Table of Contents

Phase 1 - Conceptual Design
1.1 - Fact-Finding Techniques and Information Gathering 4
   1.1.1 - Introduction to the Enterprise/Organization 4
   1.1.2 - Fact Finding Techniques 4
   1.1.3 - The Miniverse of Interest 4
   1.1.4 - Itemized Description of Entity sets and Relationship sets. 5
   1.1.5 - User Groups, Data Views, and Operations. 10
1.2 - Conceptual Database Design. 10
   1.2.1 - Entity Type Descriptions 10
   1.2.2 - Relationship Type Description 17
   1.2.3 - Related Entity Types 19
   1.2.4 - ER Diagram 20

Phase 2 - Conversion from Conceptual to Relational 21
2.1 - The ER and Relational Models 21
   2.1.1 - Descriptions of ER and Relational Models 21
   2.1.2 - Model Comparisons 23
2.2 - Conceptual to Logical Conversion Process 23
   2.2.1 - Converting Entity Types to Relations 23
   2.2.2 - Converting Relationship Types to Relations 24
   2.2.3 - Converting Extended Types to Relations 25
   2.2.4 - Database Constraints 26
2.3 - Results of ER to Relational Conversion 26
   2.3.1 - Relation Schema 26
   2.3.2 - Sample Data 29
2.4 - Sample Queries 35
   2.4.1 - Design of Queries 35
   2.4.2 - Relational Algebra Expressions 36
   2.4.3 - Relational Calculus Expressions 38

Phase 3 - Physical Implementation 39
3.1 - Relational Normalization 39
   3.1.1 - Normalization process 39
   3.1.2 - Application to Relational Model 40
3.2 - Database Implementation 44
   3.2.1 - Background Information 44
   3.2.2 - Schema and Hosting 44
3.3 - Query Implementation 46
Phase 1 - Conceptual Design

1.1 - Fact-Finding Techniques and Information Gathering

1.1.1 - Introduction to the Enterprise/Organization

We have been hired by, “Yummers,” to create a database system. They have chosen Bakersfield, CA, for their first location. They feel that their ice cream is the perfect antidote for people to fight off the long, hot summers. The enterprise is an ice cream parlor that provides many flavors of ice cream along with a choice of various toppings including fruits and candy. The ambiance is designed to provide a family friendly atmosphere with incentives through their point system to encourage repeat customers.

1.1.2 - Fact Finding Techniques

Yummers is a fictitious brand, so many of our “facts,” will come from our imagination. We will be implementing ideas we have experienced from our own visits to various ice cream and yogurt shops. Of course, in the next few weeks we will have to use our education expense accounts to remind ourselves of what ice cream and yogurt taste like as well as collect data from the menus regarding flavors and prices. In addition to visiting some of the local ice creameries in town, we will search the internet for both wholesale and retail price points for our various products.

1.1.3 - The Miniverse of Interest

Our mini-verse of interest centers on the store. Other significant entities include Employee, Customer, and Product. Employee, “Works At,” the store, which relates the Employee to the Store where they work. The Store “Serves,” the Customer relationship links the Store to the Customer with, “Serves,” having attributes keeping track of the payment received and the product given during the service transaction. The Store Sells Product connects the Store to the Products. The Products entity will keep track of products that are ordered and received as well as products that are sold to the customers.
1.1.4 - Itemized Description of Entity sets and Relationship sets.

**Store:**

The Store entity includes 6 attributes with, “Store ID,” being the unique attribute.

- **Location:** String  
  Location will hold the address of the store.

- **Store ID:** Int  
  This is the unique identifier for the Store entity.

- **Operating Cost:** Decimal  
  This attribute represents the total expenses for the store. This includes mock values for rent, electricity, product costs, etc.

- **Sales:** Decimal  
  This number represents the total sales from the sale of Product from the Store to the customers and is derived from the Store serving the Customers.

- **Gross Profit:** Decimal  
  This attribute represents the gross profit of the company and can be derived from the Sales attribute and cost of the wholesale Product attribute.

- **Net Profit:** Decimal  
  This attribute represents the net profit of the company and can be derived from the Operating Cost and Gross Profit attributes.

Yummers currently has only one store but they want to expand, so we created the database model with the ability to expand for this eventuality.

**Examples:**

1. Sally Smith Works At Store 1.
2. Store1 Sells Product 213
3. Store1 Serves Customer SallySmith31@gmail.com
**Product:**

The, “Product,” entity has 6 attributes with, “Product ID,” being the unique identifier.

- **Product ID:** Int  
  Each product will be given an ID number for the purpose of having a unique key identifier, distinguishing flavors.

- **Price:** Decimal  
  Keeps track of the price that customers pay for the product.

- **Cost:** Decimal  
  The price that the store pays for the product.

- **Name:** String  
  Gives a name to each product for easier human identification.

**Example:**

1. Store 1 Sells Product 123

**Customer:**

The “Customer,” entity has 4 attributes with, “CID,” being the unique identifier.

- **CID:** Int  
  CID is customer ID and is the unique identifying attribute.

- **Name:** String  
  Provides the name of the customer for communication.

- **Visits:** Int  
  Keeps track of the number of times the customer has visited for future rewards.

- **Email:** String  
  Holds the email for the customer and can be a unique identifying attribute.

**Example:**

Store 1 Serves Customer 1234

**Employee:**

The, “Employee,” entity has 4 attributes with, “Employee ID,” being the unique identifier.
**Employee ID:** Int
The unique identifier for the employee.

**Wage:** Decimal
Keeps track of the current wage of the employee.

**Name:** String
Can be used to identify Employee but may not be unique.

**Hours Worked:** Decimal
Provides the numbers of hours worked.

Example: Sally Smith Works At Store 1.

**Inventory:**

The weak “Inventory” entity depends on Store and has 0 attributes yet, “Store ID” is also its identifier.

**Relationships:**

**Works At:** Store (1,1) Works At (1,M) Employee.
This relationship relates the Employees to which Store they work.

Ex: Sally Smith works At Store 1.

**Carries:** Store (1,1) Carries (1,1) Inventory
This relationship relates how a store only has one Inventory.

Ex: Store 3 Carries Inventory 3.

**Contains:** Inventory(1,M) Contains (1,M) Product
This relationship states how an inventory can contain many products and one product can be in many inventories.

Ex: Inventory 3 contains Product 54 and 65.

**Contains Attributes:**

**Quantity:** Int
A non-unique attribute that holds the number of containers left in inventory. One container holds around 40 scoops of ice cream.
Type = Int  Range = (0-999)  Default = 0
Unique = False  Single Simple Value  Key Candidate = False

Expired: Bool
A derived variable that is false until the expiration date is less than the current date, meaning that the product is expired and should be disposed of.
Type = Bool  Range = (0-1)  Default = False
Unique = False  Single Simple Value  Key Candidate = False

Expiration: String
A non-unique attribute that holds the expiration date of each product, as ice cream is perishable. An expiration of null means that the product does not expire, such as with the ice cream containers, as these are usually paper and don’t need to be disposed of.
Type = String  Range = “1/1/2020” - “12/31/9999”  Default = NULL
Unique = False  Single Simple Value  Key Candidate = False

Serves:  
Inventory (1,M) Serves (1,M) Customer.
This relationship relates the Inventory being sold to the Customer who’s buying ice cream.

Ex: Items in Inventory are Served to Customer 1234

Serves Attributes:

Payment Received:
Type = Decimal  range = (0.00,999.99)  Default = 0?  Null not allowed
Unique = no  single value  simple  key candidate = no

Transaction Date:
Type = datetime  range = (022621, 123199) Default = 0000 Null not allowed
Unique = no  single value  simple  key candidate = no
**Transaction ID:**

- Type = Int
- Range = (0000000001,9999999999)
- Default = 0000000001
- Null not allowed
- Unique = yes
- Single Value
- Composite
- key candidate = yes

This ID will count up for every transaction across all stores.

**Order:**

- Type = String
- Range = (1 - 128)
- Default = “ “
- NULL not allowed
- Unique = no
- Multivalue
- Composite
- key candidate = no

**Flavor:**

- Type = String
- Range = (1 - 128)
- Default = “ “
- NULL not allowed
- Unique = no
- Multivalue
- simple
- key candidate = no

**Topping:**

- Type = String
- Range = (1 - 128)
- Default = NULL
- Unique = no
- Multivalue
- simple
- key candidate = no

**Container:**

- Type = String
- Range = (1 - 128)
- Default = “ “
- NULL not allowed
- Unique = no
- Single Value
- Simple
- key candidate = no
1.1.5 - User Groups, Data Views, and Operations.
  ● Workers should have a view to show all Employee attributes, and some Customer attributes, such as, order, purchase history, and first name.
  ● Customers should be able to view the product price, and all the Customer attributes.
  ● Administrator view should have access to all of the information in the database and can make changes to anything.

1.2 - Conceptual Database Design.

1.2.1 - Entity Type Descriptions

Store:

Store is a strong entity:

The purpose of this entity is to keep track of all of the data that occurs in each Store. There is currently only 1 Store, but the company plans to expand. By having a Store entity, we won’t have to update the model when the expansion occurs.

Relationships with Other Entities:

Employee Works At Store.

This relationship keeps track of employees that work at a store. It needs to be updated whenever someone is hired or leaves/quarters.

Store Owns Inventory:

The Inventory is a weak entity to the Store and the Store will always have 1 Inventory whether it is empty or full. The Inventory is updated whenever Wholesale Product comes in or The store serves Product to a customer.
Store Attributes:

There are 5 attributes connected to the Store entity with the Store ID attribute being the primary key.

**Location**: String

Location will hold the address of the store.

Type = String  range = 1024 characters  Default = address  Null not allowed
Unique = yes  single value  simple  key candidate = yes

**Store ID**: Integer

The sole purpose of this attribute is to be the primary key for the Store entity.

Type = Int  Range = (1,999)  Default = 1  Null not allowed
Unique = yes  single value  simple  key candidate = yes

**Operating Cost**: Decimal

This attribute represents the expenses for the store. Eventually, we would be able to generate this as a derived attribute but we won’t have time to keep track of power, rent etc. so we will have to create a number out of thin air.

Type = Decimal  range = (0,100000)  Default = 0?  Null not allowed
Unique = yes  single value  simple  Key candidate = no

**Sales**: Decimal

This number represents the total sales from the sale of Product from the employees to the customers and is derived from the Employees serving the Customers.

Type = Decimal  range = (0,1000000)  Default = 0  Null not allowed
Unique = no  single value  simple  key candidate = no
**Gross Profit**: Decimal

This attribute represents the gross profit of the company and can be derived from the Sales attribute and cost of the wholesale Product attribute.

Type = Decimal range = (0,1000000) Default = 0 Null not allowed

Unique= no single value simple Key candidate = no

**Net Profit**: Decimal

This attribute represents the net profit of the company and can be derived from the Operating Cost and Gross Profit attributes.

Type = Decimal range = (0,1000000) Default = 0 Null not allowed

Unique= no single value simple Key candidate = no

**Product:**

Product is a strong entity:

This entity holds the information about the products the store sells. Product needs to be updated as products are changed. i.e. seasonal or new flavors.

**Relationships with Other Entities:**

The inventory contains products.

**Product Attributes:**

Product has 4 attributes:
**Product ID**: Int

This is the unique attribute that each product gets, letting us tell items apart.

- Type = Int
- Range = (0-999999)
- Default = ???
- No NULL
- Unique = True
- Single Simple Value
- Key Candidate = True

**Customer Cost**: Decimal

A non-unique attribute that holds the price the customer is paying per unit of the product. i.e. scoop or container.

- Type = Decimal
- Range = (0.00-200.00)
- Default = NULL
- Unique = False
- Single Simple Value
- Key Candidate = False

**Store Cost**: Decimal

A non-unique attribute that holds the price the store is paying for the product.

- Type = Decimal
- Range = (0.00-500.00)
- Default = 0.00
- Unique = False
- Single Simple Value
- Key Candidate = False

**Product Name**: String

An attribute that holds the name of each product, as humans will see them. This includes both ice cream and toppings.

- Type = String
- Range = 256 characters
- Default = NULL
- Unique = False
- Single Simple Value
- Key Candidate = False

**Employee**: Employee is a strong entity:

The purpose of this entity is to hold information about the employees working at the store.
Relationships with Other Entities:

An Employee Works At the Store. This needs to be updated whenever someone is hired or leaves.

Employee Attributes:

There are 5 attributes in Employee, with Employee ID being the unique attribute.

Wage: Decimal

This attribute represents the amount of money the employee will be receiving as pay per hour.

<table>
<thead>
<tr>
<th>Type</th>
<th>Range</th>
<th>Default</th>
<th>Nullable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decimal</td>
<td>(12.00 - 25.00)</td>
<td>12.00</td>
<td>NULL</td>
</tr>
</tbody>
</table>

Unique = no Single Simple Value key candidate = no

Employee ID: Integer

This attribute represents the Identification Number assigned to an Employee working at the Store.

<table>
<thead>
<tr>
<th>Type</th>
<th>Range</th>
<th>Default</th>
<th>Nullable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer</td>
<td>(1 - 999999)</td>
<td>1</td>
<td>NULL</td>
</tr>
</tbody>
</table>

Unique = YES Single Simple Value key candidate = YES

Name: String

This attribute consists of First and Last parts and represents the Employee’s name.

<table>
<thead>
<tr>
<th>Type</th>
<th>Range</th>
<th>Default</th>
<th>Nullable</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>(2-256)</td>
<td>“ “</td>
<td>NULL</td>
</tr>
</tbody>
</table>

Unique = no Composite Value key candidate = no
First:

This attribute represents just the First Name of the Employee.

Type: String  Range(1-128)  Default = “ “  NO NULL
Unique = no  Single Simple Value  key candidate = no

Last:

This attribute represents just the Last Name of the Employee.

Type: String  Range(1-128)  Default = “ “  NO NULL
Unique = no  Single Simple Value  key candidate = no

Hours Worked:

This attribute represents the amount of hours the employee worked. We have created a cap of 39.5 to prevent employees from getting full-time benefits.

Type = Decimal  Range (0 - 39.5)  Default = 0  NO NULL
Unique = no  Single Simple Value  key candidate = no

Pay:

This attribute represents the amount of money the employee will receive, found by the multiplication of Hours Worked and Wage.

Type = Decimal  Range (0 - 987.50)  Default = 0  NO NULL
Unique = no  Derived Value  key candidate = no

Customer:

Customer is a strong entity:

This entity is used to keep track of customers logged into the system for benefits and to track repeated purchases.
Relationships with Other Entities:

Customers are served by the Store:

In order to become a customer, something must be served and bought by them.

Customer Attributes:

There are 5 attributes in Customer, with Email Address being the unique attribute.

**Email**: String

This holds each customer's email. (As long as they’re a member)

Type = String  Range = (1, 256)  Default = “guest”  Null Not Allowed
Unique = yes  Simple  Key candidate = Yes

**Customer ID**: Integer

This attribute represents the Identification Number assigned to a Customer buying ice cream from the Store.

Type = Integer  Range(1 - 999999)  Default = 1  NO NULL
Unique = YES  Single Simple Value  key candidate = YES

**Name (First)**:

String for the first name of the customer on file.

Type = String  Range (1, 24)  Default = ‘A’ Null Not Allowed
Unique = no  Single Simple Value  Key Candidate = No

**Name (Last)**: String

String for last name of the customer on file.
**Visits: Integer**
The number of times a customer has visited.

- **Type = Integer**
- **Range = (1, 9999)**
- **Default = 1**
- **Null not allowed**
- **Unique = no**
- **Single Simple Value**
- **Key candidate = no**

---

**Inventory:**

Inventory is a weak entity:

This entity represents what the store has in stock to sell. Its primary key would be the store key and the product ID.

**Relationships with Other Entities:**

- **The store Carries one inventory.**
- **The inventory Contains many products.**
- **The inventory is Served to the customers.**

---

**1.2.2 - Relationship Type Description**

**Works At:**

This Relationship connects the Employee to the Store. This represents the amount of time an employee spends at a store and their cost to the business.

The entities involved with this relation are Employee and Store.

Cardinality = Employee (1,M) works at (1,N) Stores
This relationship relates the Store to the Customer who is served and has attributes that store money collected and product sold.

The entities involved in the relationship are the Store and the Customer.

Cardinality = Store (1,1) Serves (1,M) Customer.

Attributes of Serves:

**Payment Received:**
- Type = Decimal  range = (0.00,999.99)  Default = 0?  Null not allowed
- Unique = no  single value  simple  key candidate = no

**Transaction Date:**
- Type = datetime  range = (022621, 123199)  Default = 0000 Null not allowed
- Unique = no  single value simple  key candidate = no

**Transaction ID:**
- Type = Int  Range = (0000000001,9999999999)  Default = 0000000001
- Null not allowed
- Unique = yes  Single Value  Composite  key candidate = yes
- This ID will count up for every transaction across all stores.

**Product ID:**
- Type = Int  Range = (0001, 9999)  Default = 0001 No NULL
- Unique = no  Single Value  Key Candidate = No

**Container ID:** Int
- This is the attribute that describes each individual ice cream serving that makes up the complete order.
- Type = Int  Range = (0-???)  Default = 1  No NULL
- Unique = no  Single Simple Value  Key Candidate = True

Contains:
The purpose of this relationship is to describe what is in the inventory

Cardinality = Inventory(1,M) Contains (1,M) Product

Attributes of Contains:

**Quantity**: Int

A non-unique attribute that holds the number of containers left in inventory. One container holds around 40 scoops of ice cream.

Type = Int    Range = (0-999)    Default = 0
Unique = False    Single Simple Value    Key Candidate = False

**Expired**: Bool

A derived variable that is false until the expiration date is less than the current date, meaning that the product is expired and should be disposed of.

Type = Bool    Range = (0-1)    Default = False
Unique = False    Single Simple Value    Key Candidate = False

**Expiration**: String

A non-unique attribute that holds the expiration date of each product, as ice cream is perishable. An expiration of null means that the product does not expire, such as with the ice cream containers, as these are usually paper and don’t need to be disposed of.

Type = String    Range = “1/1/2020” - “12/31/9999”    Default = NULL
Unique = False    Single Simple Value    Key Candidate = False

**Carries**:

The purpose of this relationship is to link an inventory to a store.

Cardinality = Store (1,1) Carries (1,1) Inventory

1.2.3 - Related Entity Types

**Works at**: Total participation between employee and store due to works at relationship
Possible Overlap: Employee discount (?) where store serves employee as customer

**Serves:** Store allows for partial participation of Serves relationship while Customer requires total participation. A store can operate without customers for some time, while a customer cannot be a customer without being served.

**Works At:** Total participation from Store and Employee.

**Contains:** Partial participation as a product can exist without being in a store’s inventory.

**Carries:** Total participation as a store must carry an inventory and an inventory is carried by a store.

1.2.4 - ER Diagram
Phase 2 - Conversion from Conceptual to Relational

In this phase of our presentation we will be concentrating on the ER and Relational models. First there is a general description of both types of models with their benefits and drawbacks. After describing the two models in general terms, we will discuss how we implemented the models into our specific project, so we can continue to produce our database design for the, “Yummers,” company.

2.1 - The ER and Relational Models

2.1.1 - Descriptions of ER and Relational Models

In order to develop a successful database, you need to collect and analyze the requirements. The requirements include the user-defined transactions such as retrievals, updates, and deletions. Once this is accomplished you need to develop a conceptual schema using a high level conceptual model. This step is called conceptual design. One of the main models developed for this purpose is the ER Model.

The Entity-Relationship Model became popular in 1976 after Peter Chen wrote a paper describing the model. The ER model defines the conceptual view of a database. It works around real-world entities and the associations among them. In ER diagrams the emphasis is on representing the schema’s rather than the instances. This is because the schema is more permanent compared to instances which change continuously. The schema is much easier to display in a small area, so people can view how all the entities, attributes, and relationships are connected to each other. The following are aspects that are included in an ER diagram.

A strong entity is an object or idea that can be easily identifiable. Some examples of strong entities that may show up in a school database could include students, teachers, classes, and courses. In the ER diagram a strong entity is depicted by a rectangle. All strong entities have some attributes or properties that give them their identity. An entity set is a collection of similar types of entities. An entity set may contain entities with attributes sharing similar values. For example, a Students set may contain all the students of a school. Entity sets may or may not be disjoint.

A weak entity is an entity set that does not have sufficient attributes for unique identification. It also does not have a primary key. Instead it has a discriminator that distinguishes itself from other weak entities in the set. The discriminator is represented by underlining with a dashed line. A weak entity is symbolized by a double rectangle and its relationship to the strong entity is designated by a double diamond.

Attributes are the properties that the entities possess and are represented by ellipses in the ER diagram. All attributes have values that must fall in a specified domain. For example, a student entity may have Name and GPA for 2 of their attribute values. The values must fall within a specified domain such as alphabetic for name and numeric for GPA.
Attributes can be simple, which means the attributes cannot be divided any further such as a telephone number. Composite attributes are attributes that are made up of more than one simple attribute. A common example is, “name,” being made up of First_name, middle_name, and last_name. A derived attribute is an attribute that is not physically placed in the database but calculated from another attribute in the database. Age is a common derived attribute and can be derived from the birth_date attribute. A dashed ellipse signifies a derived attribute. The multi-value attribute can have multiple values such as a person’s email address. The multi-value attribute is designated by a double ellipse.

A Key attribute is an attribute that can distinguish each entity from all of the other entities in the entity set and is designated by being underlined.

A relationship is designated by a diamond in the ER diagram model. It relates two or more entities to each other. A group of relationships of similar type are called relationship sets. Like an entity, a relationship can also have attributes. Attributes of relationships are called descriptive attributes.

The number of entities that are connected to each other through a relationship is referred to as the degree of the relationship. A second degree relationship relates 2 entities and is also called binary. A third degree relationship is called ternary and relates 3 entities together. An important aspect of the relationship of entities is their cardinality to each other.

The cardinality defines the number of entities in one entity set, which can be associated with the number of entities of the other set through the relationship set. There are three possible cardinalities between entities. One to one, one to many, and many to many. Another way to represent this concept is by the min/max notation. This is done by typing the minimum and maximum number of entities from entity set A that can be related to the minimum and maximum number of entities in entity set B.

The relational model for database management is an approach to managing data building on first-order predicate logic, which was first described in 1969 by English computer scientist Edgar F. Codd.

A predicate is the formalization of the mathematical concept of statement. A statement is commonly understood as an assertion that may be true or false, depending on the values of the variables that are contained in it. A predicate is a well-formed formula that can be evaluated to be either true or false, depending on the results of the conditions that are defined. Therefore, It can be considered to be a Boolean-valued function. A predicate can be made of several atomic formulas that are connected by logical connectives such as negation, conjunction, disjunction, existential quantification, and universal quantification. A predicate whose quantifiers all apply to individual elements, and not to sets or predicates, is called a first-order predicate. In this logic, data is represented in terms of tuples, grouped into relations. A database organized in terms of the relational model is a relational database.

The purpose of the relational model is to provide a method for specifying data and queries. It allows users to directly state what information is contained in the database and what information they want from it. The database management system software is able to take care
of describing data structures for storing the data and retrieval procedures for answering queries. Most relational databases use the SQL data definition and query language.

The relational model's central idea is to describe a database as a collection of predicates over a finite set of predicate variables. The model describes constraints on the possible values and combinations of values.

The content of the database at any given time is a set of relations, one per predicate variable, such that all predicates are satisfied. A request for information from the database (a database query) is also a predicate.

The fundamental assumption of the relational model is that all data can be represented by mathematical relationships. The user can place a query in the form of a predicate that can evaluate each requested relationship as either true or false. Every relationship that evaluates to true is then retrieved from the database. This process can be accomplished by using either relational algebra or relational calculus.

2.1.2 - Model Comparisons

The basic difference between the two models is that ER Models deal with entities and their relations, while relational Models deal with Tables and relations between the data of those tables. Both models are integral in developing an accessible database that provides a user with relevant, accurate and understandable data when given a query. An ER model is a conceptual model that provides a diagram of various entities with their attributes and the relationships among the entities. This graphical depiction helps to clearly understand the data structure and helps to minimize confusion and redundancy of the data. Although having a graphical depiction of relationships is helpful in understanding how everything relates as a whole, there is no practical use unless you can implement a logical data structure. The relational model is a logical model which allows us to use computers to help manipulate data through queries.

Choosing between an ER Model and a Relational Model is not an either/or proposition. Rather, an ER Model is a great conceptual model that can provide an excellent basis to be transformed into a relational model. The relational model is a logical model that provides the opportunity to have a database that can use queries to extract information in the form of tuples in order for a company to improve their decision making.

2.2 - Conceptual to Logical Conversion Process

2.2.1 - Converting Entity Types to Relations

When converting an ER Diagram to a relational model, several steps should be followed to ensure that the process is done successfully. First, create a relation for each strong entity, with one unique attribute as the primary key of the relation. All other non multivalued and derived attributes are included in the relation. Weak entities will follow the same process, with
2.2.2 - Converting Relationship Types to Relations

When converting relationships to relations, the strategy changes based on the type of relationship. The three types of relationships are 1:1, 1:N, and M:N.

A 1:1 relationship is a relationship that has only one entity on both sides of the relationship. An example for 1:1 relationship is a traditional marriage. In a monogamous marriage, each person can only be married to one other person. A 1:1 can be implemented through foreign keys, merged relationships, and cross referencing with a third relation. A foreign key is the most common and useful of options, where the primary key of one relation is stored as a foreign key in another. This solution is simple and easier to implement but if done incorrectly can lead to large amounts of wasted space. If applicable, both relations could also be merged into one relation that has the attributes of both. This option works but does not allow the relations to be separate from each other. Another option is to create a third relation that acts as a middle man between the two relations, also called a lookup table. The primary key of this relation is the primary keys of both other relations. This option allows both relations to be separate while still referencing each other, but has the downside of creating another table in the database.

A 1:N relationship is a relationship that has many entities related to one other entity. An example for this is an elementary school student and their teacher. A student only has one teacher but a teacher has many students. A 1:N relationship can be implemented through a foreign key and by cross referencing a third relation. These options work the same as they do for 1:1 relationships, and foreign keys are the preferred option because it reduces the number of tables needed.

A M:N relationship is a relationship that has many entities related to many other entities. An example for this is a high school student and their classes. A student can be in many classes and a class can have many students. Implementing M:N relationships requires using a lookup table that cross references the relationships.

Another aspect that needs to be translated to relations is multivalued attributes. A multivalued attribute is an attribute that can contain many values, such as a person that has multiple emails. The process to translate these is to create another relationship that holds both the multivalued attributes and the primary key of the entity that the multivalued attribute belongs to.
2.2.3 - Converting Extended Types to Relations

There are four methods to convert subclasses, or specialized types, and their superclass, or generalized type, to relation schemas.

The first method is for multiple relations. It relates a generalized superclass to multiple subclasses. You need to create a relation L for the superclass C with attributes ATTR(L) = {k, a1, ...} and Primary Key(L) = k. You need to create a relation Li for every subclass Si with attributes ATTR(Li) = {k} union {attributes of Si} and primary key(Li) = k. This method is able to work with any type of specialization either partial or total as well as either disjoint or overlapping.

The second method works for multiple relations but only for subclasses. For this option you create a relation Li for each subclass Si with attributes ATTR(Li) = {attributes of Si} union {k, a1, a2, ..., an} and primary key(Li) = k. This choice works well if the specialization is total, which means that every entity in the superclass must belong to at least one of the subclasses in the relation. Another requirement for this option to work is that the specialization must be disjoint, not overlapping. An overlap could cause an entity to be duplicated in several places.

The third method works when there is a single relation with one type of attribute. For this option you create a single relation L with attribute ATTR(L) = {k, a1, a2, ..., an} union {attributes of Si} … union {attributes of SM} union {T} and primary key(Li) = k. The “T” represents a discriminating attribute that identifies the subclass that each tuple belongs to. This is another option that only works for disjoint subclass specializations. Another potential drawback is that many null values may be produced with this method.

The final method is for single relations with multiple type attributes. You start by creating a single relation schema L with attributes ATTRS(L) = {k, a1, ..., an} union {attributes of Si} union {attributes of SM} union {T1, T2, ..., Tm} and primary key(Li) = k. Each Ti is a Boolean that indicates whether or not a tuple belongs to the subclass. This method can be used for specializations with subclasses that overlap.

The final step in the mapping to relations process involves unions, also called categories. The starting point to accomplish this task is to make sure there is a common key to identify the group. In some cases, all of the superclasses may have a common key, which can then be used as the key to identify the group. However, most of the time a new key called a surrogate key is needed to identify the category of entities. This key attribute, whether occurring naturally or newly created, must be included as the foreign key in each relation corresponding to the superclass of the category. This maintains the integrity of the system.
2.2.4 - Database Constraints

The importance of database constraints ties from the need for a database to be reliable and fast. Constraints are applied as a part of the conversion to SQL to delineate which kinds of data can be placed into different segments of the database. If there is any discrepancy when inputting data into a portion of the database, the action is terminated and skipped.

There several different types of constraints present in a database system. Some of these constraints are listed below, along with their descriptions.

Entity integrity constraint states that no primary key constraint can be null. This is important because if primary keys were allowed to be null then there would be a very good chance that there would be multiple entities with the same primary key value, "null."

Primary key and unique key constraints guarantee unique data they are often defined on an identity column. When you identify a primary key constraint in a database, the database creates a unique index which provides fast access to the data.

Referential (foreign key constraints) are specified between two relations and are used to maintain the consistency among tuples in the two relations. The relational constraint states that a tuple in one relation that refers to another relation must refer to an existing tuple in that relation.

Semantic constraints are usually implemented through applications and are related to business rules. Some examples of this constraint include $\text{hours\_worked} < 60$ or $\text{Salary} < 100000$.

2.3 - Results of ER to Relational Conversion

2.3.1 - Relation Schema

Store: (Store ID, Operating Costs, Net Profit, Sales, Location)

Store ID
Domain: Positive Integer (1, 999)

Operating Costs
Domain: Decimal (0, 100000)

Net Profit
Domain: Decimal (-100000.00, 100000.00)

Sales
Domain: Decimal (0.00, 100000.0)

Location
Domain: String (0,1024)
Constraints:
Primary Key: Store ID

Inventory: (Store ID)
Store ID
Domain: Positive Integer (1,999)
Constraints:
Foreign Key: Store ID

Contains: (Store ID, Product ID, Expired, Expiration, Quantity)
Store ID
Domain: Positive Integer (1,999)
Product ID
Domain: Positive Integer (0,999999)
Expired
Domain: Boolean (True/False)
Expiration
Domain: Datetime
Quantity
Domain: Positive Integer (0,999)
Constraints:
Foreign Keys: Store ID, Product ID

Serves: (Transaction ID, Transaction Date, Container ID, Product ID, Customer ID, Store ID)
Transaction ID
Domain: Positive Integer, (1,999999)
Transaction Date
Domain: Datetime
Container ID
Domain: Positive Integer, (1,2)
Product ID
Domain: Positive Integer, (1,999999)
Customer ID
Domain: Positive Integer, (1,999999)
Store ID
Domain: Positive Integer, (1,999999)
Constraints:

Primary Key: Transaction ID
Foreign Key: Product ID, Store ID

Product: (Product ID, Product Name, Customer Cost, Store Cost)
  Product ID
    Domain: Positive Integer, (0,999999)
  Product Name
    Domain: String (0,256)
  Customer Cost
    Domain: Decimal (2.50,3.50)
  Store Cost
    Domain: Decimal (10.00,40.00)

Constraints:
  Primary Key: Product ID

Employee: (Employee ID, Store ID, Wage, Hours Worked, Name, First, Last)
  Employee ID:
    Domain: Positive Integer (1,999999)
  Store ID:
    Domain: Positive Integer (1, 999)
  Wage:
    Domain: Decimal (12.00,15.00)
  Hours Worked:
    Domain: Decimal (0,39.50)
  Name:
    Domain: String (2,256)
  First:
    Domain: String (1,128)
  Last:
    Domain: String (1,128)

Constraints:
  Primary Key: Employee ID
  Foreign Key: Store ID

Customer: (Customer ID, Email, Visits, Name, First, Last)
  Customer ID:
    Domain:
  Email:
    Domain: String (2,256)
  Visits:
    Domain: Integer(1,9999)
2.3.2 - Sample Data

<table>
<thead>
<tr>
<th>Store ID</th>
<th>Operating Cost</th>
<th>Net Profit</th>
<th>Sales</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>21146.41</td>
<td>62229.49</td>
<td>83375.90</td>
<td>2670 Ryan Circle</td>
</tr>
<tr>
<td>2</td>
<td>6203.74</td>
<td>18710.26</td>
<td>24914</td>
<td>56 Melvin Center</td>
</tr>
<tr>
<td>3</td>
<td>10728.24</td>
<td>36679.46</td>
<td>47407.70</td>
<td>3513 Kingsford Hill</td>
</tr>
<tr>
<td>4</td>
<td>6597.86</td>
<td>12384.04</td>
<td>18981.90</td>
<td>9737 Summit Center</td>
</tr>
<tr>
<td>5</td>
<td>23685.41</td>
<td>67995.39</td>
<td>91680.80</td>
<td>41 Rieder Place</td>
</tr>
<tr>
<td>6</td>
<td>12118.16</td>
<td>-10037.90</td>
<td>2080.22</td>
<td>10 Maple Crossing</td>
</tr>
<tr>
<td>7</td>
<td>18485.91</td>
<td>36130.29</td>
<td>54616.20</td>
<td>367 Lighthouse Bay Street</td>
</tr>
<tr>
<td>8</td>
<td>19750.59</td>
<td>67820.51</td>
<td>87571.10</td>
<td>421 Eagan Hill</td>
</tr>
<tr>
<td>9</td>
<td>19896.23</td>
<td>26273.67</td>
<td>46169.90</td>
<td>09245 Redwing Center</td>
</tr>
<tr>
<td>10</td>
<td>21285.02</td>
<td>39481.58</td>
<td>60766.60</td>
<td>8 Cordelia Road</td>
</tr>
</tbody>
</table>

Inventory:

<table>
<thead>
<tr>
<th>Store ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>Store ID</td>
</tr>
<tr>
<td>---------</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>Transaction ID</td>
</tr>
<tr>
<td>----------------</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>9</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>9</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>9</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>8</td>
</tr>
</tbody>
</table>

Serves:
<table>
<thead>
<tr>
<th></th>
<th>Date</th>
<th>Value1</th>
<th>Value2</th>
<th>Value3</th>
<th>Value4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3/5/2021</td>
<td>1</td>
<td>10</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4/21/2020</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>12/19/2020</td>
<td>2</td>
<td>2</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>12/28/2020</td>
<td>1</td>
<td>7</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>12/29/2020</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>5/30/2020</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>10/12/2020</td>
<td>1</td>
<td>6</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>5/25/2020</td>
<td>2</td>
<td>10</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>1/6/2021</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>6/14/2020</td>
<td>1</td>
<td>4</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>10/28/2020</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>4/30/2020</td>
<td>1</td>
<td>9</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>10/21/2020</td>
<td>2</td>
<td>3</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>11/5/2020</td>
<td>2</td>
<td>5</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>1/26/2021</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>10/21/2021</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>2/9/2021</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>3/13/2020</td>
<td>2</td>
<td>8</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>7/13/2020</td>
<td>1</td>
<td>6</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>1/29/2021</td>
<td>2</td>
<td>3</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>9/4/2020</td>
<td>1</td>
<td>8</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>6/12/2020</td>
<td>1</td>
<td>7</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>4/2/2020</td>
<td>1</td>
<td>8</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>11/18/2020</td>
<td>2</td>
<td>6</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>5/4/2020</td>
<td>1</td>
<td>6</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>8/3/2020</td>
<td>2</td>
<td>1</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>11/11/2020</td>
<td>1</td>
<td>2</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Date</td>
<td>Quantity</td>
<td>Store Cost</td>
<td>Customer Cost</td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>-----------</td>
<td>----------</td>
<td>------------</td>
<td>---------------</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>4/6/2020</td>
<td>1</td>
<td>10</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>3/7/2021</td>
<td>2</td>
<td>9</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>1/28/2021</td>
<td>2</td>
<td>8</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>10/3/2020</td>
<td>1</td>
<td>7</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>11/5/2020</td>
<td>1</td>
<td>6</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>6/24/2020</td>
<td>1</td>
<td>7</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>5/3/2020</td>
<td>2</td>
<td>10</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>4/24/2020</td>
<td>1</td>
<td>4</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>8/20/2020</td>
<td>2</td>
<td>5</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>9/25/2020</td>
<td>1</td>
<td>4</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>4/15/2020</td>
<td>2</td>
<td>5</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>39</td>
<td>2/7/2021</td>
<td>2</td>
<td>10</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>7/21/2020</td>
<td>1</td>
<td>2</td>
<td>9</td>
<td></td>
</tr>
</tbody>
</table>

**Product:**

<table>
<thead>
<tr>
<th>Product</th>
<th>Name</th>
<th>Store Cost</th>
<th>Customer Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>Strawberry</td>
<td>25</td>
<td>2.50</td>
</tr>
<tr>
<td>10</td>
<td>Mint Chocolate Chip</td>
<td>25</td>
<td>2.50</td>
</tr>
<tr>
<td>11</td>
<td>Salted Caramel</td>
<td>30</td>
<td>2.50</td>
</tr>
<tr>
<td>12</td>
<td>Rocky Road</td>
<td>35</td>
<td>3.50</td>
</tr>
<tr>
<td>13</td>
<td>Coffee</td>
<td>30</td>
<td>3.50</td>
</tr>
<tr>
<td>14</td>
<td>Cookie Dough</td>
<td>35</td>
<td>2.50</td>
</tr>
<tr>
<td>15</td>
<td>Raspberry</td>
<td>30</td>
<td>3.50</td>
</tr>
<tr>
<td>16</td>
<td>Birthday Cake</td>
<td>35</td>
<td>3.50</td>
</tr>
<tr>
<td>17</td>
<td>Vanilla</td>
<td>20</td>
<td>2.50</td>
</tr>
<tr>
<td>18</td>
<td>Chocolate</td>
<td>20</td>
<td>2.50</td>
</tr>
<tr>
<td>3</td>
<td>Sprinkles</td>
<td>10</td>
<td>1.00</td>
</tr>
<tr>
<td></td>
<td>Item</td>
<td>Quantity</td>
<td>Price</td>
</tr>
<tr>
<td>---</td>
<td>--------------</td>
<td>----------</td>
<td>-------</td>
</tr>
<tr>
<td>4</td>
<td>Caramel</td>
<td>15</td>
<td>1.00</td>
</tr>
<tr>
<td>5</td>
<td>Fudge</td>
<td>30</td>
<td>1.00</td>
</tr>
<tr>
<td>6</td>
<td>Strawberries</td>
<td>20</td>
<td>1.00</td>
</tr>
<tr>
<td>7</td>
<td>Marshmallow</td>
<td>15</td>
<td>1.00</td>
</tr>
<tr>
<td>8</td>
<td>Gummy Worms</td>
<td>20</td>
<td>1.00</td>
</tr>
<tr>
<td>1</td>
<td>Cone</td>
<td>40</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>Cup</td>
<td>20</td>
<td>0</td>
</tr>
</tbody>
</table>

**Employee:**

<table>
<thead>
<tr>
<th>Employee ID</th>
<th>Store ID</th>
<th>Wage</th>
<th>Hour</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>14.38</td>
<td>26.0</td>
<td>Edith Reville</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>12.20</td>
<td>19.6</td>
<td>Borg Fahy</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>14.27</td>
<td>27.9</td>
<td>Jenda Hinners</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>14.32</td>
<td>16.0</td>
<td>Kile Zoane</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>12.39</td>
<td>23.2</td>
<td>Geoff Laurenceau</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>13.62</td>
<td>25.8</td>
<td>Melisenda MacRorie</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>13.52</td>
<td>35.2</td>
<td>Mehetabel Carling</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>14.57</td>
<td>14.6</td>
<td>Thane Yard</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>12.47</td>
<td>15.8</td>
<td>Wald Peinke</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>14.75</td>
<td>13.2</td>
<td>Andriana Culbard</td>
</tr>
</tbody>
</table>

**Customer:**

<table>
<thead>
<tr>
<th>Customer ID</th>
<th>Email</th>
<th>Visits</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><a href="mailto:jkneeland0@jalbum.net">jkneeland0@jalbum.net</a></td>
<td>45</td>
<td>Jonie Kneeland</td>
</tr>
<tr>
<td>2</td>
<td><a href="mailto:jselcraig1@fsa.gov">jselcraig1@fsa.gov</a></td>
<td>27</td>
<td>Joyous Selcraig</td>
</tr>
<tr>
<td></td>
<td>Email</td>
<td></td>
<td>Name</td>
</tr>
<tr>
<td>---</td>
<td>---------------------------</td>
<td>---</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>3</td>
<td><a href="mailto:mpattington2@google.ca">mpattington2@google.ca</a></td>
<td>118</td>
<td>Marcella Pattington</td>
</tr>
<tr>
<td>4</td>
<td><a href="mailto:lpowers3@opensource.org">lpowers3@opensource.org</a></td>
<td>11</td>
<td>Leta Powers</td>
</tr>
<tr>
<td>5</td>
<td><a href="mailto:clindenbluth4@printfriendly.com">clindenbluth4@printfriendly.com</a></td>
<td>84</td>
<td>Claudina Lindenbluth</td>
</tr>
<tr>
<td>6</td>
<td><a href="mailto:dlilliman5@about.com">dlilliman5@about.com</a></td>
<td>16</td>
<td>Danyette Lilliman</td>
</tr>
<tr>
<td>7</td>
<td><a href="mailto:rply6@constantconstact.com">rply6@constantconstact.com</a></td>
<td>6</td>
<td>Raff Ply</td>
</tr>
<tr>
<td>8</td>
<td><a href="mailto:wcrissil7@bizjournals.com">wcrissil7@bizjournals.com</a></td>
<td>34</td>
<td>Wyatt Crissil</td>
</tr>
<tr>
<td>9</td>
<td><a href="mailto:epindell8@boston.com">epindell8@boston.com</a></td>
<td>24</td>
<td>Erek Pindell</td>
</tr>
<tr>
<td>10</td>
<td><a href="mailto:ncainelli9@marketwatch.com">ncainelli9@marketwatch.com</a></td>
<td>20</td>
<td>Nevs Cianelli</td>
</tr>
</tbody>
</table>

### 2.4 - Sample Queries

#### 2.4.1 - Design of Queries

1. Select Transactions that include strawberry ice cream
2. Select Customers who have ordered every flavor of ice cream.
3. Select Most frequently ordered ice cream
4. How many hours employee with ID X has worked this week
5. Select Customer with the most visits
6. Select Customers who have ordered from every store
7. Select Employee(s) with the highest hourly pay.
8. Select Store with lowest net profit
9. Find the most popular flavor in a particular store
10. Find Stores with Strawberry Ice Cream in stock.
2.4.2 - Relational Algebra Expressions

1. Select Transactions that include strawberry ice cream
   
   \[ R \leftarrow \sigma_{\text{pname} = "Strawberry"}(\text{Product}) \]
   
   \[ R2 \leftarrow \pi_{\text{productID}(R) = \text{spid}(\text{Serves})} \]
   
   \[ R3 \leftarrow R2 \bowtie \text{Serves} \]
   
   \[ \text{Final} \leftarrow \pi_{\text{tID}}(R3) \]

2. Select Customers who have ordered every flavor of ice cream.
   
   \[ R \leftarrow \sigma((\text{productID} > 100000) \land (\text{productID} < 999999))(\text{Product}) \]
   
   \[ R2 \leftarrow \pi_{\text{productID}}(R) \]
   
   \[ R3 \leftarrow \text{serves} / R2 \]
   
   \[ R4 \leftarrow \pi_{\text{customerID}}(R3) \]

3. Select Most frequently ordered ice cream
   
   \[ R(\text{spid}, \text{numTransactions}) \leftarrow \text{spid}(G) \bowtie \text{count(plID)}(\text{Serves}) \]
   
   \[ R1 \leftarrow \sigma((\text{spid} > 100000) \land (\text{spid} < 999999))(R0) \]
   
   \[ R2(\text{sID}, \text{spid}, \text{sellcount}) \leftarrow \text{sID}(G) \bowtie \text{spid}, \text{max(numTransactions}(R1) \]
   
   \[ R3 \leftarrow G[\text{spid}], \text{max(numTransactions}(R2) \]
   
   \[ R4 \leftarrow R3 \bowtie \text{Product} \]
   
   \[ \text{final} \leftarrow \pi_{\text{pname}}(R4) \]

4. How many hours “X” employee has worked this week
   
   \[ R1 \leftarrow \sigma_{\text{ename} = "X"}(\text{Employee}) \]
   
   \[ R2 \leftarrow \pi_{\text{hoursWorked}}(R1) \]
5. Select Customer with the most visits

\[ R_1 \leftarrow (G) \max, \text{visits}^{\text{Customer}} \]

6. Select Customers who have ordered from every store

\[ R \leftarrow \sigma_{\text{storeID}}^{\text{Store}} \]
\[ R_2 \leftarrow \text{Serves}/R \]
\[ R_3 \leftarrow \pi_{\text{cID}}(R_2) \]
\[ R_4 \leftarrow R_3 \ast \text{Customer} \]
\[ \text{Final} \leftarrow \pi_{\text{cName}}(R_4) \]

7. Select Employee(s) with the highest hourly pay.

\[ R_1 \leftarrow \text{EmployeeID}, \max(\text{Wage})^{\text{Employee}} \]
\[ \text{Final} \leftarrow R_1 \ast \text{Employee} \]

8. Select Store with lowest net profit

\[ R \leftarrow G \text{ sID}, \min(G) \min(\text{netProfit})^{\text{Store}} \]
\[ R_2 \leftarrow G(\text{sID}) \min(\text{netProfit})^{(R)} \]
\[ R_3 \leftarrow \pi_{\text{sID}}(R_2) \]
\[ \text{Final} \leftarrow R_3 \ast \text{Store} \]

9. Find the most popular flavor in a particular store

\[ R_0(\text{spid, numTransactions}) \leftarrow \text{spid}(G) \text{count}(\text{tid})^{\text{Serves}} \]
\[ R_1 \leftarrow \sigma((\text{spid} > 100000)^{\land} (\text{spid} < 999999))(R_0) \]
\[ R_2(\text{sID, spid, sellcount}) \leftarrow \text{sID}(G) \text{spid}, \max(\text{numTransactions})(R_1) \]
R3<-σsID=1
R4<-R3*R2
R5<-R4*Product
Final<-πspid, name(R4)

10. Find Stores with Strawberry Ice Cream in stock.
   R1 <-sigma  pname = "strawberry" (Product)
   R2<-πstoreID(productID(R1)[contains]
   C<-πQuantity(R2)>0
   Final <- R2*C

2.4.3 - Relational Calculus Expressions

1. Select Transactions that include strawberry ice cream

   { t.Transaction_ID | Serves(t) ^ ∃ p( Product(p) ^ p.productName = "Strawberry Ice Cream" &
   t.productID = p.productID} 

2. Select Customers who have ordered every flavor of ice cream.

   Select customers who, for every flavor, there exists a transaction of that flavor for the customer.

   { c | Customer(c) ^ ∀ p( ¬Product(p) | p.productID > 1 ^ ∃ s( Serves(s) ^ s.customerID =
   c.customerID ) ) } 

3. Select Most frequently ordered ice cream

   Can’t be shown in calculus, requires algebraic count

4. How many hours X employee has worked this week

   { e.hoursWorked | Employee(e) ^ e.eID = X} 

5. Select Customer with the most visits

   { c | Customer(c) ^ ¬ ∃ x( Customer(x) ^ x.visits > c.visits ^ c.cID != x.cID ) }
6. Select Customers who have ordered from every store

Select customers who, for every store, there exists a transaction in that store for the customer.

\{ c | \text{Customer}(c) \land \forall s(\neg\text{Store}(s) \lor \exists t(\text{Serves}(t) \land t.\text{customer}_\text{ID} = c.\text{customer}_\text{ID} \land t.\text{store}_\text{ID} = s.\text{store}_\text{ID})) \} \}

7. Select Employee(s) with the highest hourly pay.

\{ e | \text{Employee}(e) \land \neg \exists x( \text{Employee}(x) \land x.\text{wage} > e.\text{wage} \land e.\text{eID} \neq x.\text{eID} ) \}

8. Select Store with lowest net profit

\{ s | \text{Store}(s) \land \neg \exists x( \text{Store}(x) \land (x.\text{netProfit} > s.\text{netProfit} \land s.\text{sID} \neq x.\text{sID}) ) \}

9. Find the most popular flavor in a particular store

Can’t be shown in calculus, requires algebraic count

10. Find Stores with Strawberry Ice Cream in stock.

\{ s | \text{Store}(s) \land \text{Product}(p) \land \text{Contains}(c) \land c.\text{storeID} = s.\text{storeID} \land p.\text{productName} = \text{“Strawberry Ice Cream”} \land p.\text{productId} = c.\text{productId} \land c.\text{quantity} > 0 \}

**Phase 3 - Physical Implementation**

3.1 - Relational Normalization

3.1.1 - Normalization process

1. What is normalization and why is it performed?

   Normalization is the process of organizing data in a database. This includes creating tables and establishing relationships between those tables according to rules designed to protect the data as well as eliminate redundancies and inconsistent dependencies.

   Informally, the goal of a relational database should include:
   
   1. Making sure that the semantics of the attributes in the schema are clear.
   2. Reducing the redundant information in the tuples.
   3. Reducing the null values in tuples.
   4. Making it impossible to produce unnecessary tuples.

   Formally, normalization is the process of analyzing the given relation schemas based on the functional dependencies and primary keys, in order to minimize redundancy and minimize insertion, deletion, and update anomalies.

2. What are the First, Second, Third, and Boyce-Codd Normal Forms?
First normal form states that the domain of an attribute must include only simple, indivisible values and that the value of any attribute in a tuple must be a single value from the domain of that attribute. In other words, it does not allow relations of relations or relations as attribute values within tuples.

Second normal form is based on full functional dependency. A relation schema passes the second normal form if every non prime attribute in the schema is fully functionally dependent on the primary key of the schema.

Third normal form is based on the concept of transitive dependency. A relation passes the third normalization form if no non-prime attribute of the relation is transitively dependent on the primary key. A functional dependency x -> y in a relation is transitive if there exists a set of attributes in z in the relation that is neither a candidate key or a subset of any key of the relation, where x -> z and z -> y hold. In more simplified terms, schemas should eliminate fields that do not depend on the key. Values in a record that are not part of that record's key do not belong in the table. In general, anytime the contents of a group of fields may apply to more than a single record in the table, consider placing those fields in a separate table.

Boyce-Codd normal form is a little more strict than 3nf. 3nf does allow for some functional dependencies to slip through in order to allow the database some flexibility providing easier implementation. The bcnf does not allow for any functional dependencies whatsoever. When decomposing the relations during the normalization process we must maintain the non additive join property and try to preserve the functional dependency preservation property.

3. What kind of anomalies can occur if a relation model is not normalized?
   There are 3 anomalies that can occur without proper normalization: update, deletion, and insertion anomalies.

4. What is the relationship between normalization and update anomalies?
   An update anomaly is a data inconsistency that results from data redundancy and a partial update. The purpose of normalization is to protect the data as well as eliminate redundancies and inconsistent dependencies. In other words, the purpose of normalization is to avoid update, insertion, and deletion anomalies.

3.1.2 - Application to Relational Model

Relation Schema Check for Normalization:
Store: (Store ID, Operating Costs, Net Profit, Sales, Location)
Store ID:
   Domain: Positive Integer (1, 999)
Operating Costs:
   Domain: Decimal (0, 100000)
Net Profit:
   Domain: Decimal (-100000.00, 100000.00)
Sales:
   Domain: Decimal (0.00, 100000.0)
Location:
   Domain: String (0,1024)
Constraints:
  Primary Key: Store ID

Normalization form analysis for Store:
Store passes 1nf because each attribute has a single value that is indivisible.
Store passes 2nf because every attribute depends on which store it belongs to.
Store passes 3nf because every attribute depends on the store for its value. For example, the location of the store depends on which store is the entity. You could not define the location based on the sales attribute. There appears to not be any exceptions to the 3nf, so Store appears to be in Boyce-Codd normalization form.

Inventory: (Store ID)
Store ID:
  Domain: Positive Integer (1,999)
Constraints:
  Foreign Key: Store ID

Normalization analysis:
Inventory is a weak entity and is also in Boyce-Codd Normalization Form because it is made of the Store primary key.

Contains: (Store ID, Product ID, Expired, Expiration, Quantity)
Store ID:
  Domain: Positive Integer (1,999)
Product ID:
  Domain: Positive Integer (0,999999)
Expiration:
  Domain: Datetime
Constraints:
  Foreign Keys: Store ID, Product ID

Normalization analysis:
Upon analyzing the contains relation we realized that we did not need the boolean expired attribute. We also found that the quantity attribute was superfluous and could lead to some ambiguity with our Serves relation. After removing these two attributes, we believe that it is in 3nf.
After implementing these changes our new contains relation is:
Contains: (Store ID, Product ID, Expiration)
Store ID:
  Domain: Positive Integer (1,999)
Product ID:
  Domain: Positive Integer (0,999999)
Expiration:
  Domain: Datetime
Constraints:
  Foreign Keys: Store ID, Product ID

Serves: (Transaction ID, Transaction Date, Container ID, Product ID, Customer ID, Store ID)
  Transaction ID:
    Domain: Positive Integer, (1,999999)
  Transaction Date:
    Domain: Datetime
  Container ID:
    Domain: Positive Integer, (1,2)
  Product ID:
    Domain: Positive Integer, (1,999999)
  Customer ID:
    Domain: Positive Integer, (1,999999)
  Store ID:
    Domain: Positive Integer, (1,999999)
Constraints:
  Primary Key: Transaction ID
  Foreign Key: Product ID, Store ID

Normalization analysis:
Serves's attributes have single values, all of the attributes are dependent on the transactionID, and the transactionID is the primary key and no other attributes can specifically define a transaction besides the transactionID. Therefore, Serves is in 3nf and possibly bcnf.

Product: (Product ID, Product Name, Customer Cost, Store Cost)
  Product ID:
    Domain: Positive Integer, (0,999999)
  Product Name:
    Domain: String (0,256)
  Customer Cost:
    Domain: Decimal (2.50,3.50)
  Store Cost:
    Domain: Decimal (10.00,40.00)
Constraints:
  Primary Key: Product ID
Normalization analysis:
Product passes 1nf and 2nf. There may be some debate in regards to 3nf because pname could be used as a primary key, but we want to keep pname to make the Product more accessible by humans. Thus, product is in 3nf.
Employee: (Employee ID, Wage, Hours Worked, Name, First, Last)
Employee ID:
    Domain: Positive Integer (1,999999)
Store ID:
    Domain: Positive (1,999)
Wage:
    Domain: Decimal (12.00,15.00)
Hours Worked:
    Domain: Decimal (0,39.50)
Name:
    Domain: String (2,256)
First:
    Domain: String (1,128)
Last:
    Domain: String (1,128)
Constraints:
    Primary Key: Employee ID
    Foreign Key: Store ID

Normalization analysis:
Employee is in 3nf with the same arguments as Product.

Customer: (Customer ID, Email, Visits, Name, First, Last)
Customer ID:
    Domain: int(1, 999999)
Email:
    Domain: String (2,256)
Visits:
    Domain: Integer(1,9999)
Name:
    Domain: String (2,256)
First:
    Domain: String (1,128)
Last:
    Domain: String (1,128)
Constraints:
    Primary Key: Customer ID, Email
Normalization analysis:
Customer appears to be in 3nf as well for reasons similar to Product and Employee.
Upon analyzing the normalization needs
Of our relation schema, we felt that we did a pretty good job of putting them in normalized form
This is probably because we started our project from scratch with implementing changes to our
ER Diagrams and then our relational models along the way. The one relation that we did have to make some changes to was the contains relation where we removed the expired and quantity attributes.

3.2 - Database Implementation

3.2.1 - Background Information

Relational database management software is used to keep track of a large amount of data stored in an ordered manner. Using this software makes it easier to run queries to find data that is needed. Client-server RDBMS allows for rapid data writing, while single-file systems like SQLite stop concurrent writing. Using a client-server RDBMS allows for better security as entry to the server can be limited by requiring authorization and limiting permissions of users. While client-server RDBMS allows for rapid data writing and better security, single-file RDBMS consists of a single file allowing for simple portability. This portability makes it easier to use the database for other platforms once it is developed, unlike the client-server RDBMS. For the Yummers brand’s database, we’ve decided to use MariaDB, as it is on the Delphi server.

3.2.2 - Schema and Hosting

The following section contains the commands used in the generation of our tables. The format of this section is:
Relation: (Relational Model for the Relations)
Command to create the Relation in MariaDB

The tables have also been populated with the data given above in Sections 2.3.2.

Store: (Store ID, Operating Costs, Net Profit, Sales, Location)
CREATE TABLE IF NOT EXISTS store (  
    storeID INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,  
    operatingCost DECIMAL(10,2) NOT NULL,  
    sales DECIMAL(10,2) DEFAULT NULL,  
    location VARCHAR(255) NOT NULL  
);  

Inventory: (Store ID)
CREATE TABLE IF NOT EXISTS inventory (  
    storeID INT UNSIGNED NOT NULL,  
    FOREIGN KEY (storeID) REFERENCES store(storeID)  
);  

Product: (Product ID, Product Name, Customer Cost, Store Cost)
CREATE TABLE IF NOT EXISTS product (  
    productId INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,  
    ...
Employee: (Employee ID, Wage, Hours Worked, Name, First, Last
CREATE TABLE IF NOT EXISTS employee ( employeeID INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY, wage DECIMAL(10,2) DEFAULT NULL, hoursWorked DECIMAL(10,2) DEFAULT NULL, eFirst VARCHAR(50) NOT NULL, eLast VARCHAR(50) NOT NULL
);

Customer: (Customer ID, Email, Visits, Name, First, Last)
CREATE TABLE IF NOT EXISTS customer ( customerID INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY, email VARCHAR(255) NOT NULL, visits INT NOT NULL DEFAULT 0, cFirst VARCHAR(50) NOT NULL, cLast VARCHAR(50) NOT NULL
);

Contains: (Store ID, Product ID, Expired, Expiration, Quantity)
CREATE TABLE IF NOT EXISTS contains ( storeID INT UNSIGNED NOT NULL, productID INT UNSIGNED NOT NULL, expiration DATETIME DEFAULT NULL, FOREIGN KEY (storeID) REFERENCES store(storeID), FOREIGN KEY (productID) REFERENCES product(productID), PRIMARY KEY (storeID, productID, expiration)
);

Serves: (Transaction ID, Transaction Date, Container ID, Product ID, Customer ID, Store ID)
CREATE TABLE IF NOT EXISTS serves ( transID INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY, transDate TIMESTAMP NOT NULL DEFAULT NOW(), containerID INT NOT NULL, productID INT UNSIGNED NOT NULL, customerID INT UNSIGNED NOT NULL, storeID INT UNSIGNED NOT NULL, FOREIGN KEY (productID) REFERENCES product(productID), FOREIGN KEY (customerID) REFERENCES customer(customerID), FOREIGN KEY (storeID) REFERENCES store(storeID)
3.3 - Query Implementation

This section contains the queries from Section 2.4 converted into MariaDB queries in order to poll our database.

1. Select Transactions that include strawberry ice cream
   SELECT serves.*, product.productName
   FROM serves INNER JOIN product ON (product.productID = serves.productID)
   WHERE product.productName = "Strawberry";

2. Select Customers who have ordered every flavor of ice cream.
   SELECT customerID, (SELECT COUNT(*) FROM product WHERE product.productID > 8) as total
   FROM serves
   GROUP BY customerID
   HAVING COUNT(productID > 8) >= total;

3. Select Most frequently ordered ice cream
   SELECT product.productID, product.productName, count(*)
   FROM product INNER JOIN serves ON (product.productID = serves.productID)
   WHERE product.productID > 8
   GROUP BY product.productID
   ORDER BY COUNT(*) desc
   LIMIT 1;

4. How many hours employee #5 has worked this week
   SELECT employee.eFirst, employee.eLast, employee.hoursWorked
   FROM employee
   WHERE employeeID = 5;

5. Select Customer with the most visits
   SELECT *
   FROM customer
   WHERE visits =
   (SELECT DISTINCT customer.visits
    FROM customer
    ORDER BY customer.visits DESC
    LIMIT 1
   );
6. Select Customers who have ordered from every store
SELECT customerID, COUNT(DISTINCT storeID)
FROM serves
GROUP BY customerID
HAVING COUNT(DISTINCT storeID) = (SELECT COUNT(*)
FROM store);

7. Select Employee(s) with the highest hourly pay.
SELECT *
FROM employee
WHERE wage =
(  
    SELECT DISTINCT employee.wage
    FROM employee
    ORDER BY employee.wage DESC
    LIMIT 1
);  

8. Select Store with lowest net profit
SELECT storeID, netProfit
FROM(
    SELECT store.storeID, store.sales, store.operatingCost, store.sales -
    store.operatingCost as netProfit
    FROM store)
ORDER BY netProfit
LIMIT 1;

9. Find the most popular flavor in Store X where X is the Store ID
SELECT product.*, COUNT(*)
FROM product INNER JOIN serves ON (product.productID = serves.productID)
INNER JOIN store ON (store.storeID = serves.storeID AND store.storeID = 7)
GROUP BY product.productID
ORDER BY COUNT(*) DESC
LIMIT 1;

10. Find Stores with ‘8’ in stock where ‘8’ is the Product ID.
SELECT store.storeID, contains.productID, product.productName
FROM store INNER JOIN contains ON (store.storeID = contains.storeID)
INNER JOIN product ON (contains.productID = product.productID)
WHERE product.productID = 8;
4.1 - Introduction

The programming logic in sql differs from most programming languages in that it uses a three value logic rather than the standard two valued Boolean logic. Standard logic systems use true or false values but due to the unknown value that can occur with "null" values, sql needs to use true, false, and unknown. Instead of using 2 by 2 matrices for truth tables, you need to use a 3 by 3 matrix. A brief synopsis of the AND operation is:

True AND True is True
False AND (True, False, or Unknown) is False
unknown and true is unknown.

For the OR operation:
True OR (True, False, Unknown) is True.
False OR False is False
Unknown OR (false, Unknown) is Unknown.

Finally for NOT,
NOT True is False
NOT False is True
NOT Unknown is Unknown

A create trigger statement is a statement you can write in order to tell the database to take specific action if certain events occur. An example of this is setting up a condition where an employee's salary should never be higher than their manager's salary. This event could occur if an employee was inserted into the database, the employee's salary changed, or the employee's manager was updated. If any of the events caused an employee's salary to be higher than their manager's salary then a trigger would be set off and a message could be sent to a manager.

A view is a table that has been derived from other tables in a database. It is helpful to create views of tables that are frequently referenced but don't exist as specific tables in the database, especially a table that may require two or more joins to create.

A Stored procedure is a prepared code in sql that you can save, so you can use it over and over again, kind of like a miniature program. You can also pass parameters to a stored procedure and the procedure will act according to the parameter that is passed to it. Stored procedures can insert, update, and delete items in the database.

In contrast, a function in sql can only return values from tables. They cannot insert, update, or delete records.

4.2 - Syntax of Programming Logic

- The majority of SQL is not case sensitive. SELECT gives the same output as select.
- SQL Statements for our database ends with ;
- SQL Queries can be broken down into 5 major categories:
- DDL: Data Definition Language - Changes the structure of tables. Commands in this category include CREATE, ALTER, and TRUNCATE
  - CREATE - Creates a new table in the database
  - ALTER - Changes the structure of the database. This command can add, remove, or modify attributes of the database
  - TRUNCATE - Removes empty entries in the table, freeing it's space in memory.
- DML: Data Manipulation Language - Modifies the structure of the database. These changes are not auto-committed, and can be rolled back in case of error. Commands included in this category include INSERT, UPDATE, and DELETE
  - INSERT - Inserts data into a row of a table
  - UPDATE - Modifies data values in specific rows of a table
  - DELETE - Removes one or more rows from a table.
- TCL: Transaction Control Language - These commands are used in tandem with DML commands to commit changes to the database. Commands in this category include COMMIT, ROLLBACK, and SAVEPOINT
  - COMMIT - Saves changes made to the database. Used after a DML command
  - ROLLBACK - undoes changes made to database. Used after a DML command
  - SAVEPOINT - Used to roll back changes to a certain point, but without rolling back all changes.
- DCL: Data Control Language - These commands are used to modify user authority within a database. Two commands that fall into this category are GRANT and REVOKE
  - GRANT - Gives users access privileges to a database
  - REVOKE - Removes user access privileges to a database
- DQL - Data Query Language - These commands are used to fetch data from the database. It is used alongside Relational Logic to sort data and return values. The only DQL command is SELECT.
  - SELECT - Operates similar to the project(σ) command in relational algebra. Selects entries based on the conditions set in the query. The SELECT command is modified by various clauses to return different types of data.
    - FROM - Conditional command specifying which table should be searched for data
    - WHERE - the conditional command that specifies what conditions should be set when looking for data in specific sections of the given table. Acts as a conditional ‘filter’ for data
    - SQL also contains various clauses that can help modify data selections and set criteria for the SELECT command.
- Subqueries
  - Nested queries within SQL are known as ‘subqueries’
- SQL can use subqueries to modify the results of a SELECT, INSERT, UPDATE, and DELETE query. These subqueries can be used as logical comparisons against other queries with the '>', '<', '='!, '>=', '<=', 'IN', 'EXISTS', 'NOT EXISTS' and '!=' functions. Subqueries are notated by parenthesis( ).
- GROUP BY: sorts data based upon the given condition
- ORDER BY: sorts data into ascending or descending order based upon given condition. Accepts the inputs ASC for ascending and DESC for descending
- Using the COUNT command, the computer will count the number of items in the result.
- Using the HAVING command, the computer can modify the results of a GROUP BY

4.3 - Implementation

4.3.1 - Views

1. Product query that only shows the customer Product name and customer cost and does not show product id or the store cost.

CREATE View ice_cream_flavors
as
select p.productName as Flavor, p.customerCost as Price
from product as p
where productID > 8;

Select command for view:
Select * from ice_cream_flavors;
results:
MariaDB [amanz]> select * from ice_cream_flavors;
+---------------------+-------+
| Flavor              | Price |
|---------------------+-------|
| Strawberry          |  2.50 |
| Mint Chocolate Chip |  2.50 |
| Salted Caramel      |  2.50 |
| Rocky Road          |  3.50 |
| Coffee              |  3.50 |
| Cookie Dough        |  2.50 |
| Raspberry           |  3.50 |
| Birthday Cake       |  3.50 |
| Vanilla             |  2.50 |
| Chocolate           |  2.50 |
create view stores_products
as
select c.storeID as Store, p.productName as Product
from product as p
inner join contains as c on c.productID = p.productID
where p.productID >2
order by c.storeID;

Command to show new table view:
select * from stores_products;

MariaDB [amanz]> select * from stores_products;

+-------+---------------------+
| Store | Product             |
|-------+---------------------|
|1 | Mint Chocolate Chip |
|2 | Rocky Road          |
|2 | Marshmallow         |
|2 | Raspberry           |
|2 | Caramel             |
|2 | Fudge               |
|2 | Rocky Road          |
|3 | Salted Caramel      |
|3 | Birthday Cake       |
|3 | Vanilla             |
|3 | Chocolate           |
|4 | Strawberries        |
|4 | Coffee              |
|4 | Raspberry           |
|5 | Gummy Worms         |
|5 | Coffee              |
|5 | Salted Caramel      |
3. Join among store, Product, serves, and Customer that displays Customer info, their transactions, and address of store instead of just customer and store ID

create view customer_transactions
as
select c.cFirst as First, c.cLast as Last, p.productName as Purchase, s.location as Store
from customer as c
inner join serves as v on c.customerID = v.customerID
inner join product as p on v.productID = p.productID
inner join store as s on v.storeID = s.storeID;

Command to show view:
select * from customer_transactions;
MariaDB [amanz]> select * from customer_transactions;
+----------+-------------+---------------------+---------------------------+
| First    | Last        | Purchase            | Store                     |
|----------+-------------+---------------------+---------------------------+
<p>| Jonie    | Kneeland    | Chocolate           | 3513 Kingsford Hill       |
| Jonie    | Kneeland    | Marshmallow         | 10 Maple Crossing         |
| Jonie    | Kneeland    | Chocolate           | 2670 Ryan Circle          |
| Jonie    | Kneeland    | Vanilla             | 56 Melvin Center          |
| Jonie    | Kneeland    | Birthday Cake       | 3513 Kingsford Hill       |
| Jonie    | Kneeland    | Raspberry           | 9737 Summit Center        |</p>
<table>
<thead>
<tr>
<th>Name</th>
<th>Bakery</th>
<th>Flavor</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jonie</td>
<td>Kneeland</td>
<td>Cookie Dough</td>
<td>41 Rieder Place</td>
</tr>
<tr>
<td>Jonie</td>
<td>Kneeland</td>
<td>Coffee</td>
<td>10 Maple Crossing</td>
</tr>
<tr>
<td>Jonie</td>
<td>Kneeland</td>
<td>Rocky Road</td>
<td>367 Lighthouse Bay Street</td>
</tr>
<tr>
<td>Jonie</td>
<td>Kneeland</td>
<td>Salted Caramel</td>
<td>421 Eagan Hill</td>
</tr>
<tr>
<td>Jonie</td>
<td>Kneeland</td>
<td>Mint Chocolate Chip</td>
<td>09245 Redwing Center</td>
</tr>
<tr>
<td>Jonie</td>
<td>Kneeland</td>
<td>Strawberry</td>
<td>8 Cordelia Road</td>
</tr>
<tr>
<td>Joyous</td>
<td>Selcraig</td>
<td>Fudge</td>
<td>09245 Redwing Center</td>
</tr>
<tr>
<td>Joyous</td>
<td>Selcraig</td>
<td>Mint Chocolate Chip</td>
<td>9737 Summit Center</td>
</tr>
<tr>
<td>Joyous</td>
<td>Selcraig</td>
<td>Cup</td>
<td>56 Melvin Center</td>
</tr>
<tr>
<td>Joyous</td>
<td>Selcraig</td>
<td>Mint Chocolate Chip</td>
<td>421 Eagan Hill</td>
</tr>
<tr>
<td>Joyous</td>
<td>Selcraig</td>
<td>Mint Chocolate Chip</td>
<td>09245 Redwing Center</td>
</tr>
<tr>
<td>Marcella</td>
<td>Pattingham</td>
<td>Coffee</td>
<td>3513 Kingsford Hill</td>
</tr>
<tr>
<td>Marcella</td>
<td>Pattingham</td>
<td>Birthday Cake</td>
<td>367 Lighthouse Bay Street</td>
</tr>
<tr>
<td>Marcella</td>
<td>Pattingham</td>
<td>Birthday Cake</td>
<td>3513 Kingsford Hill</td>
</tr>
<tr>
<td>Marcella</td>
<td>Pattingham</td>
<td>Vanilla</td>
<td>421 Eagan Hill</td>
</tr>
<tr>
<td>Leta</td>
<td>Powers</td>
<td>Raspberry</td>
<td>9737 Summit Center</td>
</tr>
<tr>
<td>Leta</td>
<td>Powers</td>
<td>Cookie Dough</td>
<td>9737 Summit Center</td>
</tr>
<tr>
<td>Leta</td>
<td>Powers</td>
<td>Strawberry</td>
<td>10 Maple Crossing</td>
</tr>
<tr>
<td>Leta</td>
<td>Powers</td>
<td>Vanilla</td>
<td>9737 Summit Center</td>
</tr>
<tr>
<td>Leta</td>
<td>Powers</td>
<td>Birthday Cake</td>
<td>367 Lighthouse Bay Street</td>
</tr>
<tr>
<td>Leta</td>
<td>Powers</td>
<td>Raspberry</td>
<td>3513 Kingsford Hill</td>
</tr>
<tr>
<td>Claudina</td>
<td>Lindenbluth</td>
<td>Sprinkles</td>
<td>421 Eagan Hill</td>
</tr>
<tr>
<td>Claudina</td>
<td>Lindenbluth</td>
<td>Cone</td>
<td>9737 Summit Center</td>
</tr>
<tr>
<td>Claudina</td>
<td>Lindenbluth</td>
<td>Coffee</td>
<td>56 Melvin Center</td>
</tr>
<tr>
<td>Danyette</td>
<td>Lilliman</td>
<td>Birthday Cake</td>
<td>367 Lighthouse Bay Street</td>
</tr>
<tr>
<td>Danyette</td>
<td>Lilliman</td>
<td>Coffee</td>
<td>367 Lighthouse Bay Street</td>
</tr>
<tr>
<td>Danyette</td>
<td>Lilliman</td>
<td>Chocolate</td>
<td>56 Melvin Center</td>
</tr>
<tr>
<td>Danyette</td>
<td>Lilliman</td>
<td>Birthday Cake</td>
<td>2670 Ryan Circle</td>
</tr>
<tr>
<td>Danyette</td>
<td>Lilliman</td>
<td>Cup</td>
<td>367 Lighthouse Bay Street</td>
</tr>
<tr>
<td>Raff</td>
<td>Ply</td>
<td>Strawberry</td>
<td>56 Melvin Center</td>
</tr>
<tr>
<td>Raff</td>
<td>Ply</td>
<td>Raspberry</td>
<td>3513 Kingsford Hill</td>
</tr>
<tr>
<td>Raff</td>
<td>Ply</td>
<td>Sprinkles</td>
<td>10 Maple Crossing</td>
</tr>
<tr>
<td>Raff</td>
<td>Ply</td>
<td>Vanilla</td>
<td>09245 Redwing Center</td>
</tr>
<tr>
<td>Wyatt</td>
<td>Crissil</td>
<td>Cone</td>
<td>10 Maple Crossing</td>
</tr>
<tr>
<td>Wyatt</td>
<td>Crissil</td>
<td>Fudge</td>
<td>41 Rieder Place</td>
</tr>
<tr>
<td>Wyatt</td>
<td>Crissil</td>
<td>Birthday Cake</td>
<td>56 Melvin Center</td>
</tr>
<tr>
<td>Wyatt</td>
<td>Crissil</td>
<td>Mint Chocolate Chip</td>
<td>3513 Kingsford Hill</td>
</tr>
<tr>
<td>Erek</td>
<td>Pindell</td>
<td>Cup</td>
<td>10 Maple Crossing</td>
</tr>
<tr>
<td>Erek</td>
<td>Pindell</td>
<td>Gummy Worms</td>
<td>8 Cordelia Road</td>
</tr>
<tr>
<td>Nevsa</td>
<td>Cianelli</td>
<td>Chocolate</td>
<td>41 Rieder Place</td>
</tr>
<tr>
<td>Nevsa</td>
<td>Cianelli</td>
<td>»¿Cone</td>
<td>2670 Ryan Circle</td>
</tr>
<tr>
<td>Nevsa</td>
<td>Cianelli</td>
<td>Birthday Cake</td>
<td>09245 Redwing Center</td>
</tr>
<tr>
<td>Nevsa</td>
<td>Cianelli</td>
<td>Sprinkles</td>
<td>421 Eagan Hill</td>
</tr>
<tr>
<td>Nevsa</td>
<td>Cianelli</td>
<td>Cup</td>
<td>421 Eagan Hill</td>
</tr>
</tbody>
</table>
4.3.2 - Stored procedures/functions

1. Adding a new ice cream to the product catalog.

DROP PROCEDURE IF EXISTS AddFlavor;
DELIMITER //
CREATE PROCEDURE `AddFlavor` (idnum INT, IceCreamFlavor VARCHAR(256), SCost DEC, CCost DEC)
BEGIN
    SELECT COUNT(*) INTO @flavorcount
    FROM product
    WHERE productName = IceCreamFlavor;

    IF @flavorcount > 0 THEN
        SELECT NULL AS productID, "Ice cream flavor already in catalog" AS 'Error';
    ELSE
        INSERT INTO product(productID, productName, customerCost, storeCost)
        VALUES (idnum, IceCreamFlavor, CCost, SCost);
        SELECT productID, NULL AS 'Error'
        FROM product
        WHERE productName = IceCreamFlavor;
    END IF;
END;
DELIMITER ;

BEFORE:

<table>
<thead>
<tr>
<th>productID</th>
<th>productName</th>
<th>customerCost</th>
<th>storeCost</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Cone</td>
<td>0.00</td>
<td>40.00</td>
</tr>
<tr>
<td>2</td>
<td>Cup</td>
<td>0.00</td>
<td>20.00</td>
</tr>
<tr>
<td>3</td>
<td>Sprinkles</td>
<td>1.00</td>
<td>10.00</td>
</tr>
<tr>
<td>4</td>
<td>Caramel</td>
<td>1.00</td>
<td>15.00</td>
</tr>
<tr>
<td>5</td>
<td>Fudge</td>
<td>1.00</td>
<td>30.00</td>
</tr>
<tr>
<td>6</td>
<td>Strawberries</td>
<td>1.00</td>
<td>20.00</td>
</tr>
<tr>
<td>7</td>
<td>Marshmallow</td>
<td>1.00</td>
<td>15.00</td>
</tr>
<tr>
<td>8</td>
<td>Gummy Worms</td>
<td>1.00</td>
<td>20.00</td>
</tr>
<tr>
<td>9</td>
<td>Strawberry</td>
<td>2.50</td>
<td>25.00</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>Mint Chocolate Chip</td>
<td>2.50</td>
</tr>
<tr>
<td>---</td>
<td>----</td>
<td>---------------------------</td>
<td>------</td>
</tr>
<tr>
<td></td>
<td>11</td>
<td>Salted Caramel</td>
<td>2.50</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>Rocky Road</td>
<td>3.50</td>
</tr>
<tr>
<td></td>
<td>13</td>
<td>Coffee</td>
<td>3.50</td>
</tr>
<tr>
<td></td>
<td>14</td>
<td>Cookie Dough</td>
<td>2.50</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>Raspberry</td>
<td>3.50</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>Birthday Cake</td>
<td>3.50</td>
</tr>
<tr>
<td></td>
<td>17</td>
<td>Vanilla</td>
<td>2.50</td>
</tr>
<tr>
<td></td>
<td>18</td>
<td>Chocolate</td>
<td>2.50</td>
</tr>
<tr>
<td></td>
<td>19</td>
<td>Avocado</td>
<td>5.00</td>
</tr>
</tbody>
</table>

**CALLING:**

MariaDB [amanz]> call AddFlavor(20, "Banana", 20.00, 3.50);

**AFTER:**

```
+-----------+---------------------+--------------+-----------+
| productID | productName         | customerCost | storeCost |
|-----------|---------------------|--------------|-----------+
|          1 | Cone                | 0.00         | 40.00     |
|          2 | Cup                 | 0.00         | 20.00     |
|          3 | Sprinkles           | 1.00         | 10.00     |
|          4 | Caramel             | 1.00         | 15.00     |
|          5 | Fudge               | 1.00         | 30.00     |
|          6 | Strawberries        | 1.00         | 20.00     |
|          7 | Marshmallow         | 1.00         | 15.00     |
|          8 | Gummy Worms         | 1.00         | 20.00     |
|          9 | Strawberry          | 2.50         | 25.00     |
|         10 | Mint Chocolate Chip | 2.50         | 25.00     |
|         11 | Salted Caramel      | 2.50         | 30.00     |
|         12 | Rocky Road          | 3.50         | 35.00     |
|         13 | Coffee              | 3.50         | 30.00     |
|         14 | Cookie Dough        | 2.50         | 35.00     |
|         15 | Raspberry           | 3.50         | 30.00     |
|         16 | Birthday Cake       | 3.50         | 35.00     |
|         17 | Vanilla             | 2.50         | 20.00     |
|         18 | Chocolate           | 2.50         | 20.00     |
|         19 | Avocado             | 5.00         | 20.00     |
|         20 | Banana              | 4.00         | 20.00     |
```

2. Delete a product ID to discontinue a product.
DROP PROCEDURE IF EXISTS DiscontinueFlavor;

DELIMITER //
CREATE PROCEDURE `DiscontinueFlavor` (idnum int)
BEGIN
    SELECT COUNT(*) INTO @flavorcount
    FROM product
    WHERE productID = idnum;

    IF @flavorcount > 0 THEN
        DELETE FROM product WHERE productID = idnum;
    ELSE
        SELECT NULL as productID, "No product with that ID" as 'Error';
    END IF;
END;
//
DELIMITER ;

BEFORE:

+-----------+---------------------+--------------+-----------+
| productID | productName         | customerCost | storeCost |
+-----------+---------------------+--------------+-----------+
| 1         | Cone                | 0.00         | 40.00     |
| 2         | Cup                 | 0.00         | 20.00     |
| 3         | Sprinkles           | 1.00         | 10.00     |
| 4         | Caramel             | 1.00         | 15.00     |
| 5         | Fudge               | 1.00         | 30.00     |
| 6         | Strawberries        | 1.00         | 20.00     |
| 7         | Marshmallow         | 1.00         | 15.00     |
| 8         | Gummy Worms         | 1.00         | 20.00     |
| 9         | Strawberry          | 2.50         | 25.00     |
| 10        | Mint Chocolate Chip | 2.50         | 25.00     |
| 11        | Salted Caramel      | 2.50         | 30.00     |
| 12        | Rocky Road          | 3.50         | 35.00     |
| 13        | Coffee              | 3.50         | 30.00     |
| 14        | Cookie Dough        | 2.50         | 35.00     |
| 15        | Raspberry           | 3.50         | 30.00     |
| 16        | Birthday Cake       | 3.50         | 35.00     |
| 17        | Vanilla             | 2.50         | 20.00     |
| 18        | Chocolate           | 2.50         | 20.00     |
| 19        | Avocado             | 5.00         | 20.00     |
| 20        | Banana              | 4.00         | 20.00     |
+-----------+---------------------+--------------+-----------+
CALLING:
MariaDB [amanz]> call DiscontinueFlavor(20);

AFTER:

<table>
<thead>
<tr>
<th>productID</th>
<th>productName</th>
<th>customerCost</th>
<th>storeCost</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Cone</td>
<td>0.00</td>
<td>40.00</td>
</tr>
<tr>
<td>2</td>
<td>Cup</td>
<td>0.00</td>
<td>20.00</td>
</tr>
<tr>
<td>3</td>
<td>Sprinkles</td>
<td>1.00</td>
<td>10.00</td>
</tr>
<tr>
<td>4</td>
<td>Caramel</td>
<td>1.00</td>
<td>15.00</td>
</tr>
<tr>
<td>5</td>
<td>Fudge</td>
<td>1.00</td>
<td>30.00</td>
</tr>
<tr>
<td>6</td>
<td>Strawberries</td>
<td>1.00</td>
<td>20.00</td>
</tr>
<tr>
<td>7</td>
<td>Marshmallow</td>
<td>1.00</td>
<td>15.00</td>
</tr>
<tr>
<td>8</td>
<td>Gummy Worms</td>
<td>1.00</td>
<td>20.00</td>
</tr>
<tr>
<td>9</td>
<td>Strawberry</td>
<td>2.50</td>
<td>25.00</td>
</tr>
<tr>
<td>10</td>
<td>Mint Chocolate Chip</td>
<td>2.50</td>
<td>25.00</td>
</tr>
<tr>
<td>11</td>
<td>Salted Caramel</td>
<td>2.50</td>
<td>30.00</td>
</tr>
<tr>
<td>12</td>
<td>Rocky Road</td>
<td>3.50</td>
<td>35.00</td>
</tr>
<tr>
<td>13</td>
<td>Coffee</td>
<td>3.50</td>
<td>30.00</td>
</tr>
<tr>
<td>14</td>
<td>Cookie Dough</td>
<td>2.50</td>
<td>35.00</td>
</tr>
<tr>
<td>15</td>
<td>Raspberry</td>
<td>3.50</td>
<td>30.00</td>
</tr>
<tr>
<td>16</td>
<td>Birthday Cake</td>
<td>3.50</td>
<td>35.00</td>
</tr>
<tr>
<td>17</td>
<td>Vanilla</td>
<td>2.50</td>
<td>20.00</td>
</tr>
<tr>
<td>18</td>
<td>Chocolate</td>
<td>2.50</td>
<td>20.00</td>
</tr>
<tr>
<td>19</td>
<td>Avocado</td>
<td>5.00</td>
<td>20.00</td>
</tr>
</tbody>
</table>

3. Find our customer of the month. 😊

DROP PROCEDURE IF EXISTS CustOfMonth;
DELIMITER //
CREATE PROCEDURE `CustOfMonth`(startDate datetime, endDate datetime)
BEGIN
    SELECT customer.cFirst, customer.cLast, count(*) as transactioncount
    FROM customer INNER JOIN serves ON (customer.customerID = serves.customerID)
    WHERE transDate > startDate AND transDate < endDate
    GROUP BY serves.customerID
    ORDER BY transactioncount desc
    LIMIT 1;
END;
//
DELIMITER ;
CALLING:
MariaDB [amanz]> call CustOfMonth ('2020-04-01', '2020-05-01');

AFTER:
+--------+--------+-------------------+
| cFirst | cLast  | Transaction_Count |
|--------+--------+-------------------|
| Leta   | Powers | 2                 |
+--------+--------+-------------------+

4.3.3 - Triggers

1. Upon deleting a store, cascade to delete all inventories and contains that have that storeID as a foreign key.

CREATE TABLE contains (  
  ...  
  CONSTRAINT `contains_ibfk_3` FOREIGN KEY (`storeID`) REFERENCES `store` ('storeID') ON DELETE CASCADE  
)

2. Upon updating an employee's wage, create a table that houses employeeID, number of hours worked before pay change, and their old wage in order to ensure correct paycheck. This table would be consulted when calculating the paycheck and should be cleared after each pay cycle in order to ensure proper pay in the future.

DELIMITER //
DROP TABLE IF EXISTS WageChange //
CREATE TABLE WageChange(
  changeID INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  employeeID INT UNSIGNED,
  oldWage DECIMAL(10,2),
  oldHours DECIMAL (10,2),
  FOREIGN KEY (employeeID) references employee (employeeID)
) //

DROP TRIGGER IF EXISTS EmployeeWageChange //
CREATE TRIGGER EmployeeWageChange
BEFORE UPDATE ON employee
FOR EACH ROW
BEGIN
  IF OLD.wage != NEW.wage THEN
    INSERT INTO WageChange (employeeID, oldWage, oldHours)
    VALUES (OLD.employeeID, OLD.wage, OLD.hoursWorked);
  END IF;
END;
3. Upon inserting a new Store, insert a new row into Inventory with the new store’s storeID.

```
DELIMITER //
DROP TRIGGER IF EXISTS NewInventory //
CREATE TRIGGER NewInventory
AFTER INSERT ON store
FOR EACH ROW
BEGIN
    INSERT INTO inventory VALUES (NEW.storeID);
END
//
DELIMITER ;
```

---

**Phase 5 - GUI Development**

**5.1 - GUI Functionalities and User Groups**

**5.1.1 - Itemized Descriptions of the GUI**

**Guest/Main:**
When a guest views the page they're able to see a general view of what Yummers stands for and their business ideals. There are four different tabs on the site to view, and depending on the size of the viewing window it could have a dropdown menu to peruse your options. The home tab is what shows when first visiting and it displays an order online button that swaps to a page where you can show what ice cream flavors are available in store. You can check multiple stores as there's an address drop down that will display available flavors based on the store selected. The products tab is your standard business's advertising and features some colorful photos and the Yummers promise of quality.

There will be a login area where if a guest is a customer, they can input their information to go to the appropriate customer view. If the information input matches an employee login, they will receive access to the appropriate employee view.

**Customers:**
Once a customer is logged in, the main view with the flavors and prices will continue to be shown but the login area will be replaced with a logout button and they have a Customer Information tab to view the information associated with the account.

In this Customer Information tab, the first and last name as well as the email address will be displayed. It will also show their total visits, and for every 12 visits, there is a celebratory image saying they will be given 1 free ice cream scoop on their next visit.

Employees:
Once an employee is logged in, the main view with the flavors and prices will be expanded to have In Stock and Expired information for the Store ID associated with that Employee's login. The employee will be able to search for a customer and find their past transactions. These will be displayed below a general summary of the customer's information including the customer's ID number, name, email, and number of visits.

Admin:
Once an Admin is logged in, every piece of information becomes available and different tabs are listed to be able to view whole tables' worth of information. An admin can update the product list with seasonal flavors and delete ones that do not perform well using specified add and delete Product buttons. In the case a store needs to close, an admin can also update this information by pressing the, unfortunate but possibly necessary, Close Store button after entering the Store ID.

5.1.2 - Screenshots and Walkthrough
View when first visiting the site.
https://delphi.cs.csub.edu/~amanz/3420/Yummers/
Order Online has a dropdown menu to select a store to check what is in stock.
https://delphi.cs.csub.edu/~amanz/3420/Yummers/store.php
5.1.3 - Demonstration of Programming Logic

Guests/Main: None of our stored procedures are used with this view. Most of this is just viewing rather than using any of our stored functions.

Customer: Customers used the login to enter the customer view which shows their own information.

Employee: Employees have access to the Discontinue Flavor button which will remove it from the Products list. If one store has an issue with an ice cream, we encourage all stores to discontinue using it.

Admin: Admins can use the Delete Store trigger and Discontinue Flavor stored procedure. Ideally we would allow them to see Customer of the Month as well, but time doesn't seem to permit that, and it would be ideal if admins could search for information from the webpage but time is once again the enemy.

5.2 - GUI Programming

5.2.1 - Server-side Programming

For each view and procedure/function, describe its intended purpose, where it gets used, and by whom. List the code or a screenshot and describe their purposes.
View 1:

The first view is the ice cream flavor menu for “Yummers.”

It is used on the website for anyone to access.

It has a list of the ice cream flavors and their associated price.

This view can be accessed by anyone including:

Guest, Customer, Employee, and Admin

CREATE View ice_cream_flavors
as
select p.productName as Flavor, p.customerCost as Price
from product as p
where productID > 8;

Select command for view:

Select * from ice_cream_flavors;

View 2:

Creates a current table showing the products, both ice cream flavors and toppings, available at each store. This view is accessible by the employees and admin, so they can look to see if any store is out of any product, thus, needs to be ordered. We initially thought that customers could use this view to see which store to visit but we decided that the customer would probably just order a different flavor if they showed up and the ice cream flavor wasn’t available. If the customer looked on the website and saw that their flavor wasn’t available, they might just drive to a different yogurt shop or ice cream parlor.

create view stores_products
as

select c.storeID as Store, p.productName as Product
from product as p
inner join contains as c  on c.productID = p.productID
where p.productID >2
order by c.storeID;

View3:

This view displays the customer names, the product name they purchased, and the address of the store where they purchased the product. It is used by employees an admin to view trends which can assist in marketing ideas and plans.

create view customer_transactions
as

select c.cFirst as First, c.cLast as Last, p.productName as Purchase, s.location as Store
from customer as c
inner join serves as v on c.customerID = v.customerID
inner join product as p on v.productID = p.productID
inner join store as s on v.storeID = s.storeID;

Stored Procedure1:

This procedure adds a new ice cream to the product catalog. The users of this procedure are the admin group and employee group. In order to use this procedure, the productID, product name, customer cost, and the store cost must be given as parameters for the procedure to work.
DROP PROCEDURE IF EXISTS AddFlavor;

DELIMITER //

CREATE PROCEDURE `AddFlavor` (idnum int, IceCreamFlavor varchar(256), SCost dec, CCost dec)
BEGIN

    SELECT COUNT(*) INTO @flavorcount
    FROM product
    WHERE productName = IceCreamFlavor;

    IF @flavorcount > 0 THEN
        SELECT NULL as productID, "Ice cream flavor already in catalog" as 'Error';
    ELSE
        INSERT INTO product(productID, ProductName, customerCost, storeCost) VALUES (idnum, IceCreamFlavor, CCost, SCost);
        SELECT productID, NULL as 'Error'
        FROM product
        WHERE productName = IceCreamFlavor;
    END IF;

END;

//

DELIMITER ;
Stored Procedure 2:

This procedure is the opposite of the previous procedure and deletes a product ID to discontinue a product. The admin group is responsible for enacting this procedure as well. For this procedure to work the only parameter necessary to input is the productID. With this information the procedure will check to make sure the product exists and if it does exist, then it will delete the product from the table.

DROP PROCEDURE IF EXISTS DiscontinueFlavor;

DELIMITER //
CREATE PROCEDURE `DiscontinueFlavor` (idnum int)
BEGIN

    SELECT COUNT(*) INTO @flavorcount
    FROM product
    WHERE productID = idnum;

    IF @flavorcount > 0 THEN
        DELETE FROM product WHERE productID = idnum;
    ELSE
        SELECT NULL as productID, "No product with that ID" as 'Error';
    END IF;

END;
//
DELIMITER ;
Stored Procedure/Function 3:

Our third function is the, "Customer of the month," function. This function can be used by admin and other employees. When you call this function and provide the date range you want for the inquiry, the customer with the most transactions will be displayed on the screen. This information can be used by admin or marketing employees for marketing strategies.

DROP PROCEDURE IF EXISTS CustOfMonth;

DELIMITER //

CREATE PROCEDURE `CustOfMonth`(startDate datetime, endDate datetime)
BEGIN

SELECT customer.cFirst, customer.cLast, count(*) as transactioncount
FROM customer INNER JOIN serves ON (customer.customerID = serves.customerID)
WHERE transDate > startDate AND transDate < endDate
GROUP BY serves.customerID
ORDER BY transactioncount desc
LIMIT 1;

END;

//

DELIMITER ;

In addition to 3 views and 3 procedures, we also implemented 3 triggers to make our database more efficient.

Trigger1:

The first trigger is used only by the admin group and is used during wage increases. When updating an employee's wage, admin creates a table that houses employeeID, number of hours worked before pay change, and their old wage in order to ensure correct paycheck. This table
would be consulted when calculating the paycheck and should be cleared after each pay cycle in order to ensure proper pay in the future.

DELIMITER //

DROP TABLE IF EXISTS WageChange //

CREATE TABLE WageChange(
  changeID INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  employeeID INT UNSIGNED,
  oldWage DECIMAL(10,2),
  oldHours DECIMAL (10,2),
  FOREIGN KEY (employeeID) references employee (employeeID)
)

Once table is created then trigger checks employee’s old wage against the new wage and then updates.

DROP TRIGGER IF EXISTS EmployeeWageChange //

CREATE TRIGGER EmployeeWageChange

BEFORE UPDATE ON employee

FOR EACH ROW

BEGIN

  IF OLD.wage != NEW.wage THEN

    INSERT INTO WageChange (employeeID, oldWage, oldHours)
    VALUES (OLD.employeeID, OLD.wage, OLD.hoursWorked);

  END IF;

END

END
Trigger 2:
The second trigger goes to work when a store is being removed from the database. Only admin will be involved in adding and removing stores from the database.

Upon deleting a store, it is necessary to implement a cascade to delete all inventories and contains that have that storeID as a foreign key.

CREATE TABLE contains (
...
CONSTRAINT `contains_ibfk_3` FOREIGN KEY (`storeID`) REFERENCES `store` (`storeID`) ON DELETE CASCADE
)

Trigger 3:
Our last trigger is also only used by the admin group. Upon opening a new store, a new store is inserted. When this occurs, insert a new row into Inventory with the new store’s storeID.

DELIMITER //
DROP TRIGGER IF EXISTS NewInventory //
CREATE TRIGGER NewInventory
AFTER INSERT ON store
FOR EACH ROW
BEGIN
    INSERT INTO inventory VALUES (NEW.storeID);

5.2.2 - Middle-tier Programming

Here is a screenshot of establishing the connection between client and server on the website:

In order to add a new ice cream to the product catalog use the command:
call AddFlavor(20, "Banana", 20.00, 3.50);
The administrator must type in the productId, the product name, the store's cost for the product and the customer's cost for the product for a successful procedure call.

Here is a screenshot using the AddFlavor procedure:
https://delphi.cs.csub.edu/~amanz/3420/Yummers/employee.php
To delete a flavor from the menu use:
call DiscontinueFlavor(20);
The only parameter necessary for the completion of this procedure is the productID.

Here is a screenshot demonstrating use of the DiscontinueFlavor procedure:
https://delphi.cs.csub.edu/~amanz/3420/Yummers/employee.php

![Screen Shot of Yummers Webpage](https://delphi.cs.csub.edu/~amanz/3420/Yummers/employee.php)

You can find the customer with the most transactions in a given time period by using:
call CustOfMonth ('2020-04-01', '2020-05-01');
This call will provide the customer name and number of transactions of that customer between the given date range. You need two date parameters for successful execution of this function. The dates are needed in yyyy-mm-dd format.

Unimplemented due to time constraints

5.2.3 - Client-side Programming
Due to time constraints we were unable to implement a sorting query, produce charts or graphs, or save reports to a file.