawyer Aylett
          47 | (800)024-2813 | Kimmi Kobi
          48 | (321)733-8484 | Paulina Dantsig
          49 | (021)082-5159 | Gunvor Svoboda
         100 | (329)275-2915 | Noelle Jefferies
(50 rows)
```

```
CREATE OR REPLACE FUNCTION f_customer_delete()
RETURNS trigger AS $BODY$
BEGIN
        DELETE FROM c_order o WHERE o.customer_num = OLD.customer_num;

        RETURN OLD;
END;
$BODY$ LANGUAGE plpgsql;

CREATE TRIGGER customer_delete
        BEFORE DELETE
        ON customer
        FOR EACH ROW
        EXECUTE FUNCTION f_customer_delete();
```

```
CREATE OR REPLACE FUNCTION f_order_delete()
RETURNS trigger AS $BODY$
BEGIN
        DELETE FROM order_quantity o WHERE o.order_id = OLD.order_id;

        RETURN OLD;
END;
$BODY$ LANGUAGE plpgsql;

CREATE TRIGGER order_delete
        BEFORE DELETE
        ON c_order
        FOR EACH ROW
        EXECUTE FUNCTION f_order_delete();
```

These are used to delete from customer because c_order references customer_num and order_quantity references order_id.

```
           39 | (283)786-1856 | Val Iwasa
           40 | (964)979-5220 | Deedee Hawtin
           41 | (140)654-2475 | Hortense Lachapelle
           42 | (955)840-5698 | Marje Blanco
           43 | (380)028-8621 | Julio Ivanov
           44 | (973)798-5419 | Andriette Astley
           45 | (346)817-6157 | Jock Arrigucci
           46 | (524)536-0625 | Sawyer Aylett
           47 | (800)024-2813 | Kimmi Kobi
           48 | (321)733-8484 | Paulina Dantsig
           49 | (021)082-5159 | Gunvor Svoboda
(49 rows)

zak=# DELETE FROM customer WHERE customer_num = 49;
DELETE 1
zak=# SELECT * FROM customer;
 customer_num | phone_number  |         name
--------------+---------------+----------------------
            0 | (873)934-0950 | Theressa Drury
            2 | (740)521-4646 | Petronille Than
            3 | (912)394-2744 | Bryan Marino
            4 | (213)180-2332 | Pamella Tahan
            5 | (141)043-6347 | Mariette Zogby
            6 | (655)550-9576 | Jeana Axon
            7 | (253)809-5095 | Kissee Zogby
            8 | (931)596-7464 | Griselda Richards
            9 | (614)839-8582 | Iris Fakhoury
           10 | (698)854-9661 | Miran Mansour
           11 | (260)647-6031 | Elizabeth Nasikovsky
           12 | (413)031-3493 | Eugenie Golovatsky
           13 | (237)733-6103 | Jerrie Rothery
           14 | (006)951-8846 | Gail Munaev
           15 | (525)424-4081 | Shannon Masih
           16 | (294)730-0495 | Kincaid Zheravin
           17 | (554)140-3969 | Forster Jaffray
           18 | (465)553-4329 | Amalee Novosiltsev
           19 | (854)308-0224 | Gerrard Liu
           20 | (986)799-9459 | Petronille Moghadam
           21 | (963)040-2936 | Darth Etherton
           22 | (647)028-9482 | Rhianon Mcghee
           23 | (116)707-8348 | Stanwood Lancaster
           24 | (514)681-5054 | Mario Jefferies
           25 | (985)733-4912 | Chase Ashida
           26 | (553)220-6689 | Rosamond Divaev
           27 | (964)137-7493 | Petronia Ventura
           28 | (308)505-0736 | Randi Vass
           29 | (222)005-4390 | Filip Wagstaff
           30 | (110)551-1365 | Carlos Nakazawa
           31 | (408)090-5765 | Jennee Ganim
           32 | (203)310-9122 | Natala Gibson
           33 | (280)515-3517 | Ester Toichkin
           34 | (583)220-6505 | Quinton Ryjkin
           35 | (735)614-5092 | Hewett Demichev
           36 | (960)251-2913 | Enrika Salzwedel
           37 | (862)515-4136 | Amber Buonarroti
           38 | (262)258-0080 | Bryna Bagmut
           39 | (283)786-1856 | Val Iwasa
           40 | (964)979-5220 | Deedee Hawtin
           41 | (140)654-2475 | Hortense Lachapelle
           42 | (955)840-5698 | Marje Blanco
           43 | (380)028-8621 | Julio Ivanov
           44 | (973)798-5419 | Andriette Astley
           45 | (346)817-6157 | Jock Arrigucci
           46 | (524)536-0625 | Sawyer Aylett
           47 | (800)024-2813 | Kimmi Kobi
           48 | (321)733-8484 | Paulina Dantsig
(48 rows)
```

```
CREATE VIEW customerview AS
        SELECT cu.customer_num AS num,
        cu.phone_number AS phone,
        cu.name AS name,
        o.order_id AS ord
        FROM customer cu INNER JOIN c_order o USING(customer_num);


CREATE OR REPLACE FUNCTION update_customerview_func()
RETURNS trigger AS
$func$
BEGIN
        UPDATE customer SET customer_num = NEW.num
        WHERE customer_num = OLD.num;
        UPDATE customer SET phone_number = NEW.phone
        WHERE phone_number = OLD.phone;
        UPDATE customer SET name = NEW.name
        WHERE name = OLD.name;
        UPDATE c_order SET order_id = NEW.ord
        WHERE order_id = OLD.ord;
        RETURN NEW;
END
$func$ LANGUAGE plpgsql;

CREATE TRIGGER insert_customerview_trig
        INSTEAD OF UPDATE ON customerview
        FOR EACH ROW EXECUTE PROCEDURE update_customerview_func();
```

This function updates the called tables when an update is applied to customer view.

```
  35 | (735)614-5092 | Hewett Demichev     |  40
  22 | (647)028-9482 | Rhianon Mcghee      |  41
  12 | (413)031-3493 | Eugenie Golovatsky  |  42
  35 | (735)614-5092 | Hewett Demichev     |  43
  42 | (955)840-5698 | Marje Blanco        |  44
  18 | (465)553-4329 | Amalee Novosiltsev  |  45
  40 | (964)979-5220 | Deedee Hawtin       |  46
  33 | (280)515-3517 | Ester Toichkin      |  47
   2 | (740)521-4646 | Petronille Than     |  48
  24 | (514)681-5054 | Mario Jefferies     |  49
(47 rows)

zak=# UPDATE customerview
zak-# SET name = 'Darth Vader'
zak-# WHERE name = 'Darth Etherton';
UPDATE 2
zak=# SELECT * FROM customerview;
 num |     phone      |        name         | ord
-----+----------------+---------------------+-----
   6 | (655)550-9576 | Jeana Axon          |   0
  23 | (116)707-8348 | Stanwood Lancaster  |   3
   5 | (141)043-6347 | Mariette Zogby      |   4
  12 | (413)031-3493 | Eugenie Golovatsky  |   6
  16 | (294)730-0495 | Kincaid Zheravin    |   7
   3 | (912)394-2744 | Bryan Marino        |   9
  19 | (854)308-0224 | Gerrard Liu         |  10
  48 | (321)733-8484 | Paulina Dantsig     |  11
  18 | (465)553-4329 | Amalee Novosiltsev  |  12
  41 | (140)654-2475 | Hortense Lachapelle |  13
  22 | (647)028-9482 | Rhianon Mcghee      |  14
  44 | (973)798-5419 | Andriette Astley    |  15
   4 | (213)180-2332 | Pamella Tahan       |  16
  28 | (308)505-0736 | Randi Vass          |  17
  10 | (698)854-9661 | Miran Mansour       |  18
  31 | (408)090-5765 | Jennee Ganim        |  19
  10 | (698)854-9661 | Miran Mansour       |  20
  47 | (800)024-2813 | Kimmi Kobi          |  21
   5 | (141)043-6347 | Mariette Zogby      |  22
  39 | (283)786-1856 | Val Iwasa           |  23
   7 | (253)809-5095 | Kissee Zogby        |  24
  48 | (321)733-8484 | Paulina Dantsig     |  25
  33 | (280)515-3517 | Ester Toichkin      |  26
  15 | (525)424-4081 | Shannon Masih       |  27
   2 | (740)521-4646 | Petronille Than     |  29
  10 | (698)854-9661 | Miran Mansour       |  30
  16 | (294)730-0495 | Kincaid Zheravin    |  31
   7 | (253)809-5095 | Kissee Zogby        |  32
  38 | (262)258-0080 | Bryna Bagmut        |  33
  13 | (237)733-6103 | Jerrie Rothery      |  34
  41 | (140)654-2475 | Hortense Lachapelle |  35
  14 | (006)951-8846 | Gail Munaev         |  36
  36 | (960)251-2913 | Enrika Salzwedel    |  37
  17 | (554)140-3969 | Forster Jaffray     |  38
  19 | (854)308-0224 | Gerrard Liu         |  39
  35 | (735)614-5092 | Hewett Demichev     |  40
  22 | (647)028-9482 | Rhianon Mcghee      |  41
  12 | (413)031-3493 | Eugenie Golovatsky  |  42
  35 | (735)614-5092 | Hewett Demichev     |  43
  42 | (955)840-5698 | Marje Blanco        |  44
  18 | (465)553-4329 | Amalee Novosiltsev  |  45
  40 | (964)979-5220 | Deedee Hawtin       |  46
  33 | (280)515-3517 | Ester Toichkin      |  47
   2 | (740)521-4646 | Petronille Than     |  48
  24 | (514)681-5054 | Mario Jefferies     |  49
  21 | (963)040-2936 | Darth Vader         |   2
  21 | (963)040-2936 | Darth Vader         |  28
(47 rows)
```

1.3 PL/pgSQL vs Microsoft SQL Server, MySQL and Oracle DBMS

A brief description about procedural languages, are a type of imperative languages, which mean that its execution is based on the statements. Such as, SQL, HTML, and BASIC, etc. The other kind of programming languages are called paradigm declarative programming languages, and their executions are just based on the expressions. Such as, C, C++, and Java, etc. Note, PL/SQL means procedural language SQL that we will use its shortcut in our coming detailed description.

There are three DBMS: Microsoft SQL server MySQL, and Oracle. These each have their benefits and disabilities when using them to create and maintain databases. We will discuss the differences and similarities between these three different DBMS and their syntaxes.

Microsoft SQL Server is a DBMS used and distributed by Microsoft.
It is the only DBMS of the three that is developed by Microsoft and not Oracle, and was released in 1989. To keep up with the advance in technology, Microsoft created this DBMS to support new features such as XML data, enhanced compression, support for structures, semi-structured data, and several add-on products to support other marketing products. It contains languages from libraries such as C#, Python, PHP, Java, Ruby, C++, Node.js. Microsoft SQL Server functions mostly through the use of the languages R and Python. It was designed to handle disaster recovery, given tools for business intelligence by various updated versions, and support to OLTP (Online transaction processing) and some embedded features for enhanced performance of a sophisticated database.

MySQL is a DBMS used and distributed by the Oracle Corporation. Operating systems supported are FreeBSD, Linux, OS X, Solaris, and Windows. Supported languages are Ada, C, C#, C++, D, Delphi, Eiffel, Erlang, Haskell, Java, Javascript, Objective-C, OCaml, Perl, PHP, Python, Ruby, Scheme, and TCL. MySQL has standard database security with table-driven security. MySQL allows the use of several expressions at several points during its statements. They can be written using literal values, column values, Null, built-in functions, and so on. All MySQL databases are relational and store data in separate tables rather than storing it all in one location. MySQL is also Open Source, meaning that anyone can take a look at the software and modify it if need be. Originally, this DBMS was meant to handle large databases much

faster than existing solutions. MySQL Server works in embedded systems, so it is possible to be provided as an embedded multithreaded library into an application.

Oracle was the DBMS created by Larry Ellison, Bob Miner, and Ed Oates in 1977 through the consultancy Software Development Laboratories. Later, it became Relational Software, Inc, and then Oracle Systems Corporation in 1983, and finally becoming Oracle Corporation. In addition, Oracle PL/SQL is a procedural language that was designed to adopt SQL statements within their rich of known syntaxes. PL/SQL is a unit program that can be easily compiled by the Oracle Database servers, and then store all the information in their databases. During run time, both PL/SQL and regular SQL can run over the same server process, which provides many benefits such as optimality and efficiency. Also, PL/SQL provides unique and strong features that automatically gives trustful, security, and portability of the Oracle databases. Features included are portability, PL/SQL stored program units, objects and partitioning, internet computing, Oracle Real Application Clusters, Grid computing, Manageability, diagnosability, and availability. Oracle functions so that all statements are all committed, or not committed at all. Like other DBMSs, Oracle allows the control of concurrency, which is simultaneous access of the same data by multiple users.

Differences and similarities between these three DBMS are quite clear in their syntaxes, their databases, and their company's servers, such as Microsoft, and Oracle.

Starting with similarities, the similarities between all of these three procedural languages are using the name of SQL (Structured Query Language) to interact between relational databases, and there are many likeness between them as well as many differences under the hood. To be fair in comparisons all of these procedural languages are powerful, provide solid features, and friendly in use. As an example of Transact-SQL or T-SQL is a central of using Microsoft SQL Server, and all of its applications are communicating with an instance of SQL Servers. All of this three languages are using block of codes that are used to write an entire program blocks, procedure, functions, packages, etc. These are languages that executing the whole block of codes unlike SQL execute only single statement. We mostly can say about PL/SQL, MySQL, T-SQL are procedural languages or sometimes called extension of SQL that can be used to build applications.

Also, the syntaxes between the three languages are similar, specially when we use the syntax in writing functions, procedures, triggers, etc. Another thing is that these languages are relational databases management system RDBMS and they are called

open source database systems as well, which means it can be used for whatever we need to use it for, either for personal use, big projects by companies, commercial use websites, etc. All of MySQL, T-SQL, and PL/SQL are supporting other programming languages, such as Java, PHP, C++, Python, etc. So by supporting dynamic and object-oriented languages, makes this MySQL, T-SQL, and PL/SQL strong, efficient, high-speed, reliable among other RDBMS languages.

The differences between all of these three procedural languages are some of these languages do not work on some of the operating systems, such as Oracle does not run with BSD, AmigaOS, and some other Linux distributions. But MySQL is working on most common operating systems, such as Window, Mac, Linux, Unix, BSD, AmigaOS, etc. In addition, Oracle has a lot of features that make Oracle very strong, such as user-defined types to XML and many business applications use Oracle because of its huge features needed for businesses, reliability, support of the Oracle and community, and flexibility in use. Also, MySQL offered fast-speed while using and implementing database systems, and we can say that small businesses or companies are demanding on MySQL more for their websites because small companies do not need the huge features that Oracle provided for them, so it seems MySQL is quite enough for small business. Comparing between PL/SQL and MySQL, PL/SQL is an advanced programming language or sometimes called the fourth generation language, and PL/SQL is provided great software engineering features that all data encapsulate, overloaded, and exceptions, so on. In MySQL side is weaker than PL/SQL and T-SQL in area of inserting, deleting data. In short, there are too many differences and similarities which cannot be count among the three languages, so all the similarities and differences that were provided are helpful for us to know how to choose the right language for the correct place.

Syntax for the three procedural languages:

The sample syntax for selective :

- if statements:
- Microsoft SQL Server'(Transact-SQL)

IF Boolean_expression

{ sql_statement | statement_block }

[ELSE

{ sql_statement | statement_block }]

.....

- MySQL

SELECT IF(expression, expr_true, expr_false);

.....

- Oracle' DBMS (PL/SQL)

IF condition THEN

{...statements to execute when condition is TRUE...}

END IF;

.....

- switch statements:
- Microsoft SQL Server'(Transact-SQL)

switch (expr)

{

case 1 : statement 1;

break;

case 2 : statement 2;

break;

```
case 3 : statement 3;
break;
}
```

.....

- MySQL

```
CASE
        WHEN condition1 THEN result1
        WHEN condition2 THEN result2
        WHEN condition_n THEN result_n
        ELSE result
END;
```

.....

- Oracle' DBMS (PL/SQL)

```
CASE [ expression ]
  WHEN ‘condition_1’ THEN result_1;
  WHEN ‘condition_2’ THEN result_2;
  WHEN ‘condition_n’ THEN result_n;
  ELSE result;
END CASE;
```

.....

The sample syntax for repetitive statements:

- For statements:
- Microsoft SQL Server'(Transact-SQL)

GO [count]

.....

- MySQL

BEGIN

label1: LOOP

Statement_list;

END LOOP label1;

.....

- Oracle' DBMS (PL/SQL)

BEGIN

FOR counter IN initial_value .. final_value LOOP

sequence_of_statements;

END LOOP;

.....

- While statements:
- Microsoft SQL Server’(Transact-SQL)

WHILE Boolean_expr

{sql_statment | statment_block | BREAK | CONTINUE }

.....

- MySQL

Label_name: WHILE condition DO

{...statements...}

END WHILE [label_name];

.....

- Oracle' DBMS (PL/SQL)

WHILE condition

LOOP

{...statements...}

END LOOP;

.....

The sample syntax for statements of creating stored procedures/functions/triggers:

- Creating stored procedures:
- Microsoft SQL Server'(Transact-SQL)

CREATE [OR ALTER] { PROC | PROCEDURE }

[schema_name.] procedure_name [; number]

[{ @parameter [type_schema_name.] data_type }

[VARYING] [= default] [OUT | OUTPUT | [READONLY]

] [,...n]

[WITH <procedure_option> [,...n]]

[FOR REPLICATION]

AS { [BEGIN] sql_statement [;] [...n] [END] }

[;]

<procedure_option> ::=

[ENCRYPTION]

[RECOMPILE]

[EXECUTE AS Clause]

.....

- MySQL

CREATE PROCEDURE

(IN con CHAR(20))

BEGIN

SELECT Name, ID FROM Employee

WHERE John = joh;

END;

.....

- Oracle' DBMS (PL/SQL)

CREATE OR REPLACE PROCEDURE

<procedure_name>

(

<parameter1 IN/OUT <data_type>

Statements

...

)

[IS | AS]

<declaration_part>

BEGIN

<execution part>

EXCEPTION

<exception handling part>

END;

.....

- Creating functions:
- Microsoft SQL Server'(Transact-SQL)

CREATE [OR ALTER] FUNCTION [schema_name.] function_name

([{ @parameter_name [AS][type_schema_name.] parameter_data_type

[= default] [READONLY] }

```
    [ ,...n ]
  ]
)
RETURNS return_data_type
    [ WITH <function_option> [ ,...n ] ]
    [ AS ]
    BEGIN
        function_body
        RETURN scalar_expression
    END
[ ; ]
```

.....

- MySQL

```
CREATE FUNCTION function_name(param1,param2,...)
        RETURNS data_type
        [NOT] DETERMINISTIC
 Statements;
END;
```

.....

- Oracle' DBMS (PL/SQL)

CREATE OR REPLACE FUNCTION

<procedure_name>

(

<parameter1 IN/OUT <data_type>

)

RETURN <data_type>

[IS | AS]

<declaration_part>

BEGIN

<function_body>

END;

.....

- Creating triggers:
- Microsoft SQL Server'(Transact-SQL)

CREATE [OR ALTER] TRIGGER [schema_name.] trigger_name

ON { table | view }

[WITH <dml_trigger_option> [,...n]]

{ FOR | AFTER | INSTEAD OF }

{ [INSERT] [,] [UPDATE] [,] [DELETE] }

[WITH APPEND]

[NOT FOR REPLICATION]

AS { sql_statement [;] [,...n] | EXTERNAL NAME <method specifier [;] > }

<dml_trigger_option> ::=

[ENCRYPTION]

[EXECUTE AS Clause]

<method_specifier> ::=

assembly_name.class_name.method_name

.....

- MySQL

CREATE

[DEFINER = user]

TRIGGER trigger_name

trigger_time trigger_event

ON tbl_name FOR EACH ROW

[trigger_order]

<trigger_body>

trigger_time: { BEFORE | AFTER }

trigger_event: { INSERT | UPDATE | DELETE }

trigger_order: { FOLLOWS | PRECEDES } another_trigger_name

.....

- Oracle' DBMS (PL/SQL)

```
TRIGGER trigger_name
  triggering_event
  [ trigger_restriction ]
BEGIN
  triggered_action;
END;
```

Phase 5: Graphical User Interface Design and Implementation

5.1 Functionalities

The purpose of the website that I created was to serve as a platform for a user to get access to the nutritional values and aspects of the sandwiches offered by Sequoia Sandwich Company. On top of this, I added some menu functionality, as well as a way to find the locations of the stores in Bakersfield.

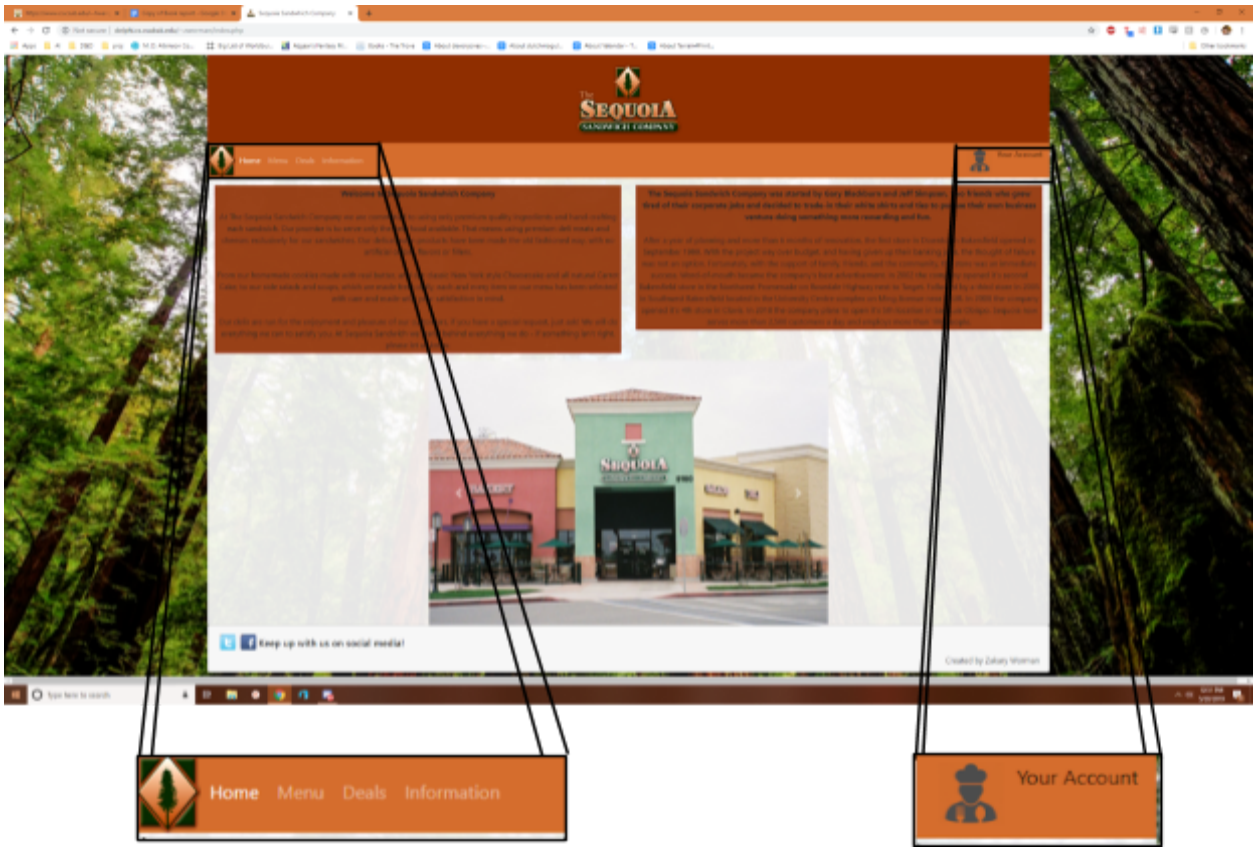
5.1.1 Itemized Descriptions of GUI

1. The Home Page:
   Simple as it may be, the landing page for my website simply shows the description of the company and their values, and a carousel of photos for each of the locations. This gives the customer an idea of what the restaurant is about and the type of people that work there. There is a navigation bar across all the webpages to easily access other parts of the website and to sign in. It also clearly displays which page the user is currently sitting on.
2. Sign-in Modal:
   In order to sign in, the user must look at a popup menu that asks for a username and a password. At the bottom of the modal rests a registration link, so a new user can easily create an account.
3. Registration Page:
   The registration page is simply a number of boxes present for the user to fill out in order to complete their account. Once created, the account will track all their orders.
4. Account Page:
   There is an account page for each user that is unique to them. Again, the focus of my part was on the nutrition, so I did not spend a lot of time developing this page. However, it is clear the direction I intended to go, should I decide to have continued developing the webpage. Currently, it shows all the previous orders of the signed in user.
5. Menu Page:
   The menu page shows every item available at the store. Once clicked, a modal pops up with the price and all nutritional values of the item. It has a checkbox if

the user wants to favorite and item, and an add to order option. Due to the scope of the project, I did not implement any cart option or a way to save favorite items. This would be another case of things to add on if I decided to continue.

6. Deals Page:
   This was intended to be a place to show all the on sale sandwiches currently available at the store. However, I only had it add random sandwiches to the list to demonstrate what kind of info would be on here. It would have been better to add the discount prices easily available to the user, but the true values from the database are still presented on the items here the same as they would be seen on the menu.
7. Information Page:
   The information page shows the map of Bakersfield with all of the store locations marked. It also holds links to both reports for the project.
8. Report 1 (Nutrition Report):
   This report shows all the produce items that make up a menu item followed by the nutritional values for that menu item. It also later shows all the nutritional values for every produce item.
9. Report 2 (Sandwich Frequency):
   This report displays a bar graph of each menu item, the number of purchases, and the percentage of the purchases is posses.

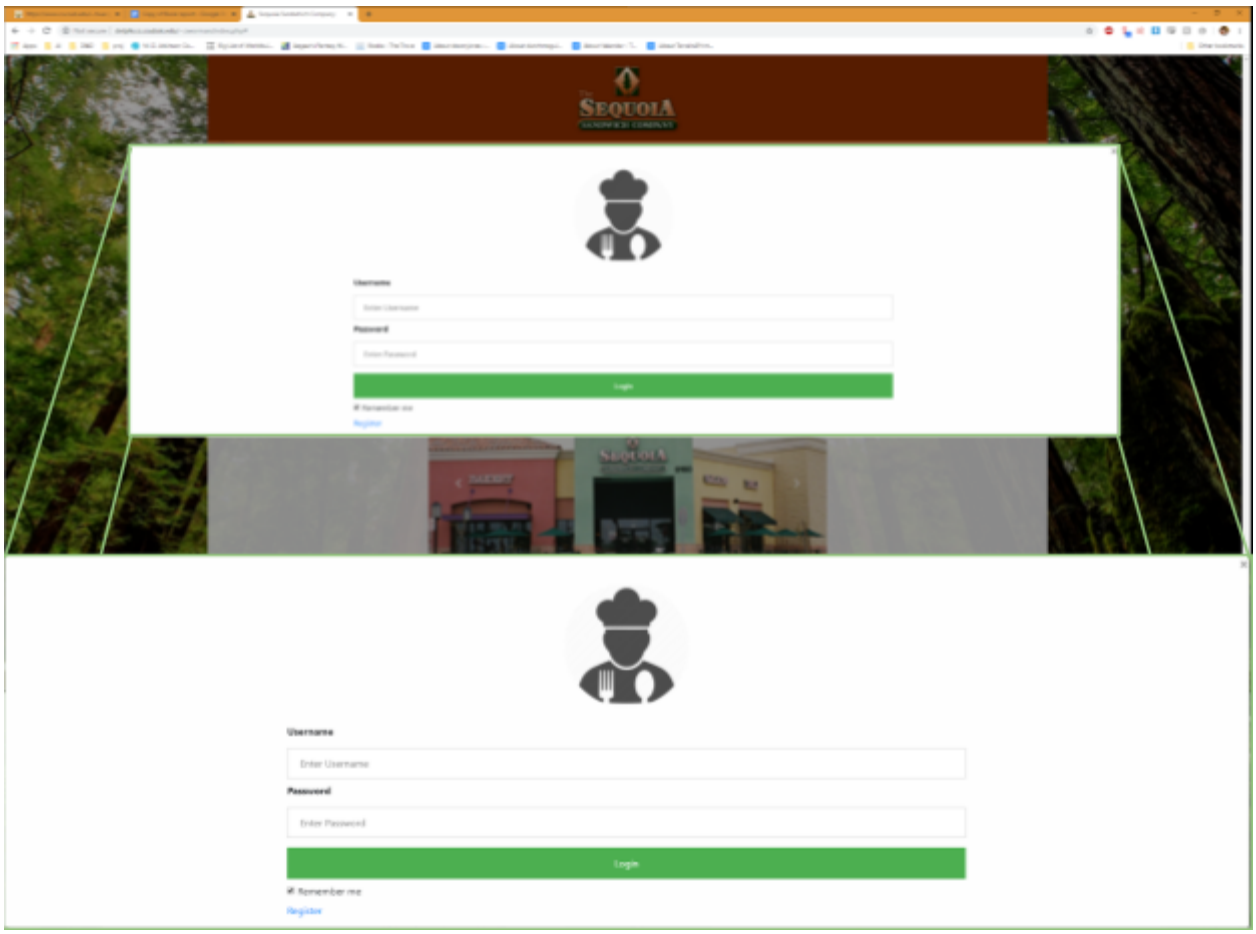
5.1.2 Screenshots and Descriptions



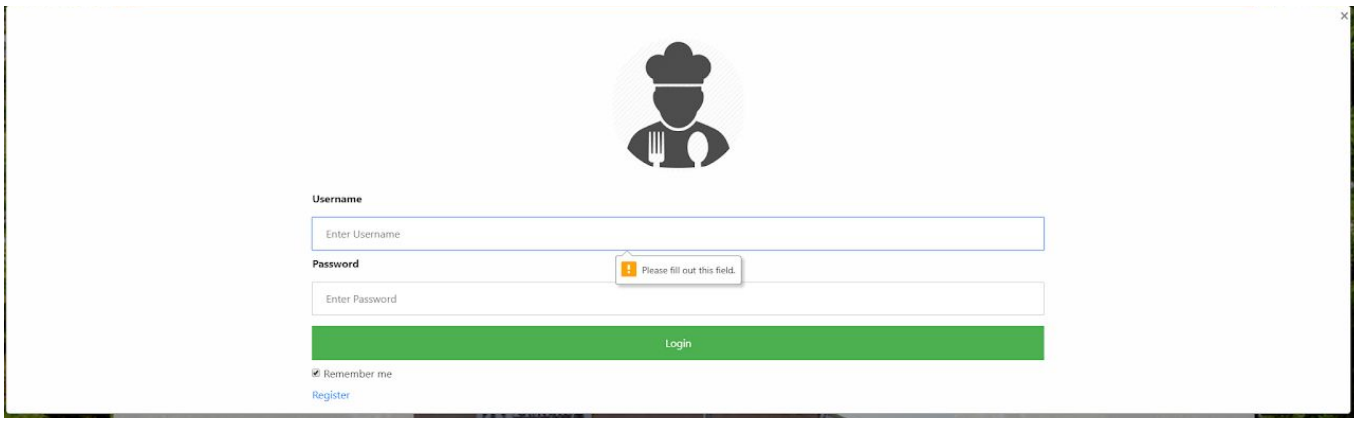
The left side of this navbar allows for easy access to the other pages of this site. As can be seen from the headings of “Home”, “Menu”, “Deals”, and “Information”.

The other side shows an avatar image with an account tag. This tag uses javascript to display a modal that can be used to sign in.

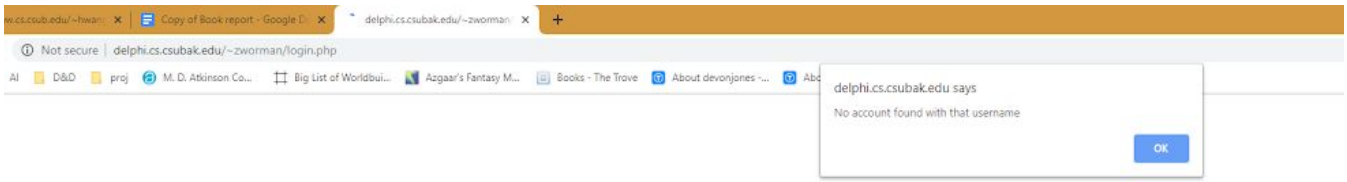
As such:



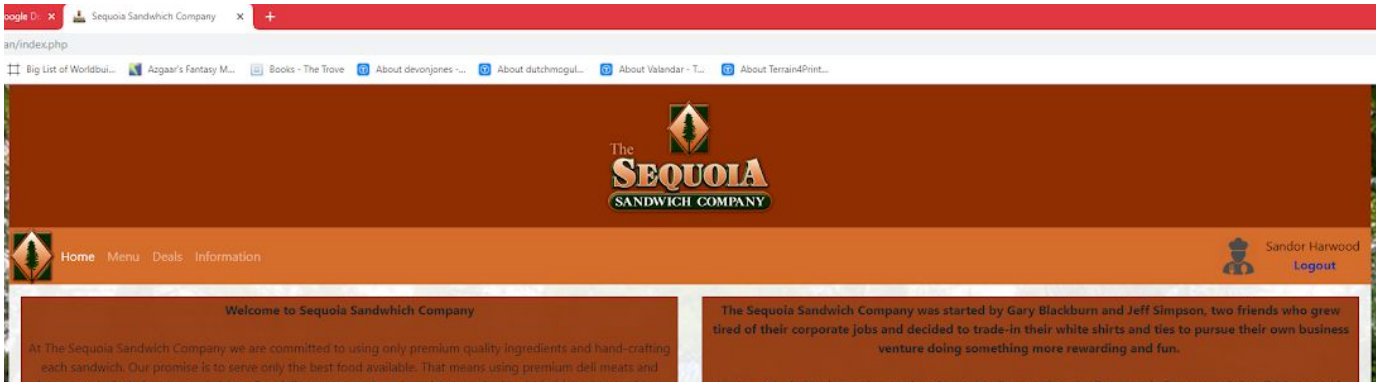
From here, the user inputs their username and password and then it is verified against the data in the database. If either field is empty an error is displayed.



If either field is incorrect, another error is displayed.

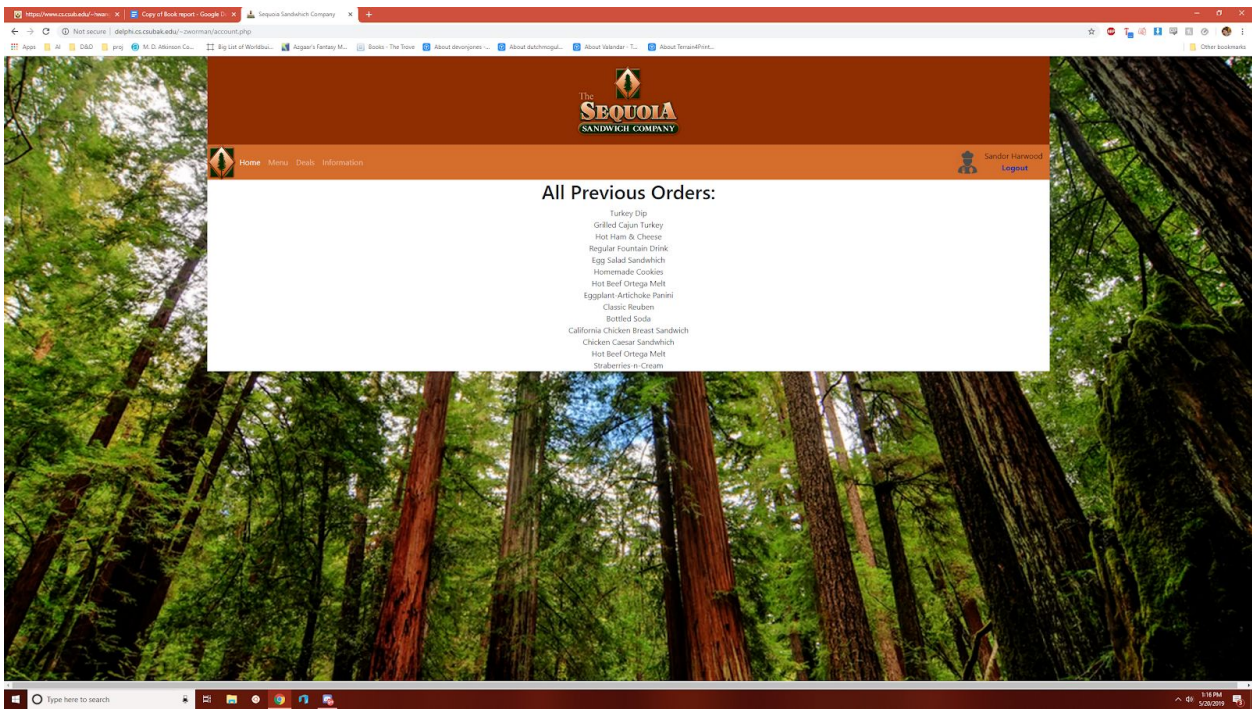


On a successful log in, the name of the user is displayed instead of the “Your Account” fill in.

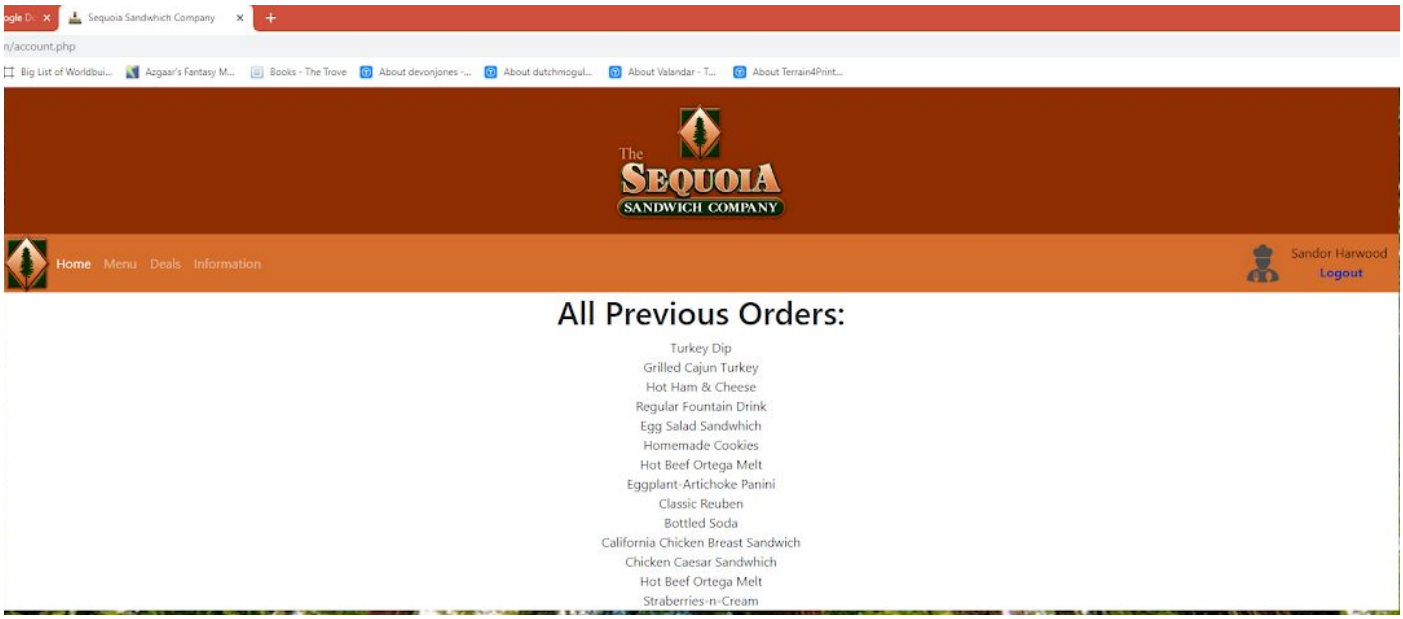


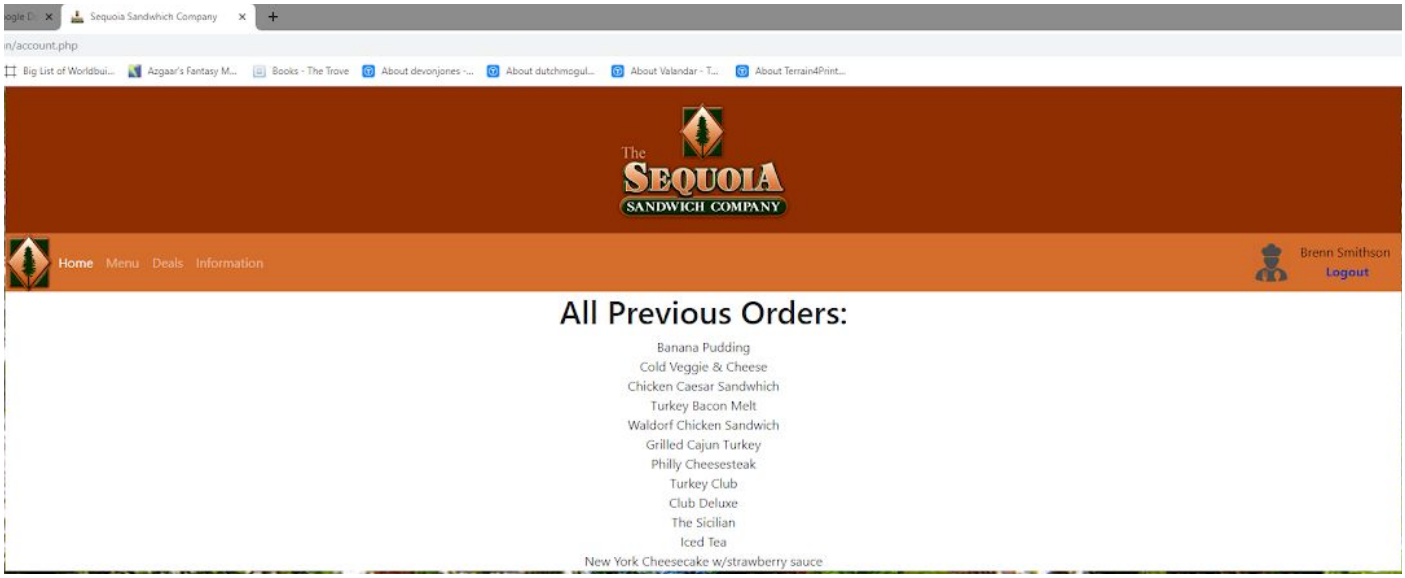
The logout button ends the session and send the signed out user back to the Home page.

If the user clicks their name, they are brought to their account page.



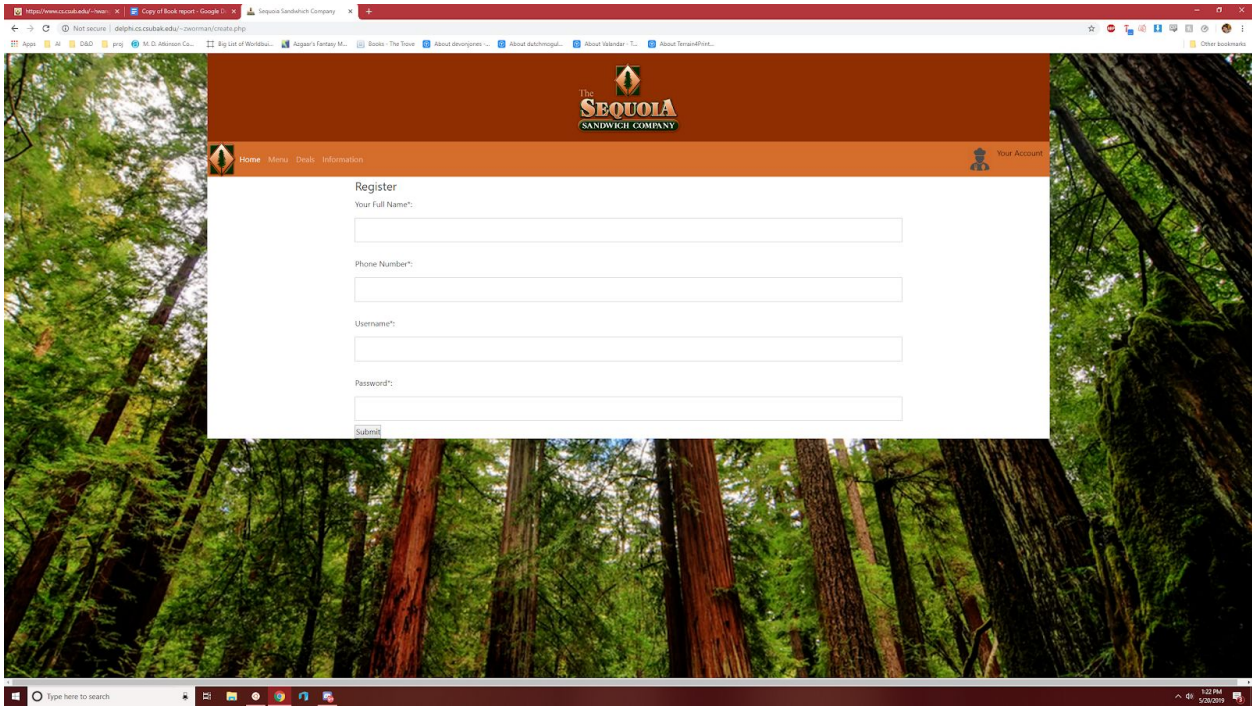
This page displays all their previous purchases. As can be seen when I sign in with a different user.





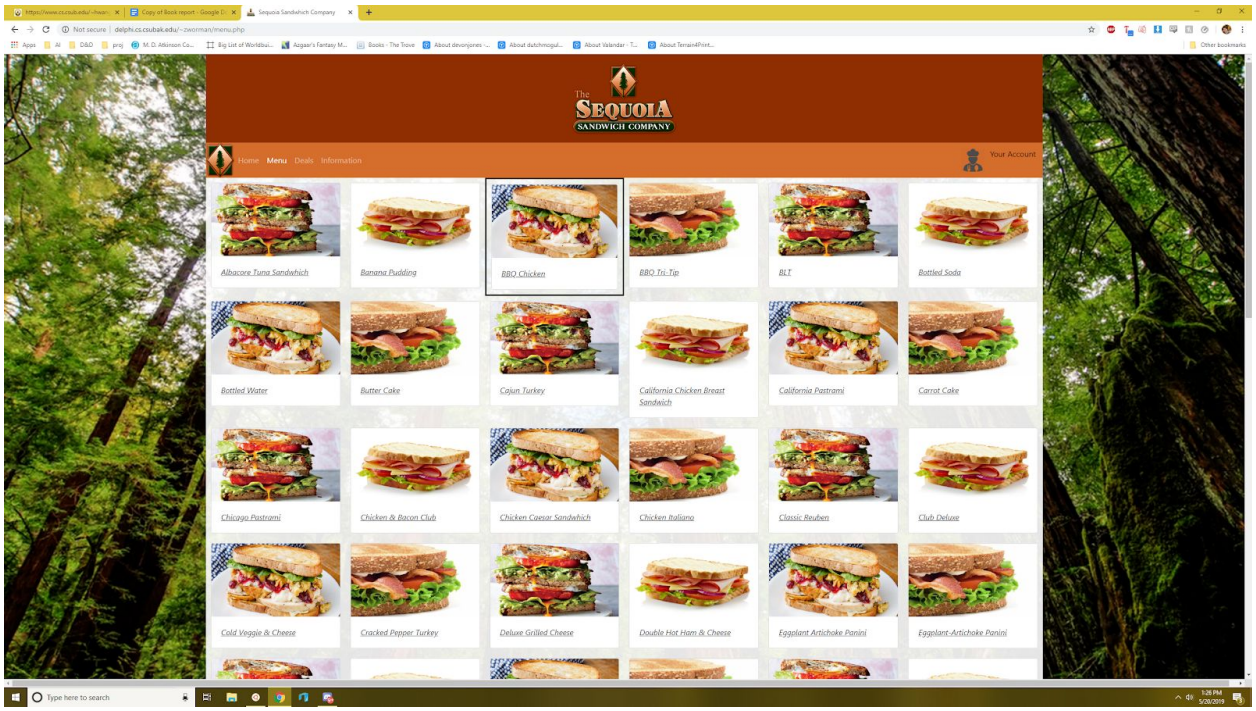
Different users, different names, and different order history. This is true for the 200 customer accounts in the database and the 2000+ orders saved in the database.

Going back to the login modal. If the user is not in the database, they can click register to begin the process of making an account.

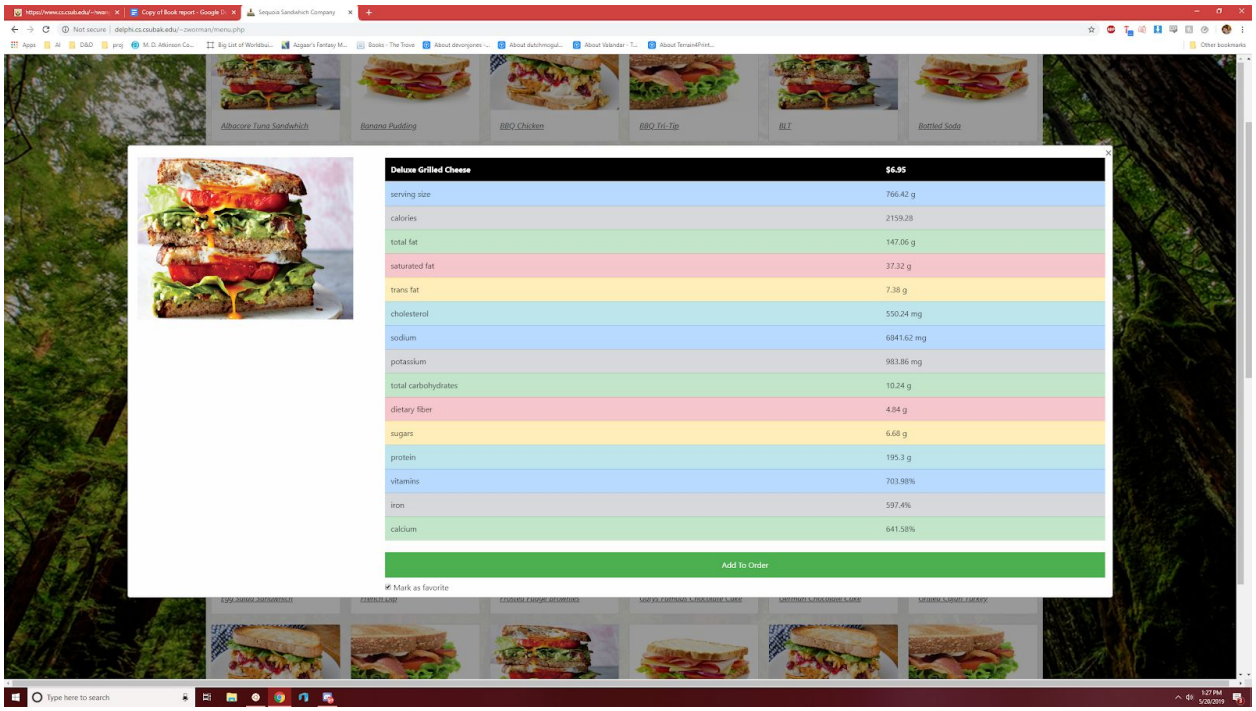


Once all the fields are filled in and the submit button is clicked, that user will be added to the database and signed in immediately.

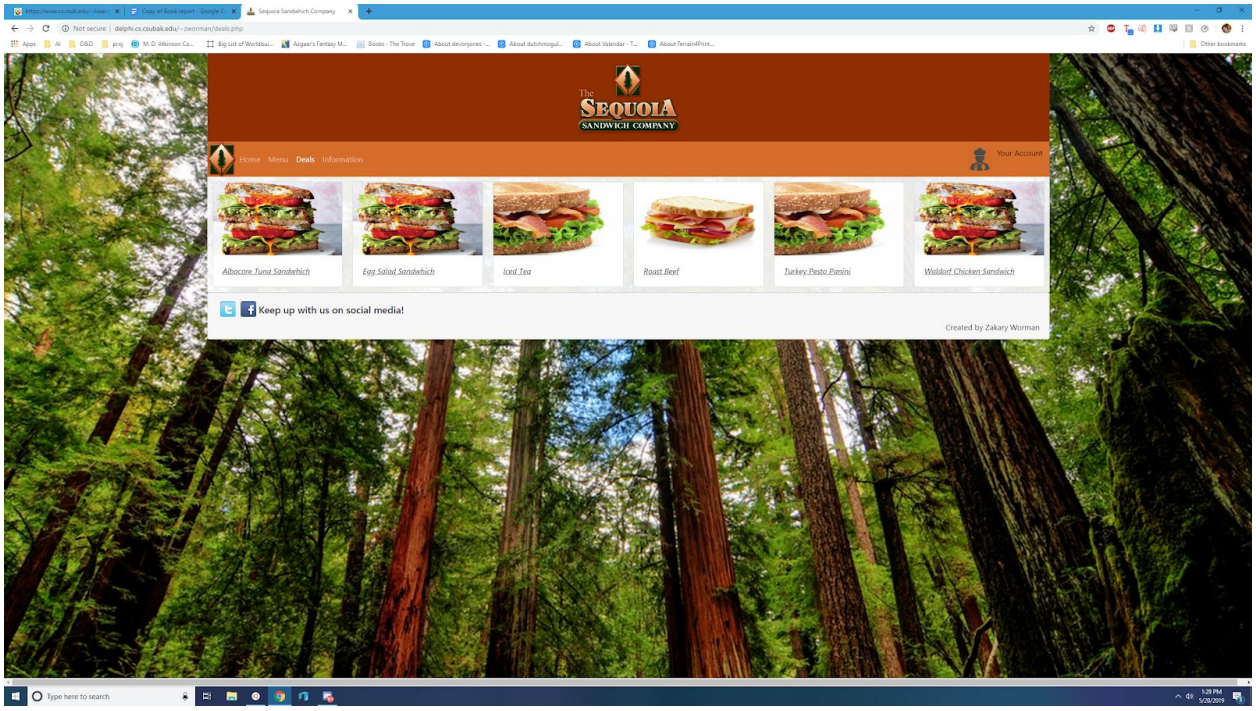
As with the sign in, all the fields are required and will display an error if a person does not fill them all out.



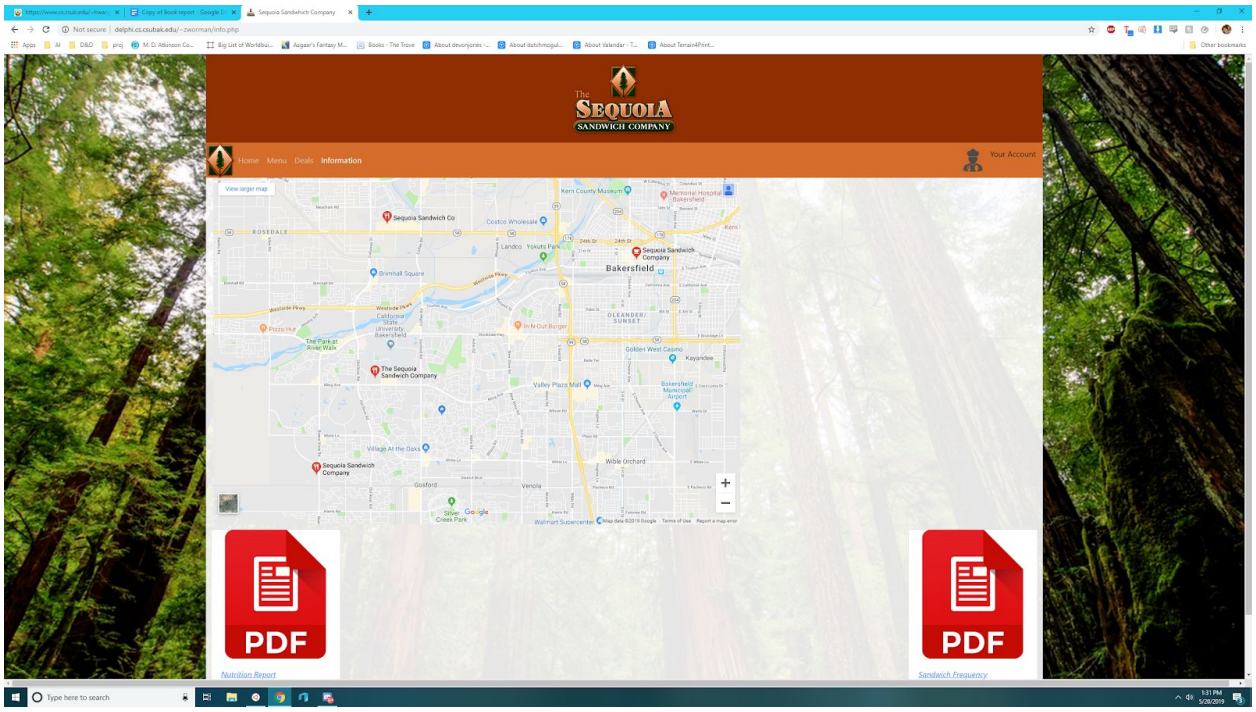
The menu page shows all 64 menu items, and the one being hovered by the mouse is given a border. This allows the user to see which one will be selected when they click.



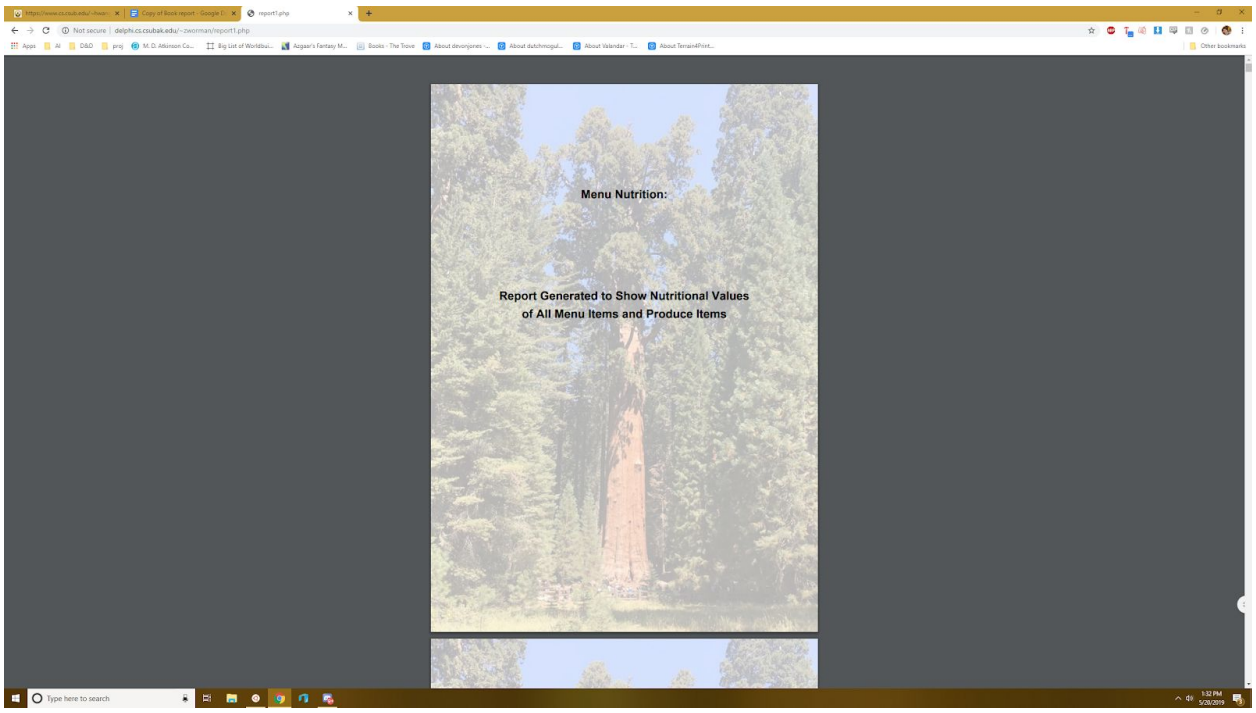
Once an item is clicked, the values retrieved from the database are presented to the user. Every item has a number of relations to the produce items that make it up, and every produce item has 15 relations to the nutritional attributes of each item. Therefore, each data present on the screen is a relation of 5 different tables. This was made into a view for easier access.



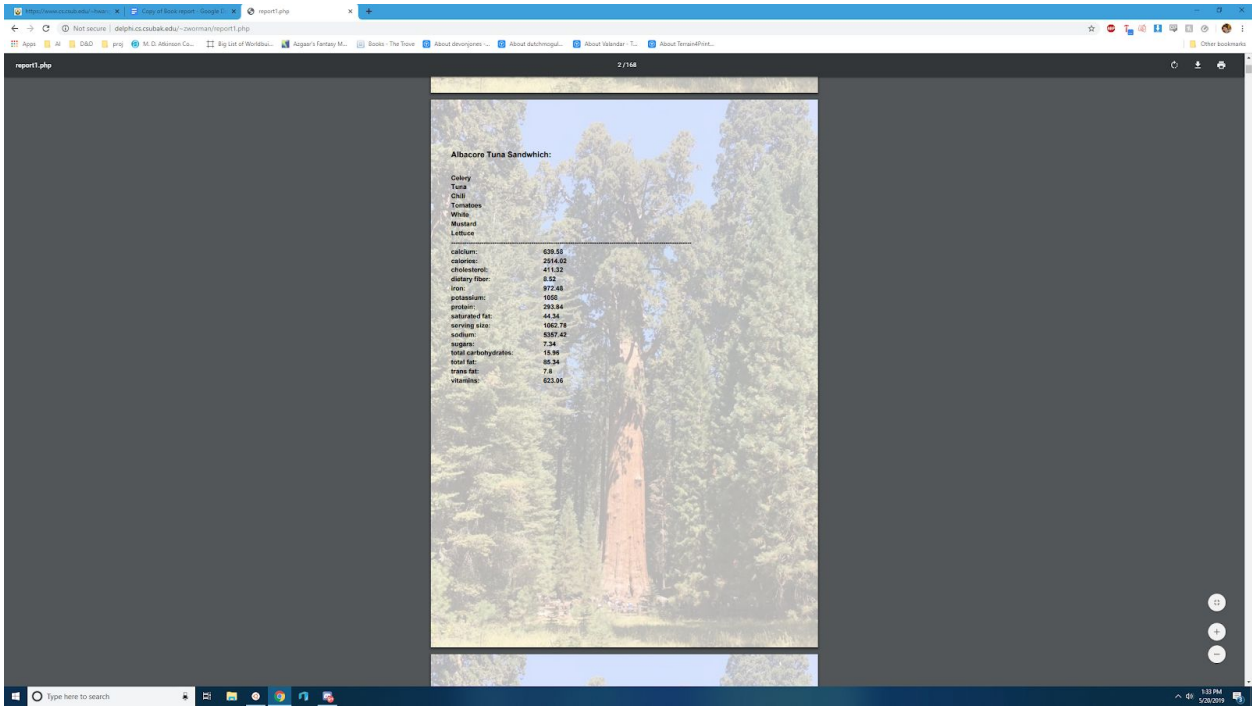
The deals page shows randomly selected items to be on sale for the user. They also have a modal that works the exact same as the one for the menu items.



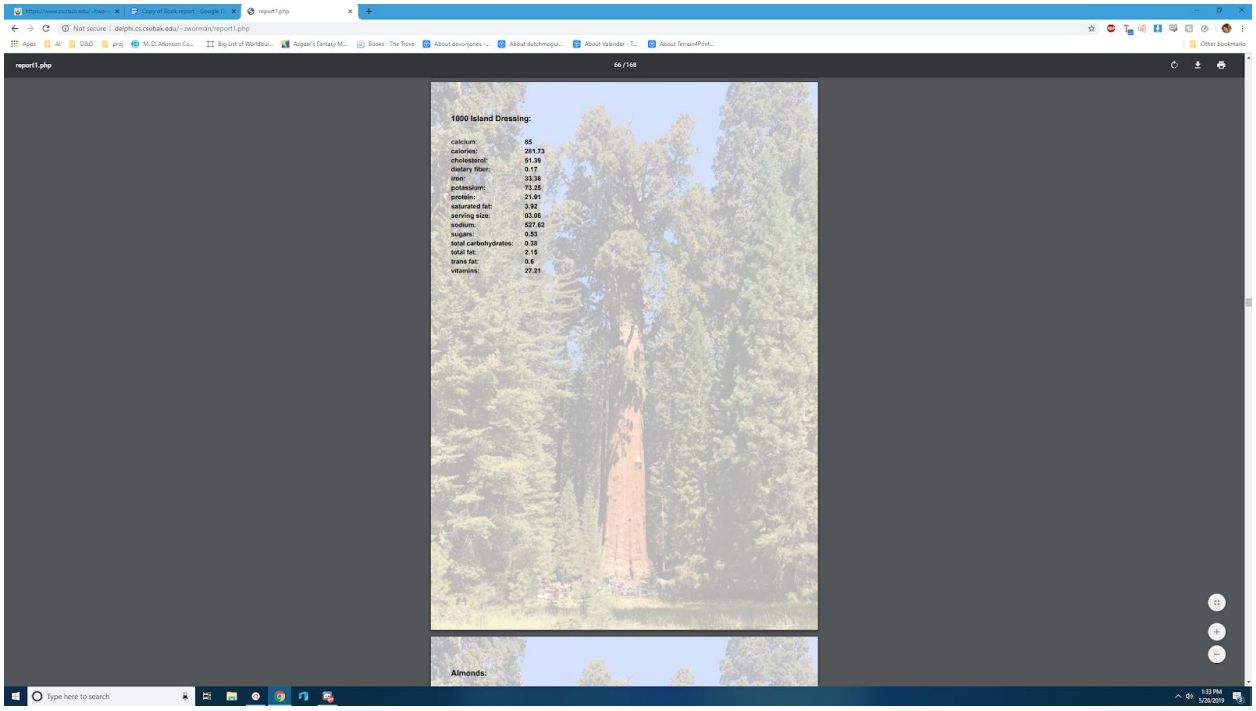
The information page displays the map with every location of a Sequoia Sandwich Company Restaurant on it. Below that are the links to my two reports.



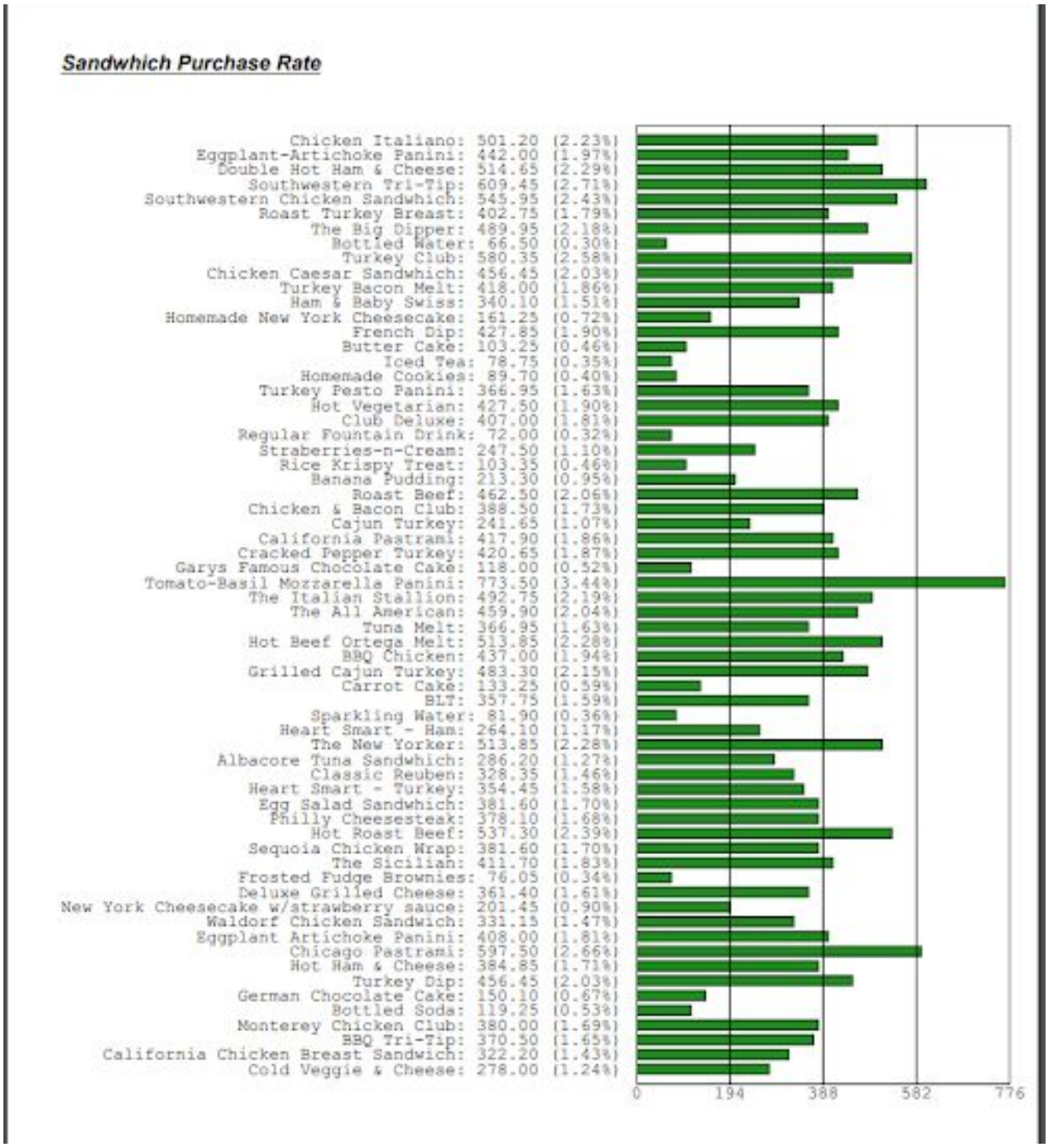
Report 1 displays the nutritional values of every menu item:



And every produce item:


report1.php
66 /168
1000 Island Dressing:
calcium: 85
calories: 281.73
cholesterol: 51.39
dietary fiber: 0.17
iron: 33.38
potassium: 73.25
protein: 21.91
saturated fat: 3.92
serving size: 93.06
sodium: 927.62
sugars: 0.53
total carbohydrates: 0.39
total fat: 2.15
trans fat: 0.6
vitamins: 27.21
Almonds:

The Second Report shows the frequency that each sandwich as purchase as a bar graph.



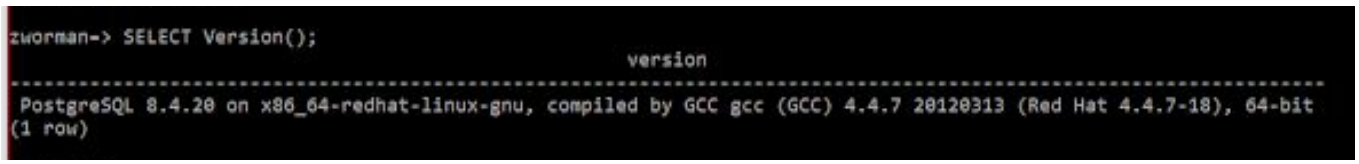
The last thing to add is the links to the company's social media. This is present at the footer of each webpage.

Each logo is a button that links to the social media page.

5.1.3 Views, Tables, and Triggers

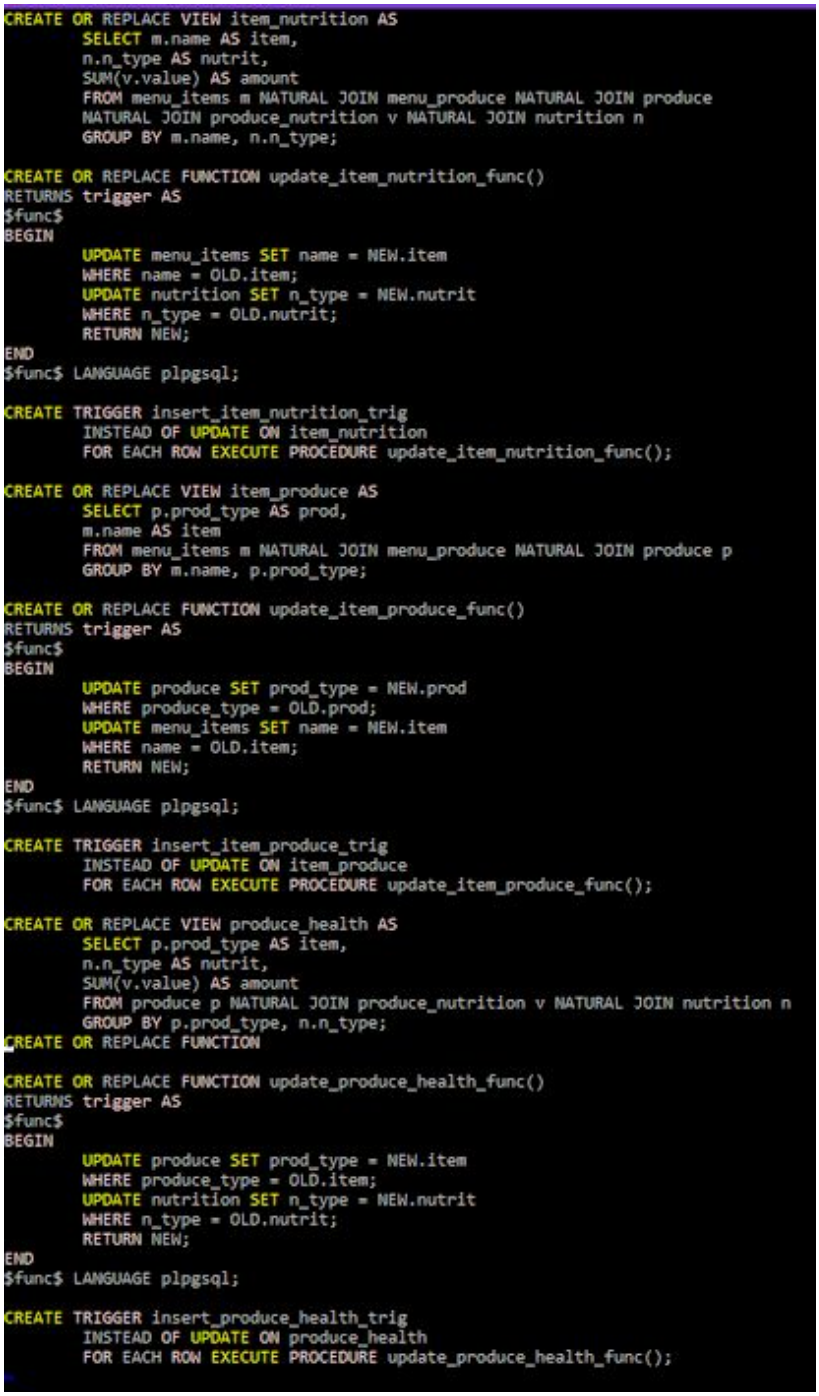
I used views to display the numerous natural joins required to display all the nutritional values from produce to menu items. I created triggers and functions for each of these to allow updating from the view to the tables that they call.

Because the Delphi server was only running version 8.4.1, I was unable to use procedures.



5.2 Programming

5.2.1 Server-Side Programming



```
CREATE OR REPLACE VIEW item_nutrition AS
        SELECT m.name AS item,
        n.n_type AS nutrit,
        SUM(v.value) AS amount
        FROM menu_items m NATURAL JOIN menu_produce NATURAL JOIN produce
        NATURAL JOIN produce_nutrition v NATURAL JOIN nutrition n
        GROUP BY m.name, n.n_type;

CREATE OR REPLACE FUNCTION update_item_nutrition_func()
RETURNS trigger AS
$func$
BEGIN
        UPDATE menu_items SET name = NEW.item
        WHERE name = OLD.item;
        UPDATE nutrition SET n_type = NEW.nutrit
        WHERE n_type = OLD.nutrit;
        RETURN NEW;
END
$func$ LANGUAGE plpgsql;

CREATE TRIGGER insert_item_nutrition_trig
        INSTEAD OF UPDATE ON item_nutrition
        FOR EACH ROW EXECUTE PROCEDURE update_item_nutrition_func();

CREATE OR REPLACE VIEW item_produce AS
        SELECT p.prod_type AS prod,
        m.name AS item
        FROM menu_items m NATURAL JOIN menu_produce NATURAL JOIN produce p
        GROUP BY m.name, p.prod_type;

CREATE OR REPLACE FUNCTION update_item_produce_func()
RETURNS trigger AS
$func$
BEGIN
        UPDATE produce SET prod_type = NEW.prod
        WHERE produce_type = OLD.prod;
        UPDATE menu_items SET name = NEW.item
        WHERE name = OLD.item;
        RETURN NEW;
END
$func$ LANGUAGE plpgsql;

CREATE TRIGGER insert_item_produce_trig
        INSTEAD OF UPDATE ON item_produce
        FOR EACH ROW EXECUTE PROCEDURE update_item_produce_func();

CREATE OR REPLACE VIEW produce_health AS
        SELECT p.prod_type AS item,
        n.n_type AS nutrit,
        SUM(v.value) AS amount
        FROM produce p NATURAL JOIN produce_nutrition v NATURAL JOIN nutrition n
        GROUP BY p.prod_type, n.n_type;
CREATE OR REPLACE FUNCTION

CREATE OR REPLACE FUNCTION update_produce_health_func()
RETURNS trigger AS
$func$
BEGIN
        UPDATE produce SET prod_type = NEW.item
        WHERE produce_type = OLD.item;
        UPDATE nutrition SET n_type = NEW.nutrit
        WHERE n_type = OLD.nutrit;
        RETURN NEW;
END
$func$ LANGUAGE plpgsql;

CREATE TRIGGER insert_produce_health_trig
        INSTEAD OF UPDATE ON produce_health
        FOR EACH ROW EXECUTE PROCEDURE update_produce_health_func();
```

Used to find the nutritional values for each menu item.

Function to update tables if the view is updated.

Trigger to run function on view update.

View to show each produce item used in each menu item.

Function to update tables from view.

Trigger to run on view update.

View to find produce nutritional facts.

Function to update tables from view.

Trigger to run when view is updated.

5.2.2 Mid-Tier Programming

```php
<?php
        $dbconn = pg_connect("host=localhost dbname=zworman user=zworman password=Cis4-nay");
?>
```

This line was used in a config.php to be able to call it from any function to easily begin the connection to the database.


```php
<?php
session_start();
if(isset($_SESSION["loggedin"]) && $_SESSION["loggedin"] == true) {
    header("location: index.php");
    exit;
}

require_once "config.php";

$username = $password = "";
$username_err = $password_err = "";

if($_SERVER["REQUEST_METHOD"] == "POST"){
    $username = trim($_POST["username"]);

    $password = trim($_POST["password"]);

    if($username_err == false && $password_err == false){
        $stmt = "SELECT name, username, password, customer_num FROM customer WHERE username = '";
        $stmt .= $username;
        $stmt .= "';";
        $result = pg_query($dbconn, $stmt);
        if(pg_num_rows($result) == 1) {
            $arr = pg_fetch_array($result);
            if($password == $arr['password']) {
                session_start();
                $_SESSION["loggedin"] = true;
                $_SESSION["id"] = $arr['name'];
                $_SESSION["username"] = $arr['username'];
                $_SESSION["user"] = $arr['customer_num'];
                header("location: index.php");
                exit;
            } else {
                $password_err = "Invalid Password";
            }
        } else {
            $username_err = "No account found with that username";
        }
    } else {
        echo "Something went wrong. Please try again";
    }
    pg_close($db_connect);
    $error = $username_err . " " . $password_err;
    echo "<script type='text/javascript'>";
    echo "alert('$error');";
    echo 'window.location.href="index.php";';
    echo "</script>";
    exit;
}
?>
```

This code is to check to make sure that a person's imputed username and password are in the database.

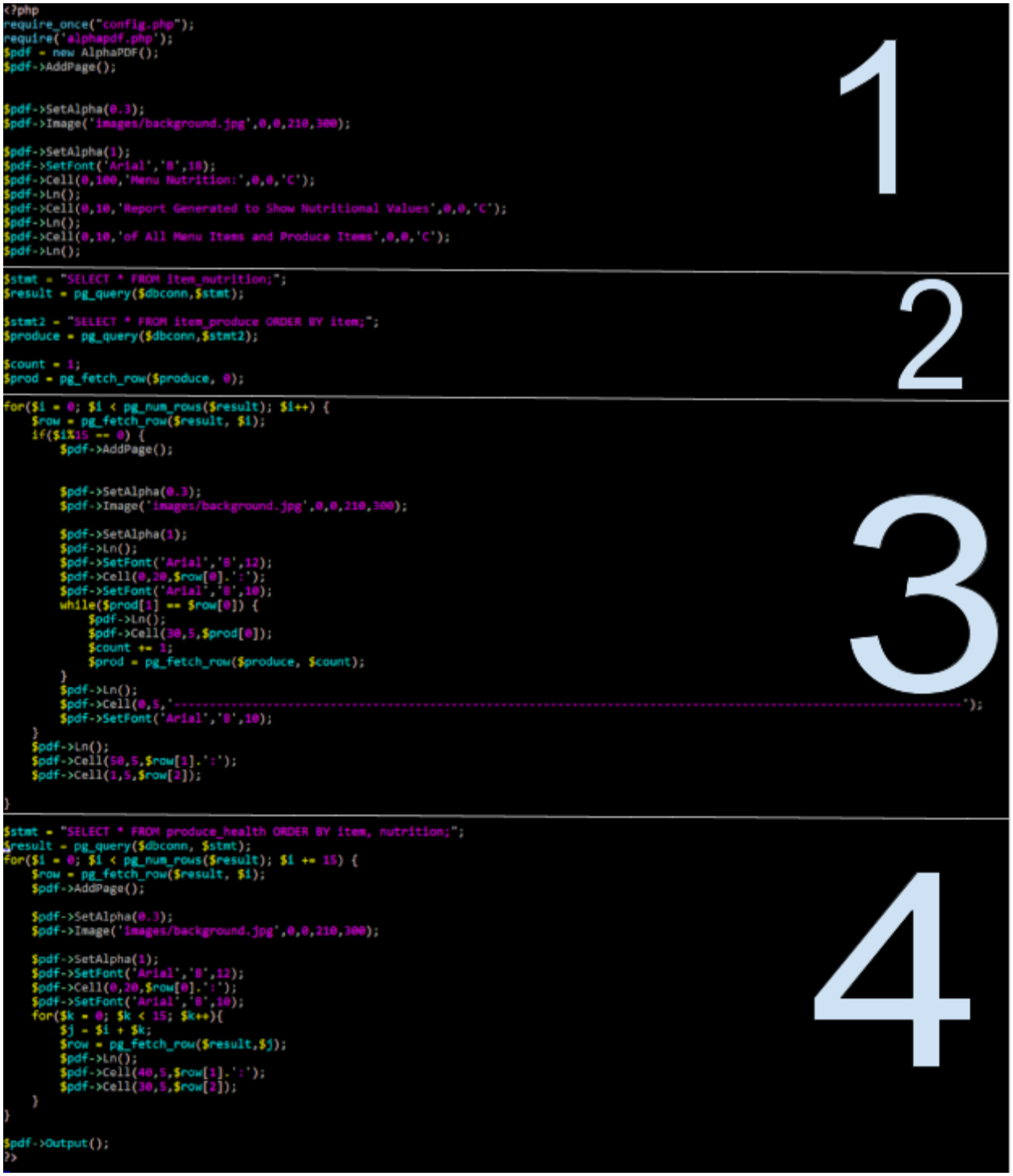
1) Checks to see if the user is already logged in, and if so, they are pushed back to the main page
2) Starts connection to the database, creates variables for later use, and get the posted username and password
3) As long as there are no errors, the database is queried to retrieve the name, username, and password given the username inputted in the login field
4) As long as there are not multiple users with the same username, the password submitted is checked against the password saved with the given username. If they are the same, then the SESSION variables are set and the user is pushed back to the main page
5) If the password was invalid an error is loaded into the preset variables
6) If the username was incorrect, there is an error loaded into the variables
7) If anything else happened to prevent the successful loading of the user account, an error message is displayed.
8) The database connection is closed, any error messages are printed out for the user to see, and the window is set back to the main home page.

```php
<?php

require_once('config.php');

if($_SERVER["REQUEST_METHOD"] == "POST"){
    $username = trim($_POST["username"]);

    $password = trim($_POST["password"]);

    $name = trim($_POST["name"]);

    $phone_number = trim($_POST["phone"]);
    $stmt = "SELECT name, username, password, customer_num FROM customer;";
    $result = pg_query($dbconn, $stmt);
    $num = pg_num_rows($result)+1;
    $stmt = "INSERT INTO customer VALUES(".$num.",'".$phone_number."','".$name."','".$username."','".$password."');";
    $result = pg_query($dbconn, $stmt);
    $arr = pg_fetch_array($result);
    session_start();
    $_SESSION["loggedin"] = true;
    $_SESSION["id"] = $name;
    $_SESSION["username"] = $username;
    $_SESSION["user"] = $num;
    header("location: index.php");
    pg_close($db_connect);
    $error = $username_err . " " . $password_err;
    echo "<script type='text/javascript'>";
    echo "alert('$error');";
    echo 'window.location.href="index.php";';
    echo "</script>";
    exit;
}
?>
```

This one is very similar to the one above to sign a user in, except this is to register a user.

Once a connection is made to the database, the posted variables are loaded into variables in php. The database is queried to get the number of rows. This is incremented

to set the user's number. A new insert statement is made that submits everything to the database, the user is signed in by setting the SESSION variables, the database connection is dropped, and the user is pushed back to the main page.




```php
<?php
require_once("config.php");
require('alphapdf.php');
$pdf = new AlphaPDF();
$pdf->AddPage();


$pdf->SetAlpha(0.3);
$pdf->Image('images/background.jpg',0,0,210,300);

$pdf->SetAlpha(1);
$pdf->SetFont('Arial','B',18);
$pdf->Cell(0,100,'Menu Nutrition:',0,0,'C');
$pdf->Ln();
$pdf->Cell(0,10,'Report Generated to Show Nutritional Values',0,0,'C');
$pdf->Ln();
$pdf->Cell(0,10,'of All Menu Items and Produce Items',0,0,'C');
$pdf->Ln();
```

```php
$stmt = "SELECT * FROM item_nutrition;";
$result = pg_query($dbconn,$stmt);

$stmt2 = "SELECT * FROM item_produce ORDER BY item;";
$produce = pg_query($dbconn,$stmt2);

$count = 1;
$prod = pg_fetch_row($produce, 0);
```

```php
for($i = 0; $i < pg_num_rows($result); $i++) {
    $row = pg_fetch_row($result, $i);
    if($i%15 == 0) {
        $pdf->AddPage();


        $pdf->SetAlpha(0.3);
        $pdf->Image('images/background.jpg',0,0,210,300);

        $pdf->SetAlpha(1);
        $pdf->Ln();
        $pdf->SetFont('Arial','B',12);
        $pdf->Cell(0,20,$row[0].':');
        $pdf->SetFont('Arial','B',10);
        while($prod[1] == $row[0]) {
            $pdf->Ln();
            $pdf->Cell(30,5,$prod[0]);
            $count += 1;
            $prod = pg_fetch_row($produce, $count);
        }
        $pdf->Ln();
        $pdf->Cell(0,5,'----------');
        $pdf->SetFont('Arial','B',10);
    }
    $pdf->Ln();
    $pdf->Cell(50,5,$row[1].':');
    $pdf->Cell(1,5,$row[2]);

}
```

```php
$stmt = "SELECT * FROM produce_health ORDER BY item, nutrition;";
$result = pg_query($dbconn, $stmt);
for($i = 0; $i < pg_num_rows($result); $i += 15) {
    $row = pg_fetch_row($result, $i);
    $pdf->AddPage();

    $pdf->SetAlpha(0.3);
    $pdf->Image('images/background.jpg',0,0,210,300);

    $pdf->SetAlpha(1);
    $pdf->SetFont('Arial','B',12);
    $pdf->Cell(0,20,$row[0].':');
    $pdf->SetFont('Arial','B',10);
    for($k = 0; $k < 15; $k++){
        $j = $i + $k;
        $row = pg_fetch_row($result,$j);
        $pdf->Ln();
        $pdf->Cell(40,5,$row[1].':');
        $pdf->Cell(30,5,$row[2]);
    }
}

$pdf->Output();
?>
```

This code is used to create report 1.

1) The fpdf system is established, and a new alph pdf for creating transparent images is created. A universal background image is loaded to the first page, and a title put on the page as well.
2) The database is queried to set up variables to be used later.
3) For every element in the initial query statement, a new page is made where the background is set, then the label. A while loop is then ran on the second query statement to insert detailed information along the next couple rows in the pdf. Finally, the remaining values are printed out for the item.
4) Another query is made and stored. Another for loop is used to go through every entree in the query, and it is outputted to the screen after setting the background image.

```php
<?php
require_once("config.php");
require('diag.php');

$pdf = new PDF_Diag();
$pdf->AddPage();

$stmt = "SELECT name, sum(price) from c_order NATURAL JOIN menu_items GROUP BY name;";
$result = pg_query($dbconn, $stmt);

$data = array();
for($i = 0; $i < pg_num_rows($result); $i++) {
    $row = pg_fetch_row($result);
    $data += array($row[0] => number_format($row[1], 2));
}

$pdf->SetFont('Arial', 'BIU', 12);
$pdf->Cell(0, 5, 'Sandwhich Purchase Rate', 0, 1);
$pdf->Ln(8);
$valX = $pdf->GetX();
$valY = $pdf->GetY();
$pdf->BarDiagram(200,260, $data, "%l:\t%v\t(%p)", array(34,139,34));
$pdf->SetXY($valX, $valY + 80);

$pdf->Output();
?>
```

This creates report 2. It takes all the data from the query result and stores it into an array. It then passes that data to a function that outputs it into grouped bars that represent the frequency of their usage.

```php
<?php
session_start();
$account = "";
if(isset($_SESSION["loggedin"])) {
    $account = $_SESSION["id"];
} else {
    $account = "Your Account";
}
?>
```

This code is at the top of every single webpage in order to see if the user is logged in or not.

```php
<?php
require_once "config.php";
$stmt = "SELECT name,price FROM menu_items ORDER BY name;";
$result = pg_query($dbconn,$stmt);

$stmt2 = "SELECT * FROM item_nutrition;";
$nutrition = pg_query($dbconn,$stmt2);
for($i = 0; $i < pg_num_rows($result); $i += 1) {
    $val = pg_fetch_result($result,$i,0);
    $price = pg_fetch_result($result,$i,1);
    echo'<div class="col-sm-2" onmouseover="this.style.border=\'solid #000000\'" onmouseout="this.style.border=\'none\'"style="padding: 10px;" onclick="document.getElementById(\'menu'.$i.'\').style.display=\'block\'">
        <div class="card">
        <img class="card-img-top" src="./images/sandwich'.($i%4).'.jpg" alt="Card image cap" style="height: 150px;">
        <div class="card-body"><u><i>';
    echo $val;
    echo
        '</i></u></div>
        </div>
        </div>';
    echo '<div id="menu'.$i.'" class="modal">
        <form class="modal-content animate" action="" method="post">
            <span onclick="document.getElementById(\'menu'.$i.'\').style.display=\'none\'" class="align-self-end close" title="Close Modal">&times;</span>
            <div class="row">
            <div class="col-sm-3" style="min-height: 100%">
                <img src="./images/sandwich'.($i%4).'.jpg" class="img-fluid align-self-start d-none d-lg-block" style="margin: 0; padding: 20px; padding-top: 0;" alt="Avatar">
            </div>

            <div class="col-lg-9 col-sm-0">
                <div class="container-fluid">
                    <table class="table table-xs order-0">
                        <thead>
                        <tr style="color: white; background-color: black;">
                            <th scope="col">'.$val.'</th>
                            <th scope="col"></th>
                            <th scope="col"></th>
                            <th scope="col"i>$'.number_format($price,2).'</th>
                        </tr>
                        </thead>
                        <tbody>
                        <tr class="table-primary">
                            <td scope="row">'.pg_fetch_result($nutrition,$i*15+8,1).'</td>
                            <td></td>
                            <td></td>
                            <td>'.pg_fetch_result($nutrition,$i*15+8,2).' g</td>
                        </tr>
                        <tr class="table-secondary">
                            <td scope="row">'.pg_fetch_result($nutrition,$i*15+1,1).'</td>
                            <td></td>
                            <td></td>
                            <td>'.pg_fetch_result($nutrition,$i*15+1,2).'</td>
                        </tr>
                        <tr class="table-success">
                            <td scope="row">'.pg_fetch_result($nutrition,$i*15+12,1).'</td>
                            <td></td>
                            <td></td>
                            <td>'.pg_fetch_result($nutrition,$i*15+12,2).' g</td>
                        </tr>
                        <tr class="table-danger">
                            <td scope="row">'.pg_fetch_result($nutrition,$i*15+7,1).'</td>
                            <td></td>
                            <td></td>
                            <td>'.pg_fetch_result($nutrition,$i*15+7,2).' g</td>
                        </tr>
                        <tr class="table-warning">
                            <td scope="row">'.pg_fetch_result($nutrition,$i*15+13,1).'</td>
                            <td></td>
                            <td></td>
                            <td>'.pg_fetch_result($nutrition,$i*15+13,2).' g</td>
                        </tr>
                        <tr class="table-info">
                            <td scope="row">'.pg_fetch_result($nutrition,$i*15+2,1).'</td>
                            <td></td>
                            <td></td>
                            <td>'.pg_fetch_result($nutrition,$i*15+2,2).' mg</td>
                        </tr>
                        <tr class="table-primary">
                            <td scope="row">'.pg_fetch_result($nutrition,$i*15+9,1).'</td>
                            <td></td>
                            <td></td>
                            <td>'.pg_fetch_result($nutrition,$i*15+9,2).' mg</td>
                        </tr>
                        <tr class="table-secondary">
                            <td scope="row">'.pg_fetch_result($nutrition,$i*15+5,1).'</td>
                            <td></td>
                            <td></td>
                            <td>'.pg_fetch_result($nutrition,$i*15+5,2).' mg</td>
```

These 2 pictures show how the menu was created. An in-line html and javascript is used.

First, the database queries for the view I created previously. It then uses echo to create the html for the cards that are displayed. A unique card is created for each menu item as the array loops through all the inputs. Then, each card is given an html case of onmouseover and onmouseout to create a border around the card being hovered by the mouse. Once clicked, a modal is opened that is unique to that menu item because of this loop. There could be 1000 menu items, and each would be given its own, unique modal.

The deals webpage section used a similar method except instead of looping through the entire view, it picked random numbers to use.

5.2.3 Client-Side Programming

```
<script>
var modal = document.getElementById('login');
var login = "<?php echo $_SESSION['loggedin'] ?>";
// When the user clicks anywhere outside of the modal, close it
window.onclick = function(event) {
    if (event.target.tagName == 'A' && login == true) {
        window.location.href = "account.php";
    }
    if (event.target == modal || login == true) {
        modal.style.display = "none";
    }
}
</script>
```

The first event of this section is used to check to see if the user is logged in and that they are trying to access their account page. If they are, it relocates the browser to the account page.

The second event handler is used to display and to hide the modals.

5.3 Survey

Outcome	Response (1-10)
An ability to analyze a problem, and identify and define the computing requirements and specifications appropriate to its solution.	9
An ability to design, implement and evaluate a computer-based system, process, component, or program to meet your desired needs. An ability to understand the analysis, design, and implementation of a computerized solution to a real-life problem.	9
An ability to communicate effectively with a range of audiences. An ability to write a technical document such as a software	8

specification white paper or a user manual.	
An ability to apply mathematical foundations, algorithmic principles, and computer science theory in the modeling and design of computer-based systems in a way that demonstrates comprehension of the tradeoffs involved in design choices.	9