KIWI’S COFFEE SHOP

CMPS 3240 - Fall 2018

Karen Salinas

Group 18

Table of Contents

Phase 1: Conceptual Database Design

The goal of the conceptual database design phase is to collect the fact-finding techniques and gather data to develop a conceptual database model to convert into a logical database model in the future for Kiwi's Coffee Shop. The fact-finding and data gathering in the first phase contains the data views and operations where data is collected and documented to be organized by user groups.

The phase is set into sections to briefly describe the documented information used in the enterprise and conceptual database model. The first section of the phase introduces the enterprise by describing the purpose of the business and its activities. The second section of the phase describes the fact-finding methods that is used to gather data and the operations performed on data to view who will be entering data into the database and what reports are generated from the database. The third section of the phase corresponds to designing the conceptual database for the enterprise and is set to briefly describe the major entity sets and relationship sets included in the database.

The phase contains a fourth section that concentrates on itemizing the entity and relationship sets where each entity is identified. When the entities are identified and described, a relationship between the entities is established. The last section describes the user groups, data views, and operation to describe the group of users who will be accessing the database, and the type of views the user groups will be viewing of all data that is accessible to the user groups. The operations will depend on the user groups since there are employees and customers for the enterprise.

1.1 Fact-Finding Techniques and Information Gathering

The fact-finding techniques and information gathering is the most critical aspect of the initial stages of Kiwi's Coffee Shop database application. The following section will briefly describe details over the techniques used and conceptual database design created for Kiwi's Coffee Shop.

1.1.1 Introduction to Enterprise/Organization

Kiwi's Coffee Shop is an establishment located in Los Angeles serving a wide selection of hot and cold coffee beverages. The menu is available online to give the customer the advantage to select their options in advance before arriving to the establishment. In the coffee shop's website, a preview of available products is displayed to give customers multiple options of hot and cold coffee beverages, including additional items that many not be coffee. After the customer selects a product(s), the customer has the choice to check-out or continue shopping for other available products. When the customer proceeds to check-out, the customer is given the option to cancel or continue the transaction of the order. If the order is transacted, the employee prepares and serves the coffee after they receive the order via online. The customer checks on the status to view the process of the order from start to finish. When the order is done, the customer arrives to the coffee shop and shows the digital receipt to an employee to verify their order to receive their item.

1.1.2 Description of Fact-Finding Techniques

The development of the conceptual database design includes fact-finding techniques and information gathering that enables to acquire knowledge of terminology, requirements, constraints, insight of problems, and the primary concerns of the establishment and the user/groups using the system. The fact-finding techniques used for Kiwi's Coffee Shop database includes research and examination of documentation. The research technique will be used to gather information from the Internet and reference books that provides information over approaching problems and application design. The examination of documents that describes problems affecting coffee shop enterprises and its current system will help gain insight to creating an effective database. The probability of using other fact-finding techniques will depend if the database requires more information. Other techniques includes questionnaires, observation of establishment in operation, and interviews.

1.1.3 Enterprise Conceptual Database Design

The design of the conceptual database for Kiwi's Coffee shop is intended for the customer to order online and have their order prepared by a barista by the time they arrive to the coffee shop. The database focuses on the customers, orders, employees, suppliers, categories, status, and products. The database documents orders placed from customers, where orders contain the items offered for the customer. After the order confirms that enough items are available, the barista employee receives the order and prepares the item. The items prepared are charged to the bill that the customer is required to pay. When the customer pays the bill, the employee gives the customer the product. The employees who manage the coffee shop are in charge of restocking items through the suppliers.

1.1.4 Itemized Description of Entity Sets and Relationship Sets

The appropriate information gathered for the conceptual database sets to construct the E-R model that graphically represent data of the entities and relationships. The following provides a list of entity type descriptions, their relationship to one another, and multiplicities.

Entity Types

Customer: Customer ID, Name, Email, Address, Phone Number

A customer is a person that purchases items online from Kiwi's Coffee Shop. Customers are optionally required to add their personal information. Customers are given the ability to place an order, cancel an order, and/or continue browsing for other items online.

Employee: Employee ID, Name, Role, Email

The employee consists of employed individuals at Kiwi's Coffee shop. Employees have the ability to login to the administrator page to edit, update, and delete entities. Each employee is given a role of either manager, barista, or cashier. The entity contains the email and phone number.

Status: Status_ID, Status

The status is generated when the customer finalizes their order. The status indicates when the order is processed, in preparation, ready to go, and finished. The status indicates whether the customer is ready to pick up their order in the shop.

Category: Category ID, Name

The category is included with the product to divide the products based on their characteristics.

Order: Order ID, Payment, Total

An order is placed by a customer in Kiwi's Coffee Shop. The order is given a unique order ID, payment method, and total amount. The order can be cancelled before checking out. After checking out, the order contains a status where the order is processed by the employee.

Product: Product_ID, Name, Historical Data Price

The product is a good purchased by a customer. The products have names and a historical data price. The price changes dependending the season of the product. The product is supplied by the supplier when stock is running low.

Supplier: Supplier ID, Name, Address, Phone Number

The supplier supplies products requested by the Kiwi's Coffee Shop when product stock runs low. The supplier entity contains a unique supplier ID, supplier name, supplier phone number, and supplier address.

Relationship Types

Customer **Places** Order; *Cardinality:* 1..N; *Participation:* Total/Partial;

Employee **Sets** Order; *Cardinality:* M..N; *Participation:* Total/Total;

Employee **Prepares** Order; *Cardinality:* 1..N; *Participation:* Total/Partial;

Order **Has** Status; *Cardinality:* 1..N; *Participation:* Partial/Partial;

Order **Contains** Products; *Cardinality:* M..N; *Participation:* Total/Total;

Products **Includes** Category; *Cardinality:* N..1; *Participation:* Total/Partial;

Supplier **Supplies** Products; *Cardinality:* N:1; *Participation:* Total/Partial;

1.1.5 User Groups, Data Views, and Operations

The user groups in Kiwi's Coffee Shop consists of customers and employees. The data views of the coffee shop has data accessible for the customer user group to view all of the products from the coffee shop. The employee user group data views all of the data within the administrator page where they can add, remove, update contents. The operations performed by the customer in while shopping in Kiwi's Coffee Shop online website is viewing the adding items in their shopping cart, containing an incomplete order, containing a purchasing history, adding and removing items from the current order, and canceling an order after they have purchased an order. The operations of the employee consists of processing the order and giving the order a status to notify the customer when their order is ready.

1.2 Conceptual Database Design

The conceptual database design for Kiwi's Coffee Shop comprises of modeling entities and their relations to an Entity Relationship Diagram(E-R). The purpose of a conceptual design is to collect all identified requirements for the final physical model. The E-R diagram helps organize the data in the project into entities and define the relationships between the entities. The process of collecting and modeling data produces a good database structure where data can be stored and retrieved in an efficient manner.

The creation of the E-R diagram begins through cases studies where entities are identified. The entities contain roles, events, locations, and concepts that end-users may want to store data. The next step completed in the previous section consists of finding the relationships between the entities. After the relationships are established, the E-R diagram is developed to connect the entities and their relations where cardinalities are used to determine the number of occurrences of one entity for a single occurance of the related entity. The primary and foreign keys are included in each entity where the primary keys are defined for any unique data attributes that contain one occurrence for each entity. Through the process, information collected to fill out the details for the attributes is put in the entities. The process continues with mapping attributes where the attributes are matched to entities.

Finally, the E-R diagram is completed and takes into account all of the process to go through the final stage where the fully attributed final E-R diagram contains the completed data. This consists of the model's entities, attributes, relations, and multiplicities.

1.2.1 Entity Set Description

Name: Category

Description: The category entity stores the category included in the products. Categories are created to organize products within the enterprise. The categories can be deleted by an employee when products no longer share a characteristic. An employee can insert new categories when a product's characteristic does not exist.

Category Attributes:

Attribute Name	Category ID	Name
Description	(Primary Key) Unique identification of category.	The name of the product
Domain / Type	Integer(11)	Varchar (255)
Value / Rang e	1-11 integers	0-255 Chars
Default Value	None	None
Null Value Allowed	No	No
Unique	Yes	No
Single or Multi-value	Single	Multi- Value
Simple or Composite	Simple	Simple

Candidate Keys: Category ID

Primary Keys: Category ID

Strong/Weak Entity: Weak

Fields to be indexed: Category ID

Entity: Customer

Description: The customer entity stores basic information about the customer such as the first and last name, address, state, city, phone number, and email. The customer is assigned a unique customer ID when they log onto the website with their email. The customer is required to insert personal information containing their first and last name, email, phone number, address, state, city, and zip code. Customers are allowed to update the attributes inside the customer's entity, except for customer ID. The customer is able to delete attributes, but are unable to delete the customer ID for record purposes within the coffee shop database.

Customer Attributes:

Attribute Name	Customer ID	Name	Email	Address	Phone Number
Description	(Primary Key) Number assigned to uniquely identify customer	Customer First Name, Middle Name, Last Name	The customer's email address	The address where the customer lives containing the street, city, state, and zip code	The customer's phone number
Domain / Type	Integer	Varchar (65)	Varchar (60)	Varchar (157)	Numeric
Value / Range	1- 11 Integers	20 char, 20 char, 25 char	60 chars	1-157 chars	11 ints
Default Value	0-MaxID	None	None	100 chars, 50 chars, 2 chars, 5 ints	10 ints
Null Value Allowed	No	Yes	No	Yes	Yes
Unique	Yes	No	Yes	No	No
Single or Multi-value	Single	Multi-Value	Single	Multi-Value	Multi-Value
Simple or Composite	Simple	Composite	Simple	Composite	Composite

Candidate keys: Customer ID

Primary key: Customer ID

Strong/Weak Entity: Strong

Fields to be indexed: Customer ID

Name: Employee

Description: The employee entity stores employee information regarding their name, role, address, and email. The manager inserts new employee entries when an employee is hired. Updating entries occurs when an employee is promoted to a new role or changes their email address. Deleting employee entries only occurs when an employee is no longer working for the coffee shop.

Employee Attributes:

Attribute Name	Employee ID	Name	Role	Email
Description	(Primary Key) Unique identification placed to every employee	First name, Middle name, Last Name	The employee's role in the coffee shop such as "Barista", "Cashier", "Manager"	The employee's email address
Domain / Type	Integer	Varchar (20), Varchar (20), Varchar (25)	Varchar (50)	Varchar (60)
Value / Range	1-11 ints	1-65 chars	50 chars	60 chars
Default Value	0-MaxID	None	None	None
Null Value Allowed	No	No	No	No
Unique	Yes	No	No	Yes
Single or Multi-value	Single	Multi- Value	Multi-Value	Single
Simple or Composite	Simple	Composite	Simple	Simple

Candidate Keys: Employee ID

Primary Keys: Employee ID

Strong/Weak Entity: Strong

Fields to be indexed: Employee ID

Name: Order

Description: The order entity stores the payment and total information. The order generates a unique identification to keep track of the order after it had been placed from the customer in the order items. Orders are inserted when the customer places an order. The order entity can not be deleted by the customer and employee since a historical purchase records when an order was cancelled or placed.

Order Attributes:

Attribute Name	Order ID	Payment	DateTime	Total
Description	(Primary) Unique identification of status	The type of payment method (Cash, Credit Card)	Time stamp yyyy-mm-Dd hh:mm:ss	The total for the order
Domain / Type	Integer	Varchar(255)	4 chars, 2 chars, 2 ints, 2 ints, 2 ints, 2 ints	Numeric
Value / Rang e	1-11 ints	1-255 characters	yyyy-mm-dd hh:mm:ss	(10,2)
Default Value	0-MaxID	None	No	None
Null Value Allowed	No	No	No	No
Unique	Yes	No	Single	No
Single or Multi-value	Single	Multi-Value	Multi-Value	Single
Simple or Composite	Simple	Simple	Simple	Simple

Candidate Keys: Order ID

Primary Keys: Order ID

Strong/Weak Entity: Weak

Fields to be indexed: Order ID

Name: Products

Description: The product entity stores a unique primary key consisting of the product ID. The product contains the product's name, and the historical data price. The products contain a historical data price since the product's price changes depending on the season of the coffee beans. Inserting tuples exist when there are new products. The new price is updated, whereas the current and past price can not be changed. The deletion cannot be used to delete the product tuples since the discontinuity of a product is identified, and is kept for recorded data.

Products Attributes:

Attribute Name	Product ID	Name	Product Historical Price
Description	Unique identification of the product	The name of the product	Historical data price of product
Domain / Type	Integer	Varchar (255)	Numeric
Value / Range	1-11 integers	0-255 Chars	(10,2)
Default Value	0-MaxID	None	None
Null Value Allowed	No	No	No
Unique	Yes	No	No
Single or Multi-value	Single	Multi- Value	Single
Simple or Composite	Simple	Simple	Simple

Candidate Keys: Product ID

Primary Keys: Product ID

Strong/Weak Entity: Strong

Fields to be indexed: Product ID

Name: Status

Description: The status entity tracks the status of the order to identify if the order is ready or not ready. The system allows for employees to update when the order is being processed, when the order is ready to go, and when it is done. The status is not deleted to keep record of the orders for the same month, otherwise if the online status is outdated, the employee is able to delete tuples of data to keep the data up to date.

Status Attributes:

Attribute Name	Status ID	Status
Description	(Primary Key) Unique identification of status	The status of the order after the customer places the order, indicating a “New Order”, “In Preparation”, “Ready to Go” and “Done”.
Domain / Type	Big Integer	Varchar(40)
Value / Range	1-11 Integers	40 Chars
Default Value	0-MaxID	New Order, In Preparation, Ready to Go, Done
Null Value Allowed	No	No
Unique	Yes	No
Single or Multi-value	Single	Multi-Value
Simple or Composite	Simple	Simple

Candidate Keys: Status ID

Primary Keys: Status ID

Strong/Weak Entity: Weak

Fields to be indexed: Status ID

Entity: Supplier

Description: The supplier entity supplies products when supplies are running low in Kiwi's Coffee Shop. The supplier information is exhibited consisting of the supplier's name, phone number, street, state, city, and zip code. The supplier's street, phone number, state, city, and zip code can be updated by the employee when the supplier reports changes within their business. The employee is able to insert new suppliers when the coffee shop contracts additional suppliers. The employee is able to delete a supplier when the coffee shop no longer possesses a contract with the supplier, or when the supplier informs of no longer being in business.

Supplier Attributes:

Attribute Name	Supplier ID	Supplier Name	Supplier Phone Number	Supplier Street	Supplier State	Supplier City	Supplier Zip Code
Description	Primary Key of the Supplier	Name of the Supplier	The supplier's phone number	Supplier's Street	The supplier's state	The supplier's city	The supplier's zip code
Domain / Type	Integer	Varchar (40)	Numeric (11)	Varchar (100)	Varchar (3)	Varchar (40)	Integer
Value /Range	1-11 Ints	1-40 Chars	11 ints	1-100 Chars	3 Chars	1-40 Chars	5 ints
Default Value	0-MaxID	None	0000000000 - 9999999999	None	2 Chars	None	00000-99999
Null Value Allowed	No	No	No	No	No	No	No
Unique	Yes	No	No	No	No	No	No
Single or Multi-value	Single	Single	Multi- Value	Multi-Value	Single	Single	Single
Simple or Composite	Simple	Simple	Composite	Simple	Simple	Simple	Simple

Candidate keys: Supplier ID

Primary key: Supplier ID

Strong/Weak Entity: Strong

Fields to be indexed: Supplier ID

1.2.2 Relationship Set Description

Relationships is an association between several entities. Furthermore, a relationship set is a set of relationships of the same type. Usually, whenever an attribute of one entity type refers to another entity type, some sort of relationship exists.

Relationship: Customer ***places*** Order
Description: The customer places more than one orders.
Entity Set Involved: Customer, Order
Mapping Cardinality: 1:N
Descriptive Field: DateTime
Participation Constraint : Total participation for the customer, and partial participation for the order. The customer always exist before the order, and the order only exist when the customer places the order.

Relationship: Employee ***sets*** Status
Description: The employee sets the status when the order is being prepared, ready to go, and done.
Entity Set Involved: Employee, Status
Mapping Cardinality: M:N
Descriptive Field: DateTime
Participation Constraint : Total participation for the Employee, and total participation for Status. An employee will always exist and may set multiple status. The status will already exist when an order had been placed from the customer.

Relationship: Employee ***prepares*** Order
Description: The employee prepares more than one orders.
Entity Set Involved: Employee, Order
Mapping Cardinality: 1:N
Descriptive Field: DateTime
Participation Constraint: Partial participation for the Order, and total participation for the Employee. An order must be prepared by employee. Furthermore, more than one orders may exist and there could be one employee with the role "barista" preparing them.

Relationship: Order ***has*** Status

Description: The order has a status that had been placed.
Entity Set Involved: Order, Status
Mapping Cardinality: 1:N
Descriptive Field: DateTime, Total
Participation Constraint: Partial participation for Order and partial participation for Status. A status can not exit if an order does not exist. A status may contain many status.

Relationship: Order ***contains*** Products
Description: The order must contain products.
Entity Set Involved: Products, Order
Mapping Cardinality: M:N
Descriptive Field: DateTime
Participation Constraint: Total participation for Product and Total participation for Orders. An order may contain more than one products. There will always be products available for an order.

Relationship: Products ***includes*** Category
Description: The products must include a category.
Entity Set Involved: Products, Categories
Mapping Cardinality: N:1
Descriptive Field: None
Participation Constraint : Total participation for products, partial participation for categories. There will be one to many products in a category.

Relationship: Supplier ***supplies*** Products
Description: The supplier supplies products to restock inventory.
Entity Set Involved: Supplier, Product
Mapping Cardinality: N:1
Descriptive Field: Quantity, Date Time
Participation Constraint: Total participation for Supplier and partial participation for Product. The supplier always exist in the database since they are supply the products. The products do not exist when they are out of stock, and can exist only when they are reordered and supplied from the supplier.

1.2.3 Related Entity Set

Specialization consists of entities divided into sub-grounds based on their characteristics. Higher level entities are specialized in lower level entities. For instance, the employee entity contains positions that specializes into three categories, the highest being management, the middle being baristas, and the lowest being cashiers.

Generalization is the process of gathering common properties from a set of entities and creating a generalized entity from it. The process occurs by gathering two higher level entities with common attributes. For instance, the customer and the employee can be generalized to a higher entity called a Person since both share common attributes such as name, phone number, and address.

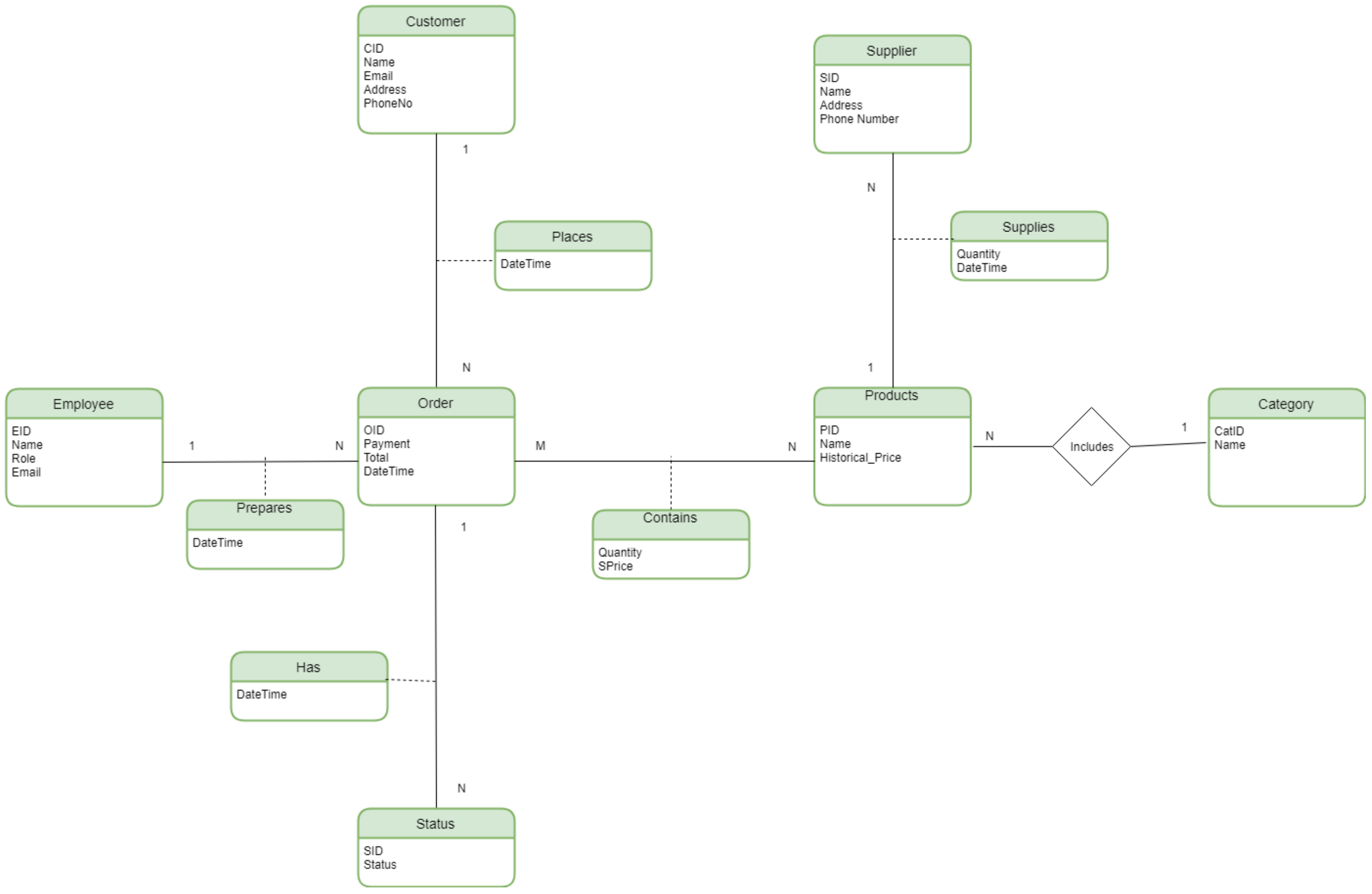
Specialization and generalization share disjointedness and total participation constraints. The case of a subclass type of a person can either be a customer or an employee, but not both. In the case total participation constraint, the specialization of employee is total since there are three possible positions consisting of manager, barista, and cashier.

Aggregation is when a relationship corresponding entity are into a higher-level entity. In the case of the coffee shop database, entities are not aggregated.

1.2.4 E-R Diagram

An Entity-Relationship (ER) Diagram graphically represents entities and their relationships to one another. The E-R Diagram consists of entities, attributes, and multiplicities. An entity is an object (person, place, or event) that can have data stored about them. The attribute contains the characteristics of entities that comprises a primary key (unique or foreign). A multiplicity contains the cardinality and participation constraints. The cardinality represents the maximum number of possible relationship occurrences in which a specific entity can participate in. Three degrees of cardinalities are one-to-one (1:1), one-to-many (1:M), and many-to-many (M: N) relationships. The participation constraints (total and partial) represent if all or only some entity occurrences participate in a relationship. The following E-R Diagram displays the entities, attributes, and multiplicities briefly described for Kiwi’s Coffee Shop Enterprise.

Kiwi’s Coffee Shop E-R Diagram



Phase 2: Conceptual Database and Logical Database

The purpose of Phase 2 is to convert the conceptual database to a logical database. The conceptual database model defines what the system contains in terms of entities, attributes, and relationships. In contrast, the logical database model defines how the system should be implemented and adds further information to the conceptual model elements. Furthermore, the following sections will specify the steps of the conversion of the conceptual database model to a logical database model. This is accomplished by using the current E-R diagram model to a relational model.

The first section of Phase 2 is the expanded description of the E-R model and relational model. The second section specifies compared and differentiated the E-R model and relational model by exhibiting their advantages and disadvantages. The section advances in converting entity and relationship types to relations, and progresses into the database constraints.

The third section of Phase 2 converts the E-R model database to a relational database design. The section proceeds to state the relational schemas of Kiwi's Coffee Shop local database. The sample data of the relations is exhibited by the design relation instance for each relation that corresponds to entity sets that contain ten tuples, and relations which corresponds to relationship with more than sixty tuples.

The last section of Phase 2 contains a brief description of the sample queries for Kiwi's Coffee Shop. The section proceeds to the design of the queries in three formal querying languages of relational algebra, tuple relational calculus, and domain relational calculus. Further, the section includes the domain relational calculus expressions for the queries.

2.1 E-R model and Relational model

The E-R model and relational model are two different database designs. The E-R model is contained within the conceptual database design and the relational database model is a representative of the logical database model. The following will explain a description of the E-R and relational model include the history, major features, and objective. Further, the models are compared with their similarities and differences in connection with the advantages and disadvantages.

2.1.1 Description of E-R model and relational model

The Entity-Relationship (ER) Model was first introduced and published by Dr. Peter Chen in 1976. Dr. Chen was influenced by the previous work of one colleague, Charles Bachman. Furthermore, Dr. Chen's work started a new field of research and practice in Conceptual Modeling. Dr. Chen's work has been modified to the Enhanced Entity-Relationship Model (EER) that is a high-level conceptual data model containing the extensions of the original E-R model.

The E-R model is a popular high-level conceptual data model in the form of a graphical representation that is particularly used for database applications and database design tools to exhibit entities and their relationship to each other. In addition, the E-R model contains diagrammatic notation known as E-R diagrams.

The major features in the E-R model includes entities, attributes, cardinalities, and relationships. The entities represent real-world objects whether they are inanimate or animate, and are easily identifiable. Entities can be set in a collection called entity sets and share similar types of entities with similar attributes within attributes. The attributes in the E-R model contains entities that are displayed using properties, including values. In addition, domain and range of values that can be assigned to attributes exists, where the types of attributes consist of simple, composite, derived, simple-valued, and multi-valued attributes. The collection of attributes is called keys and uniquely identifies an entity within an entity-set by super, candidate, and primary keys. Moreover, the relationship sets are a set of relationships of the similar type, and contains attributes called descriptive attributes. The degree of a relationship is binary, ternary, and n-ary that defines the degree of the relationship. Furthermore, cardinalities define a number of entities in one entity set and are associated with a number of entities from other relationship sets. For this reason, cardinalities can be mapped in one-to-one, many-to-one, and many-to-many.

The purpose of the E-R model is to construct a reflection of the real world in a database, and gives an intermediate step to simply define a database. The E-R model allows developers to design a database with entity-relationship diagrams.

The Relational Model was first introduced and published by Dr. E. F. Codd in 1969 in his paper, “A relational model of data for large shared data banks”. The paper is widely accepted as part of the development in database systems, although other set-oriented models had been previously introduced. Furthermore, the relational model became available in the early 1980’s in the SQL/DS system on the MVS operating system by IBM and Oracle DBMS. The current state of the relational model (DBMS or RDBMS) is contained in DB2 and Informix Dynamic Servers, Oracle and Rdb, SQLserver and Access, including PostgresQL and MySQL.

The relational model is the conceptual basis of relational databases, and is a method of structuring data using relations that consists of grid-like mathematical structures of columns and rows. The model depicts the database as a collection of relations. The physical form of a relational model is a table. In addition, the relational model uses mathematics and mathematical terms such as domains, unions, and ranges, though uses features and conditions that are exhibited in simple English.

The major features in a relational model consists of tables with characteristics of tuples, attributes, domains, and relations. In addition, the model includes features like integrity constraints, domain constraints, key constraints and constraints on NULL values, including the superkey, candidate key, and primary key. The notation in a relational model sets it apart from the conceptual E-R model, whereby is a reference point when creating tables in SQL programs and allows a simple way to transfer dependencies and re-create composite keys in a graphical user interface (GUI). The relational model has modification operations including Insert, Delete, and Update.

The purpose of a relational model is to offer a declarative method for establishing data and queries where users directly declare what information they want from a database, and allows the DBMS to take charge of specifying data structures for storing the data and retrieval procedure to answering queries. In overall context, the relational model is used for organizing and understanding the structure of a relation.

2.1.2 Comparison of Two Different models

The E-R model and relational model is compared based on their description, relationship, and mapping. The advantages and disadvantages of both models will be specified in the following subsection. In addition, both models have their differences where the E-R model is entity specific, and relational model is table specific and are similarly both models are types of data modeling which is approached to physically design a database.

Advantages and Disadvantages

In regard to the E-R model, the advantages and disadvantages will be briefly stated. The advantage of the E-R model includes of straightforward relation representation after designing an E-R diagram for a database application. In addition, another advantage includes easily converting the E-R diagram to another model. Two disadvantages of the E-R model contain of not having industry standard notation for developing an E-R diagram, and is popular for a high-level design.

Furthermore, in conjunction to the E-R model, the relational model has advantages and disadvantages that is differential from the previous model. The advantages of the relational model comprise ease of use since rows and columns are easier to understand, flexibility of information between different tables that are linked and extracted to manipulate operators. In addition, other advantages include precision in the usage of relational algebra and relational calculus, implemented security control and authorization by simply moving sensitive attributes in a given table into a separate relation that contains its own authorization controls, data independence which easily achieve with normalization structure, and data manipulation. The disadvantages in the relational model includes performance when the number of tables between relationships are large, physical storage consumption, and slow extraction of meaning from data.

Differences and Similarities

The E-R model and relational model is compared in this subsection on their description, relationship, and mapping. The E-R model exhibits the collection of objects called entities and relations. In contrast, the relational model represents the collection of tables and the relation between those tables. The E-R model describes data as entity-set, relational-set, and attributes. Conversely, the relational model describes data in a table as domain, attributes and tuples. The relationship of the E-R model is easier to understand the relationship between entities, whereas the relational model is less easy to reach a relation between tables in a Relational Model. The mapping in the E-R model consists of mapping cardinalities, whereas the relational model does not map cardinalities.

2.2 From Conceptual Database to Logical Database

The following subsection will describe the process of converting the E-R model entity types to the relations of the relational model. The methods will describe the conversion of the E-R model and relational model that consists of strong and weak entities, simple and composite attributes, and single and multi-valued attributes.

The next subsection includes the conversion of the relationship types to relations. The subsection gives a description of the used methods for converting the relationship types with one-to-one, one-to-many, many-to-one, and many-to-many relationships. In addition, the subsection includes the descriptions of the 'IsA' and 'HasA' relationships, along with relationship types involving other relationship types, recursive relationships that involves one entity types, relationships involving more than two entity types, including relationship and category types.

The last subsection will discuss the database constraints that briefly describes what constraints are, and the purpose of constraints. The constraints defined and how DBMS enforces the constraints includes entity constraints, primary key and key constraints, referential constraints, check constraints and business rules.

2.2.1 Converting Entity Types to Relations

Strong Entity Types

The strong entity in the E-R model is described of comprising of a primary key, independent of any other entity in a schema, conveyed as a single rectangle, may or may not have total participation in a relationship, and the relation between two strong entities is denoted by a single diamond called a relationship. In the relational model, the strong entity is converted to a relation in the following description. The entity type becomes a table, the single-valued attributes becomes a column, derived attributes are ignored, composite attributes are represented by components, multi-valued attributes are represented by a separate table, and key attributes of the entity type becomes the primary key of the table. In addition, the simple attributes are included in the relation.

Weak Entity Types

The weak entity in the E-R model contains a partial discriminator key where it depends on the strong entity for its existence. The weak entity is conveyed with a double rectangle and relationship shared between the weak and strong entity is denoted by a double diamond, and always has total participation in the identifying relationship displayed by a double line. The conversion of a weak entity type to a relation is transformed as its own table, with the primary key of the strong entity acting as a foreign key in the table. The foreign key, including the key from the weak entity, forms the composite primary key of the table.

Simple and Composite Attributes

The simple attributes are not converted in the relation, they remain unchanged and are transferred directly to the relational model. The composite attributes only create attributes in the relation for their simple components. For this reason, the composite attribute is a set of simple component attributes.

Single and Multi-Valued Attributes

The single-valued attributes of an entity become the attributes of a relation. This occurs in the process of creating a table with simple single-valued attributes.The multi-valued relation is converted into a relation with a composite primary key containing an attribute value, including the primary key of the attribute's entity. This is accomplished when the attribute of an entity identifies the corresponding relation. Further, the relation is generated representing the attribute when the relation contains the simple, single-valued attribute. This leads for the addition of two primary key attributes that includes creating a foreign key reference for the primary key attributes. Moreover, the primary key is a composite key containing of the primary key of a single-valued attribute and primary key attribute.

2.2.2 Converting Relationship Types to Relations

The following section will deliberate the methods for converting relationship types to relations. The entity types converted includes: Cardinality Constraints (1:1, 1:N, M:N), superclass and subclass (IsA) relationship, 'HasA' relationship, a relationship type involving other relationship types, recursive relationships that involves one entity type, relationships involving more than two entity types, relationship and categories (union) types.

Mapping of binary relationship types with 1:1 cardinality constraint

The 1:1 relationship type will be converted to relations in the method of mapping the relationship types. The binary 1:1 relationship R identifies the relation schemas that belongs to entity types participating in R. The method encourages adding single-value attributes of relationship types as attributes of R. The following method is split into three approaches involving the foreign key approach, merged relationship approach, and cross-reference approach.

1. Foreign key Approach:

The foreign key approach comprises of adding a primary key of one participating relation as foreign key attribute of the other, and be represented as R. In addition, if only one side is total participation, then it must be represented as R. The foreign key attribute should be unique.

2. Merged relationship approach:

The merged relationship approach consists of combining the two relation schemas into one, therefore representing R. In addition, one of the primary keys should be unique.

3. Cross-reference approach:

The cross-reference approach consists of creating a new relation schema for R with two foreign key attributes both being duplicates of primary keys. The approach requires for one of the attributes to b e declared as a primary key, and the others as unique.

Mapping of binary relationship types with 1:N and N: 1 cardinality constraint

The cardinality constraints of 1: N (and N: 1) of relationship types will be converted to relations in the method of mapping the relationship types. The binary 1: N relationship consists the primary key of the relation on the '1' side to become a foreign key in the relation on the 'N' side. The following method is split into two approaches involving the foreign key approach, relationship relation approach using the relation schema with symbol S.

1. <u>Foreign Key Approach:</u>

The foreign key approach comprises of identifying the relation schema S which represents the participating entity type at the N-side of the 1: N relationship type. The approach requires for a primary key of the other entity type (1-side) as foreign key in S.

2. <u>Relationship relation approach:</u>

The relationship relation approach consists of creating a new relation schema for S with two foreign key attributes being duplicates of both primary keys. The foreign key attributes are declared for the relation schema corresponding to the participating entity type on the N-side as a primary key. This includes the single-valued attributes of relationship type as attributes of S.

Mapping of binary relationship types with M: N cardinality constraint

The cardinality constraints of M: N of relationship types will be converted to relations in the method of mapping the relationship types. The binary M: N relationship creates a new table that represents the relationship and contains two foreign keys among each of the participants in the relationship. In addition, the primary key of the new table is merges with the two foreign keys. The following will expand the binary M: N relationship type information. This is achieved when a new relation S is created for each binary M: N relationship type.

The method is approached when the primary key of participating entity types is included as foreign key attributes in S. The attributes are generated as primary key of S. This includes any simple attributes of relationship type in S.

Mapping of superclass / subclass for the 'IsA' relationship

The E-R diagram involves relationship types that can be specified by one class as a subclass of another by creating an 'IsA' relationship. This occurs when an entity type contains certain entities that have special properties not shared by all entities, and suggest two entity types merge with an 'IsA' relationship type. The 'IsA' relationship is converted to a relation by mapping the superclass and subclasses in two options involving multiple relation schemas and single relation schemas. The multiple relation schemas focus on subclasses determined by relation schemas, and single relation schemas focuses on the subclasses determined by type attributes.

1. **Multiple relation schemas – superclass and subclass**:
The approach proceeds to mapping the superclass and all subclasses to their own relation schema. In addition, the approach work for any class hierarchy, though will result in single-attribute relation schemas for subclasses without specialized attributes. The relation schemas each includes all attributes that are part of their own entity type, in addition with the same primary key that comes from the superclass. The subclass relation schema primary key can also be a foreign key to the superclass relation schema.

2. **Multiple relation schemas – subclass:**
The multiple relation schema approach follows the mapping of the subclass, and only works for subclasses that are total disjoint specialization. The approach is not advised for overlapping specializations since it would lead copies of entities in a subclass relation. In addition, the approach creates relation schemas for each subclass that contains all the attributes of the subclass, including all the attributes of the superclass, and a primary key selected from the superclass.

3. **Single relation schema – one type attribute:**
The single relation schema approach creates a single relation schema with all the attributes of the superclass and subclasses, additionally with a type attribute with a value that informs where each tuple belongs to which class. The approach is effective for subclasses that are disjoint. As a consequence, this may result in many NULL values if subclasses have attributes.

4. **Single relation schema – multiple attributes:**
The approach is preceded by the creation of a single table with all the attributes of the superclass and subclass, adding Boolean type attributes for each subclass whose values informs if a tuple belongs to that particular subclass. The approach is effective for overlapping subclasses and disjoint subclasses. This may result in many NULL values if the subclasses contain many attributes.

Mapping of 'HasA' relationship

The 'HasA' relationship differs from 'IsA' relationship since it is dynamic binding based on composition, whereas 'IsA' relationship is static binding based on inheritance. The 'HasA' in the E-R model uses instance variables that are reference to other objects. Furthermore, the 'HasA' relationship is a composition relationship where an object belongs to another object. In other words, the entity belongs to multiple subclasses. The 'HasA' relationship shares two similar methods to the 'IsA' relationship. The two methods are multiple relations containing the superclass/ subclass, and the single relation with multiple type attributes.

Mapping relationship types involving other relationship types

The relationship type is mapped when an entity or relationship is linked with another relationship type (either an entity or relationship) to create a primary key for the relationship. The approach of consisting a primary key for each relationship is led by the foreign key approach or cross reference approach, and is utilized depending on the cardinality.

Mapping of Recursive relationships

The recursive relationship in the E-R model is converted by mapping the recursive relationships in one-to-many and many-to-many relations. The recursive relationship is transformed in the one-to-many as a recursive foreign key contained in the same relation. Moreover, the recursive relationship in the many-to-many is converted into two relations. The first relation is for the entity type, and the other is for an associative relationship in which the primary key contains two attributes, both taken from the primary key of the entity.

Mapping of relationships with more than 2 entity types

The mapping approach of a relationship with more than two entity types occurs when a relationship type R is linked with more than two entity types. The relationship type R contains a primary key for the participating entity types as foreign keys. The foreign keys are merged to create the primary key of the relationship type R, though some foreign keys can be omitted from the primary key if their relation of an entity is on the 1: N cardinality constraint.

Mapping of relationship and categories (or union) types

The E-R model exhibits a union type (or category) as a collection of objects (entities) that is the union of objects of different entity types. The union type uses the mapping method to convert to a relation. The method is approached in two processes of the superclass containing the same key, and the other a different key. The case of the superclass containing a different key is preceded by a new key attribute, the surrogate key. The surrogate key is a foreign key in each relation belonging to a superclass, including that the type attribute to the category relation. The second case is approached when the superclasses have the same key without a surrogate key. The superclasses consisting of the same key must contain a primary key from the superclasses.

2.2.3 Database Constraints

The database constraints serve as restrictions on the actual values in a database states and are derived from the rules in the database mini-world. The purpose of database constraints is to be part of the database schema to ensure the integrity of the data being stored into the table columns. The database constraints in this section involves domain constraints, key constraints, constraints on NULLs, entity integrity constraints, and referential integrity constraints. The constraints can be divided into three main categories including constraints that are inherent model-based or implicit constraints, constraints directly expressed in schemas of the data model when specified in the data definition language called schema-based constraints or explicit constraints.

The last main category called application-based (other names includes semantic constraints or business rules) includes constraints that can not be directly expressed in the schemas of the data model and is required to be expressed and implemented by the application programs. The following will expand the description of database constraints and how database management system enforces the constraints of entity constraints, primary key and unique key constraints, referential constraints, check constraints and business rules.

Entity Integrity Constraint

The entity integrity constraint states that a primary key value can not be NULL. This exist since the primary key value is used to identity individual tuples in a relation. The NULL values for the primary key denote that some tuples can not be identified. Furthermore, a table can contain a NULL value other than the primary key field. The DBMS enforces for the entity integrity constraint by not allowing operations such as INSERT and UPDATE to produce an invalid primary key. The system rejects any operation creates duplicate primary key or one containing NULLs.

Primary and Unique Key Constraint

The primary key constraint helps identify a unique tuple from a table, and does not allow NULL values. The table can only consist of one primary key constraint that may consists of single or multiple fields. The primary index is created on the column where the primary key is defined. The foreign key can refer to the primary key. The primary key must contain unique values and can not contain any NULL values. The unique key constraint consists of one or more columns as an alternative to a primary key constraint. The unique key constraint enforces uniqueness and permits NULL values. These constraints are useful when primary keys are already established on a table and ensures that values in one or more columns are unique. The DBMS in the unique key constraint creates a clustered or non-clustered index to enforce the uniqueness. The case when the index type is not specified results for the DBMS to create a non-clustered index, though a clustered unique index can be created if no clustered index is defined on the table. The DBMS in the primary key constraint enforces uniqueness of the primary key constraint by creating a clustered or non-clustered unique index. The DBMS creates a clustered index when no other clustered index exists when the primary key is created without specifying the index type. In addition, all primary keys in the columns must be configured as NOT NULL.

Referential Constraint

The referential integrity constraint is described between two relations and is used to maintain the consistency among tuples in the two relations. The constraint states that a tuple in one relation that refers to another relation must refer to an existing tuple in that relation. Furthermore, the referential integrity constraint is the mechanism of foreign keys maintained in a system. The prior description of one relation that refers to another relation is applied when the constraint requires valid foreign keys must always reference to an existing primary key or contains a NULL. The foreign key does not require to reference an existing primary key. The DBMS enforces for the referential integrity constraint where the foreign key must follow the rules by no operation (INSERT, UPDATE) can create a non-NULL foreign key unless a corresponding primary key exists. The referential integrity constraint in the DBMS enforces the primary key where no operation (UPDATE, DELETE) can remove or change a primary key while a referencing foreign key exists.

Check Constraint and Business Rules

The check constraint is a rule that identifies acceptable column values for data in a row with the DBMS. The constraint helps enforce domain integrity and can validate the domain integrity of a single column or multiple of columns. The check constraint is used to limit the value range that can be placed in a column. This works when the check constraint is defined on a single column that allows only certain values to embody the column or implemented in a table that can limit the values in certain columns based on values in other columns in the tuple. In the DBMS, the check constraint can be defined in either a CREATE TABLE statement or an ALTER TABLE statement, including the DROP CONSTRAINT. The check constraints goal when building application is to make sure that data that is entered in the database meets all the business rules (procedures or constraints). The data validation is a critical part of the application and sets requirements that data must meet when it is developed.

2.3 Converting the Conceptual Database into a Relational/Logical Database

The following section will convert the conceptual model to a logical database for Kiwi's Coffee Shop. The first subsection contains the relational schema in the logical design of the database specifying the attributes, domain, and all constraints consisting of entity/primary keys, referential constraint, business constraints that the relation may contain, including candidate keys. The second subsection samples the data of the relation by designing the relation instances for each relation containing tuples in the table body. The relations that correspond to entity sets contain ten tuples and relations corresponding to relationship sets contain sixty to one hundred tuples.

2.3.1 Relation Schema for Your Local Database

The relational schema of the local database will specify the attributes and their domain, all constraints consisting of the entity/primary key, referential and business constraints, and candidate keys. The entity sets from the conceptual database design containing the entity, attributes, and their types are included in the following relational schemas.

Relation Schema: Category

Category(<u>catid</u>, name)

Attributes	Domain
catid	Integer, 11 ints, Primary key
name	Varchar(255)

Candidate Keys: catid

Entity Integrity/Primary Key Constraint: “catid” is the primary key and cannot be NULL

Uniqueness Constraint: “catid” is unique.

Business Constraint: Name cannot be NULL.

Referential Constraint: None

Derivation: Category

Description: Derives from Category entity.

Relation Schema: Customer

Customer(<u>cid</u>, fname, mname, lname, email, street, city, state, zipcode, phone_no)

Attributes	Domain
cid	Integers, 11 Integers, Primary key
fname	Varchar(1-20)
mname	Varchar(1-20)
lname	Varchar(1-25)
email	Varchar(1-60)
street	Varchar(1-100)
city	Varchar(1-50)
state	Varchar(2)
zipcode	Varchar(5)
phone_no	Numeric(11)

Candidate Keys: cid

Entity Integrity/Primary Key Constraint: "cid" is primary key and cannot be NULL

Uniqueness Constraint: "cid" and "email" are unique.

Referential Constraint: None

Business Constraint: First name, last name, and email cannot be NULL.

Derivation: Customer

Description: Derives from Customer entity.

Relation Schema: Employee

Employee(eid, fname, mname, lname, role, email)

Attributes	**Domain**
eid	Integer, 11 ints, 0–MaxID, Primary key
fname	Varchar(1-20)
mname	Varchar(1-60)
lname	Varchar(1-50)
role	Varchar(1-50)
email	Varchar(1-60)

Candidate Keys: eid

Entity Integrity/Primary Key Constraint: eid is the primary key and cannot be NULL

Uniqueness Constraint: "eid" and "email" must be unique.

Business Constraint: First name, last name, and email cannot be NULL.

Referential Constraint: None

Derivation: Employee

Description: Derives from the Employee entity.

Relation Schema: Orders

Orders(o<u>id</u>, <u>cid</u>, payment, datetime, total)

Attributes:	Domain:
oid	Integer, 11 Integers, Primary key
payment	Varchar (1-255)
datetime	Date: yyyy-mm-dd, Time: HH:MM:SS
total	Numeric (10,2)
cid*	Integer, 11 ints

Candidate Keys: oid

Entity Integrity/Primary Key Constraint: “oid” is primary key and cannot be NULL

Uniqueness Constraint: “oid” is unique.

Referential Constraint: “cid” has to be a valid Customer primary key reference to be a foreign key.

Business Constraint: Payment, total, and datetime cannot be NULL. Datetime is generated when status is changed.

Derivation: Orders

Description: Derives from Orders entity. Products ‘contains’ orders, orders ‘has’ status, and employee ‘prepares’ orders. Derives from Customer ‘places’ orders.

Relation Schema: Products

Products(pid, name, hdprice)

Attributes	Domain
pid	Integer, 11 ints, Primary Key
name	Varchar(1-255)
hdprice	Numeric (10,2)

Candidate Keys: pid

Entity Integrity/Primary Key Constraint: “pid” is primary key and cannot be NULL

Uniqueness Constraint: “pid” is unique.

Referential Constraint: None

Business Constraint: Name and hdprice cannot be NULL.

Derivation: Product

Description: Derives from Product entity. Product ‘includes’ categories and supplier ‘suppliers’ products.

Relation Schema: Status

Status(<u>stid</u>, status)

Attributes	Domain
stid	Integers, 11 ints, Primary Key
status	Varchar(1-40)

Candidate Keys: stid

Entity Integrity/Primary Key Constraint: “stid” is primary key and cannot be NULL

Uniqueness Constraint: “stid” is unique.

Referential Constraint: None

Business Constraint: None

Derivation: Status

Description: Derives from Status entity.

Relation Schema: Supplier

Supplier(sid, name, phone_no, street, state, city, zipcode)

Attributes	Domain
sid	0-MaxInt
name	Varchar(40)
phone_no	Numeric(11), 0000000000-9999999999
street	Varchar(100)
state	Varchar(3)
city	Varchar(40)
zipcode	Integer, 5 ints

Candidate Keys: sid

Entity Integrity/Primary Key Constraint: “sid” is primary key and cannot be NULL

Uniqueness Constraint: “sid” is unique.

Referential Constraint: “sid” is a foreign key in the Product entity.

Business Constraint: None

Derivation: Supplier

Description: Derives from the Supplier Entity.

Relation Schema: Contains

Contains(pid, oid, qty, sprice)

Attributes	Domain
pid*	Integer, 11 ints
oid*	Integer, 11 ints
qty	Integer, 11 ints
sprice	Numeric

Candidate Keys: pid, oid

Entity Integrity/Primary Key Constraint: “pid” and “oid” cannot be NULL

Uniqueness Constraint: “oid” and “pid” must be unique

Referential Constraint: “pid” and “oid” must refer from Product and Orders primary keys.

Business Constraint: None

Derivation: Contains

Description: Derives from the Contains entity relationship.

Relation Schema: Has

Has(stid, soid, datetime)

Attributes	Domain
stid*	Integer, 11 ints
soid*	Integer, 11 ints
datetime	Date: yyyy-mm-dd, Time: HH:MM:SS

Candidate Keys: soid, stid

Entity Integrity/Primary Key Constraint: "soid" and "stid" cannot be NULL

Uniqueness Constraint: "soid" and "stid" must be unique

Referential Constraint: "soid" and "stid" must refer from Orders and Status primary keys.

Business Constraint: datetime cannot be NULL.

Derivation: Has

Description: Derives from the Has entity relationship.

Relation Schema: Includes

Includes(catid, pid)

Attributes	Domain
catid*	Integer, 11 ints
pid*	Integer, 11 ints

Candidate Keys: catid, pid

Entity Integrity/Primary Key Constraint: “catid” and “pid” cannot be NULL

Uniqueness Constraint: “catid” and “pid” must be unique

Referential Constraint: “catid” and “pid” must refer from Category and Products primary keys.

Business Constraint: None

Derivation: Includes

Description: Derives from the Includes entity relationship.

Relation Schema: Prepares

Prepares(eid, oid, datetime)

Attributes	**Domain**
eid*	Integer, 11 ints
oid*	Integer, 11 ints
datetime	Date: yyyy-mm-dd, Time: HH:MM:SS

Candidate Keys: eid, oid

Entity Integrity/Primary Key Constraint: "oid" and "eid" cannot be NULL

Uniqueness Constraint: "oid" and "eid" must be unique

Referential Constraint: "oid" and "eid" must refer from Orders and Employee primary keys.

Business Constraint: datetime cannot be NULL.

Derivation: Prepares

Description: Derives from the Prepares entity relationship.

Relation Schema: Supplies

Supplies(sid, pid, datetime)

Attributes	**Domain**
sid*	Integer, 11 ints
pid*	Integer, 11 ints
datetime	Date: yyyy-mm-dd, Time: HH:MM:SS

Candidate Keys: sid, pid

Entity Integrity/Primary Key Constraint: “sid” and “pid” cannot be NULL

Uniqueness Constraint: “sid” and “pid” must be unique

Referential Constraint: “sid” and “pid” must refer from Orders and Employee primary keys.

Business Constraint: datetime cannot be NULL.

Derivation: Supplies

Description: Derives from the Supplies entity relationship.

2.3.2. Sample Data of Relation

The following displays sample data of relations designed to show the tuples and the table body coinciding to each relation. The sample data objective is to demonstrate a case scenario in Kiwi's Coffee Shop. Furthermore, the relations corresponding to entity sets displays a minimum of ten tuples. The relations corresponding to relationships displays sixty to one hundred tuples. The sample data will additionally be displayed in Postgres.

Table 1: Category

CatID	Name
1	Hot Coffee
2	Cold Coffee
3	Hot Tea
4	Cold Tea
5	Breakfast
6	Fruits
7	Baked Goods
8	Snacks
9	Frappuccino
10	Bottled Drinks

Table 2: Customer

ID	Fname	Mname	Lname	Gender	Email	Street	City	State	ZipCode	Phone_no
1	Samori	James	Price	Male	sprice@gmail.com	2009 Marine Ave	Gardena	CA	90249	2132082705
2	Michelle	Ashley	Jackson	Female	mjackson@gmail.com	2099 W 41 St Dr	Los Angeles	CA	90062	3234188114
3	Zack	Henry	Sweyd	Male	zsweyd@gmail.com	3485 W 54th St	Los Angeles	CA	90043	3239368156
4	Julie	Kim	Dihn	Female	jdihn@gmail.com	141 S Pepper St	Orange	CA	92868	7142796812
5	Delilah	Deborah	Mitchell	Female	dmitchell@gmail.com	1608 W 84th Pl	Los Angeles	CA	90047	3238922984
6	Stephanie	Martha	Osbourne	Female	sosbourne@gmail.com	230 N Finch St	Anaheim	CA	92807	6575492957
7	Sean	John	Desmond	Male	sdesmond@gmail.com	20197 Hinsdale Ave	Torrance	CA	90503	3103200807
8	Richard	Edward	Santillano	Male	rsantillano@gmail.com	1201 Kendrick CT	Corona	CA	92881	9512649908
9	Kitty	Shae	Vo	Female	kvo@gmail.com	21233 Main St.	Garden Grove	CA	92840	7144178090
10	Kyle	Trevor	MacLean	Male	kmaclean@gmail.com	905 3rd St	Hermosa Beach	CA	90254	3103722765
11	Miller	Alex	Ramirez	Male	mramirez@gmail.com	1111 Success Ave	Los Angeles	CA	90059	3235170342
12	Anthony	Nick	Delgado	Male	adelgado@gmail.com	2223 St. Anne Pl.	Santa Ana	CA	92704	7144801805
13	Xavier	Gus	Robles	Male	xrobles@gmail.com	2345 Spurgeon Ave.	Santa Ana	CA	92703	7149535956
14	Hienny	Shirly	Tran	Female	htran@gmail.com	3242 Bristol Street	Santa Ana	CA	92704	7148007852
15	Ellen	Alana	Ijebor	Female	eijebor@gmail.com	9051 StockDale HWY #233	Bakersfield	CA	93311	6616645222
16	Hue-An	Lily	Tran	Female	huetran@gmail.com	9051 StockDale HWY #241	Bakersfield	CA	93311	6616612426
17	Kelly	Dahlia	Moua	Female	kmoua@gmail.com	9051 StockDale HWY #264	Bakersfield	CA	93311	6616616565
18	Donovan	Noah	Innes	Male	dinnes@gmail.com	2223 New Hope Ave	Garden Grove	CA	92705	7144856565
19	Molly	Khloe	Gorman	Female	mgorman@gmail.com	122 Main St.	Orange	CA	92703	7146005202
20	Alejandra	Gisele	Flores	Female	aflores@gmail.com	5239 McFadden Ave.	Santa Ana	CA	92704	7144866161

Table 3: Employee

EID	Fname	Mname	Lname	Role	Email
1	Boone	Maurizio	Wrightham	Barista	mwrightham0@prlog.org
2	Hamlen	Ronnie	Satterly	Cashier	rsatterly1@homestead.com
3	Haleigh	Tallia	McLoughlin	Barista	tmcloughlin2@scientificamerican.com
4	Myrilla	Elise	Stonner	Barista	estonner3@gizmodo.com
5	George	Mickey	Zeal	Cashier	mzeal4@samsung.com
6	Henrie	Haskell	Grumley	Barista	hgrumley5@cocolog-nifty.com
7	Umeko	Blondy	Minton	Barista	bminton6@liveinternet.ru
8	Quintana	Paulita	Bauman	Cashier	pbauman7@tripod.com
9	Collen	Windham	Roderham	Cashier	wroderham8@adobe.com
10	Issi	Therese	Tott	Barista	ttott9@bluehost.com
11	Stephi	Talia	Penman	Cashier	tpenmana@princeton.edu
12	Amandi	Madelina	Frackiewicz	Barista	mfrackiewiczb@devhub.com
13	Cathleen	Earlie	Vigors	Cashier	evigorsc@behance.net
14	Dell	Chelsy	Upjohn	Barista	cupjohnd@cbc.ca
15	Boigie	Donia	Markos	Manager	dmarkose@zdnet.com
16	Rubi	Francois	Farncombe	Barista	ffarncombef@weibo.com
17	Chip	Rosalinde	Linger	Barista	rlingerg@livejournal.com
18	Sidney	Perkin	Tawse	Cashier	ptawseh@webmd.com

19	King	Cassy	Chiddy	Barista	cchiddyi@thetimes.co.uk
20	Malissa	Olenka	Dumsday	Cashier	odumsdayj@ameblo.jp
21	Coleman	Linnet	Perrin	Cashier	lperrink@jugem.jp
22	Blair	Melvin	Prowse	Barista	mprowsel@phpbb.com
23	Imogene	Aleksandr	Fairhead	Cashier	afairheadm@bigcartel.com
24	Valerye	Coralie	Hardacre	Barista	chardacren@craigslist.org
25	Sinclare	Devin	Koeppe	Cashier	dkoeppeo@nasa.gov
26	Rowland	Gavrielle	McClenan	Barista	gmcclenanp@facebook.com
27	Brietta	Zacharias	Stanfield	Barista	zstanfieldq@nsw.gov.au
28	Giavani	Rafferty	Tondeur	Barista	rtondeurr@bizjournals.com
29	Leanora	Rora	Daniells	Cashier	rdaniellss@posterous.com
30	Elston	Concordia	Ethelston	Barista	cethelstont@mozilla.org

Table 4: Orders

id	payment	datetime	cid	total
1	card	9/27/2018 23:00	4	8
2	cash	9/28/2018 1:00	1	2
3	card	9/28/2018 1:02	1	7.5
4	card	9/28/2018 1:02	15	4
5	card	9/28/2018 1:06	14	7.5
6	card	9/28/2018 1:10	7	8.5
7	cash	9/28/2018 1:12	8	9.25
8	card	9/28/2018 1:15	7	9
9	card	9/28/2018 1:19	5	8.75
10	card	9/28/2018 1:20	6	1.5
11	card	9/28/2018 1:21	15	10
12	card	9/28/2018 1:25	18	5.75
13	card	9/28/2018 1:29	19	8.5

14	card	9/28/2018 1:30	9	7
15	card	9/28/2018 1:31	10	9
16	cash	9/28/2018 1:32	2	5
17	card	9/28/2018 1:35	4	9.75
18	card	9/28/2018 1:37	13	15.5
19	card	9/28/2018 1:39	16	8.2
20	card	9/28/2018 1:40	17	12.25
21	cash	9/28/2018 1:41	4	3.5
22	card	9/28/2018 1:45	9	5
23	card	9/28/2018 1:49	11	6.2
24	card	9/28/2018 1:51	12	6.2
25	card	9/28/2018 1:55	1	11.5
26	cash	9/28/2018 1:59	19	15.7
27	card	9/28/2018 2:02	20	12.25

28	cash	9/28/2018 2:05	1	7
29	card	9/28/2018 2:09	14	10.75
30	cash	9/28/2018 2:02	1	7.5
31	card	9/28/2018 3:02	15	6.5
32	cash	9/28/2018 3:06	14	5.2
33	card	9/28/2018 3:10	7	7.5
34	cash	9/28/2018 3:12	8	8
35	card	9/28/2018 3:15	7	9.25
36	card	9/28/2018 3:19	5	6.5
37	card	9/28/2018 4:20	6	7.5
38	card	9/28/2018 4:21	15	6.2
39	card	9/28/2018 4:25	18	4
40	card	9/28/2018 4:29	19	3.5
41	card	9/29/2018 1:01	9	10

42	card	9/29/2018 1:05	10	12.25
43	cash	9/29/2018 1:09	2	6.25
44	card	9/29/2018 1:20	4	3.5
45	card	9/29/2018 1:21	13	8
46	card	9/29/2018 1:25	16	3.5
47	card	9/29/2018 1:28	17	9.25
48	cash	9/29/2018 1:29	4	7.5
49	card	9/29/2018 1:35	9	3.75

Table 5: Products

PID	Name	HDP
1	Black Coffee	2.00
2	White Vanilla Coffee	3.50
3	Cafe Latte	2.50
4	Cafe Mocha	3.50
5	Macchiato	3.50
6	Cafe Au Late	4.00
7	Espresso	3.50
8	Cafe Cubano	3.00
9	Hazelnut Coffee	3.50
10	Green Tea	1.75
11	Earl Grey Tea	2.75
12	Jasmine Tea	1.00
13	Oolong Tea	2.25
14	Thai Milk Tea	2.25
15	Matcha Green Tea	4.50
16	Almond Milk Tea	2.50
17	Cafe Americano	3.50
18	Cafe Latte	2.50

19	White Tea	1.75
20	Coconut Milk Tea	2.00
21	Panini	6.50
22	Banana	0.50
23	Passion Iced Tea	2.50
24	Cinnamon Orange Iced Tea	2.70
25	Egg and Cheese Sandwich	4.50
26	Apricot Porridge	3.50
27	Boiled Eggs	0.50
28	Apple	0.60
29	Butter Croissant	1.25
30	Banana Muffin	1.50
31	Chocolate Muffin	1.50

Table 6: Status

id	status
1	New Order
2	Order Pending
3	Processing
4	Preparing Order
5	Almost Ready
6	Ready to Go
7	Done

Table 7: Supplier

SID	Name	Street	State	City	ZipCode	PhoneNo
1	Elevated Coffee and Vending	675 W High St	OH	Orrville	44667	3307656289
2	Sunshine Coffee	1301 Harrison St	CA	San Francisco	94103	4158665397
3	Snap Coffee	208 Broadway E	WA	Seattle	98102	2067371088
4	Pulse Coffee Vending	901 Harrison St	CA	Oakland	94607	4154803574
5	Sunbelt Coffee Supplier	31 W Puainako St	HI	Hilo	96720	8089879975
6	Fortune Coffee Vending	3050 Barranca Pkwy	CA	Irvine	92606	9494352050
7	BigBlue Coffee Enterprises	480 Furnace Brook Pkwy	MA	Quincy	2169	6176554222
8	GreenStar Coffee	3043 W 11th Ave	OR	Eugene	97402	5416669892
9	River Valley Coffee Vending	222 Vine St	PA	Philadelphia	19107	4843841116
10	Ventura Coffee Supplier	233 E Campus Way	CA	Ventura	93003	8058505532

Table 8: Contains

pid	oid	qty	sprice
3	1	1	3
25	1	1	4.5
22	1	1	0.5
1	2	1	4
4	3	1	3.5
9	3	1	3.5
28	3	1	0.5
28	4	1	0.5
9	4	1	3.5
4	5	1	3.5
20	5	1	2
27	5	1	0.5
31	5	1	1.5
21	6	1	6.5
1	6	1	2
15	7	1	4.5
26	7	1	3.5
15	7	1	1.25

31	8	1	1.5
26	8	1	3.5
9	8	1	3.5
28	8	1	0.5
13	9	1	2.25
7	9	1	3.5
30	9	1	1.5
31	9	1	1.5
30	10	1	1.5
2	11	1	3.5
3	11	1	2.5
28	11	1	0.5
13	12	1	2.25
4	11	1	3.5
4	12	1	3.5
6	13	1	4
8	13	1	3
30	13	1	1.5
7	14	1	3.5
2	14	1	3.5

2	15	1	3.5
4	15	1	3.5
28	15	1	0.5
30	15	1	1.5
31	16	1	1.5
5	16	1	3.5
6	17	1	4
13	17	1	2.25
26	17	1	3.5
26	18	1	3.5
7	18	1	3.5
8	18	1	3
6	18	1	4
30	18	1	1.5
4	19	1	3.5
1	19	1	2
24	19	1	2.7
13	20	1	2.25
21	20	1	6.5
5	20	1	3.5

26	21	1	3.5
23	22	1	2.5
22	22	1	0.5
1	22	1	2
4	23	1	3.5
24	23	1	2.7
24	24	1	2.7
7	24	1	3.5
21	25	1	6.5
5	25	1	3.5
30	25	1	1.5
8	26	1	3
4	26	1	3.5
21	26	1	6.5
26	27	1	3.5
24	26	1	2.7
13	27	1	2.25
21	27	1	6.5
5	28	1	3.5
30	28	1	1.5

1	28	1	2
1	29	1	2
13	29	1	2.25
21	29	1	6.5
26	30	1	3.5
6	30	1	4
21	31	1	6.5
24	32	1	2.7
3	32	1	2.5
17	33	1	3.5
6	33	1	4
30	34	1	1.5
21	34	1	6.5
4	35	1	3.5
5	35	1	3.5
13	35	1	2.25
21	36	1	6.5
6	37	1	4
5	37	1	3.5
4	38	1	3.5

24	38	1	2.7
16	39	1	2.5
30	39	1	1.5
4	40	1	3.5
21	41	1	6.5
5	41	1	3.5
21	42	1	6.5
4	42	1	3.5
13	42	1	2.25
13	43	1	2.25
6	43	1	4
17	44	1	3.5
21	45	1	6.5
30	45	1	1.5
5	46	1	3.5
13	47	1	2.25
4	47	1	3.5
5	47	1	3.5
4	48	1	3.5
6	48	1	4

30	48	1	1.5
13	49	1	2.25
30	49	1	1.5

Table 9: Has

soid	stid	datetime
2	1	9/28/2018 1:00
1	1	9/28/2018 1:00
3	1	9/28/2018 1:02
4	1	9/28/2018 1:02
2	2	9/28/2018 1:06
3	2	9/28/2018 1:10
4	2	9/28/2018 1:12
5	1	9/28/2018 1:00
6	1	9/28/2018 1:02
4	3	9/28/2018 1:02
3	3	9/28/2018 1:06
2	3	9/28/2018 1:10
1	2	9/28/2018 1:12

2	4	9/28/2018 1:00
2	5	9/28/2018 1:02
3	3	9/28/2018 1:02
1	3	9/28/2018 1:06
1	4	9/28/2018 1:10
1	5	9/28/2018 1:12
2	6	9/28/2018 1:00
2	7	9/28/2018 1:02
3	4	9/28/2018 1:02
4	4	9/28/2018 1:06
4	5	9/28/2018 1:10
5	2	9/28/2018 1:12
5	3	9/28/2018 1:00
6	2	9/28/2018 1:02

1	7	9/28/2018 1:02
3	5	9/28/2018 1:06
3	6	9/28/2018 1:10
4	6	9/28/2018 1:12
4	7	9/28/2018 1:00
5	4	9/28/2018 1:02
5	5	9/28/2018 1:02
6	3	9/28/2018 1:06
5	6	9/28/2018 1:10
6	4	9/28/2018 1:12
5	7	9/28/2018 1:00
6	5	9/28/2018 1:02
6	6	9/28/2018 1:02
6	7	9/28/2018 1:06

9	1	9/28/2018 1:12
8	1	9/28/2018 1:15
10	1	9/28/2018 1:19
11	1	9/28/2018 1:20
9	2	9/28/2018 1:21
10	2	9/28/2018 1:25
11	2	9/28/2018 1:12
12	1	9/28/2018 1:15
13	1	9/28/2018 1:19
11	3	9/28/2018 1:20
10	3	9/28/2018 1:21
9	3	9/28/2018 1:25
8	2	9/28/2018 1:12
9	4	9/28/2018 1:15

9	5	9/28/2018 1:19
10	3	9/28/2018 1:20
8	3	9/28/2018 1:21
8	4	9/28/2018 1:25
8	5	9/28/2018 1:12
9	6	9/28/2018 1:15
9	7	9/28/2018 1:19
10	4	9/28/2018 1:20
11	4	9/28/2018 1:21
11	5	9/28/2018 1:25
12	2	9/28/2018 1:12
12	3	9/28/2018 1:15
13	2	9/28/2018 1:19
8	7	9/28/2018 1:20

10	5	9/28/2018 1:21
10	6	9/28/2018 1:25
11	6	9/28/2018 1:12
11	7	9/28/2018 1:15
12	4	9/28/2018 1:19
12	5	9/28/2018 1:20
13	3	9/28/2018 1:21
12	6	9/28/2018 1:25
13	4	9/28/2018 1:12
12	7	9/28/2018 1:15
13	5	9/28/2018 1:19
13	6	9/28/2018 1:20
13	7	9/28/2018 1:21
7	1	9/28/2018 1:25

7	1	9/28/2018 1:12
15	1	9/28/2018 1:29
17	2	9/28/2018 1:30
18	2	9/28/2018 1:31
16	2	9/28/2018 1:32
17	1	9/28/2018 1:35
18	1	9/28/2018 1:37
19	3	9/28/2018 1:29
20	3	9/28/2018 1:30
18	3	9/28/2018 1:31
17	2	9/28/2018 1:32
16	4	9/28/2018 1:35
15	5	9/28/2018 1:37
16	3	9/28/2018 1:29

16	3	9/28/2018 1:30
17	4	9/28/2018 1:31
15	5	9/28/2018 1:32
15	6	9/28/2018 1:35
15	7	9/28/2018 1:37
16	4	9/28/2018 1:29
16	4	9/28/2018 1:30
17	5	9/28/2018 1:31
18	2	9/28/2018 1:32
18	3	9/28/2018 1:35
19	2	9/28/2018 1:37
19	7	9/28/2018 1:29
20	5	9/28/2018 1:30
15	6	9/28/2018 1:31

17	6	9/28/2018 1:32
17	7	9/28/2018 1:35
18	4	9/28/2018 1:37
18	5	9/28/2018 1:29
19	3	9/28/2018 1:30
19	6	9/28/2018 1:31
20	4	9/28/2018 1:32
19	7	9/28/2018 1:35
20	5	9/28/2018 1:37
19	6	9/28/2018 1:29
20	7	9/28/2018 1:30
20	1	9/28/2018 1:31
20	1	9/28/2018 1:32
14	1	9/28/2018 1:35

14	1	9/28/2018 1:37
23	2	9/28/2018 1:39
22	2	9/28/2018 1:40
24	2	9/28/2018 1:41
25	1	9/28/2018 1:45
23	1	9/28/2018 1:49
24	3	9/28/2018 1:51
25	3	9/28/2018 1:39
26	3	9/28/2018 1:40
27	2	9/28/2018 1:41
25	4	9/28/2018 1:45
24	5	9/28/2018 1:49
23	3	9/28/2018 1:51
22	3	9/28/2018 1:39

23	4	9/28/2018 1:40
23	5	9/28/2018 1:41
24	6	9/28/2018 1:45
22	7	9/28/2018 1:49
22	4	9/28/2018 1:51
22	4	9/28/2018 1:39
23	5	9/28/2018 1:40
23	2	9/28/2018 1:41
24	3	9/28/2018 1:45
25	2	9/28/2018 1:49
25	7	9/28/2018 1:51
26	5	9/28/2018 1:39
26	6	9/28/2018 1:40
27	6	9/28/2018 1:41

22	7	9/28/2018 1:45
24	4	9/28/2018 1:49
24	5	9/28/2018 1:51
25	3	9/28/2018 1:39
25	6	9/28/2018 1:40
26	4	9/28/2018 1:41
26	7	9/28/2018 1:45
27	5	9/28/2018 1:49
26	6	9/28/2018 1:51
27	7	9/28/2018 1:39
26	1	9/28/2018 1:40
27	1	9/28/2018 1:41
27	1	9/28/2018 1:45
27	1	9/28/2018 1:49

21	2	9/28/2018 1:51
21	2	9/28/2018 1:39
30	2	9/28/2018 1:55
29	1	9/28/2018 1:59
31	1	9/28/2018 2:02
32	3	9/28/2018 2:05
30	3	9/28/2018 2:09
31	3	9/28/2018 2:02
32	2	9/28/2018 1:55
33	4	9/28/2018 1:59
34	5	9/28/2018 2:02
32	3	9/28/2018 2:05
31	3	9/28/2018 2:09
30	4	9/28/2018 2:02

29	5	9/28/2018 1:55
30	6	9/28/2018 1:59
30	7	9/28/2018 2:02
31	4	9/28/2018 2:05
29	4	9/28/2018 2:09
29	5	9/28/2018 2:02
29	2	9/28/2018 1:55
30	3	9/28/2018 1:59
30	2	9/28/2018 2:02
31	7	9/28/2018 2:05
32	5	9/28/2018 2:09
32	6	9/28/2018 2:02
33	6	9/28/2018 1:55
33	7	9/28/2018 1:59

34	4	9/28/2018 2:02
29	5	9/28/2018 2:05
31	3	9/28/2018 2:09
31	6	9/28/2018 2:02
32	4	9/28/2018 1:55
32	7	9/28/2018 1:59
33	5	9/28/2018 2:02
33	6	9/28/2018 2:05
34	7	9/28/2018 2:09
33	1	9/28/2018 2:02
34	1	9/28/2018 1:55
33	1	9/28/2018 1:59
34	1	9/28/2018 2:02
34	2	9/28/2018 2:05

34	2	9/28/2018 2:09
28	2	9/28/2018 2:02
28	1	9/28/2018 1:55
37	1	9/28/2018 3:02
36	3	9/28/2018 3:06
38	3	9/28/2018 3:10
39	3	9/28/2018 3:12
37	2	9/28/2018 3:15
38	4	9/28/2018 3:19
39	5	9/28/2018 3:02
40	3	9/28/2018 3:06
41	3	9/28/2018 3:10
39	4	9/28/2018 3:12
38	5	9/28/2018 3:15

37	6	9/28/2018 3:19
36	7	9/28/2018 3:02
37	4	9/28/2018 3:06
37	4	9/28/2018 3:10
38	5	9/28/2018 3:12
36	2	9/28/2018 3:15
36	3	9/28/2018 3:19
36	2	9/28/2018 3:02
37	7	9/28/2018 3:06
37	5	9/28/2018 3:10
38	6	9/28/2018 3:12
39	6	9/28/2018 3:15
39	7	9/28/2018 3:19
40	4	9/28/2018 3:02

40	5	9/28/2018 3:06
41	3	9/28/2018 3:10
36	6	9/28/2018 3:12
38	4	9/28/2018 3:15
38	7	9/28/2018 3:19
39	5	9/28/2018 3:02
39	6	9/28/2018 3:06
40	7	9/28/2018 3:10
40	1	9/28/2018 3:12
41	1	9/28/2018 3:15
40	1	9/28/2018 3:19
41	1	9/28/2018 3:02
40	2	9/28/2018 3:06
41	2	9/28/2018 3:10

41	2	9/28/2018 3:12
41	1	9/28/2018 3:15
35	1	9/28/2018 3:19
35	3	9/28/2018 3:02
44	3	9/28/2018 4:20
43	3	9/28/2018 4:21
45	2	9/28/2018 4:25
46	4	9/28/2018 4:29
44	5	9/28/2018 4:32
45	3	9/28/2018 4:20
46	3	9/28/2018 4:21
47	4	9/28/2018 4:25
48	5	9/28/2018 4:29
46	6	9/28/2018 4:32

45	7	9/28/2018 4:20
44	4	9/28/2018 4:21
43	4	9/28/2018 4:25
44	5	9/28/2018 4:29
44	2	9/28/2018 4:32
45	3	9/28/2018 4:20
43	2	9/28/2018 4:21
43	7	9/28/2018 4:25
43	5	9/28/2018 4:29
44	6	9/28/2018 4:32
44	6	9/28/2018 4:20
45	7	9/28/2018 4:21
46	4	9/28/2018 4:25
46	5	9/28/2018 4:29

47	3	9/28/2018 4:32
47	6	9/28/2018 4:20
48	4	9/28/2018 4:21
43	7	9/28/2018 4:25
45	5	9/28/2018 4:29
45	6	9/28/2018 4:32
46	7	9/28/2018 4:20
46	1	9/28/2018 4:21
47	1	9/28/2018 4:25
47	1	9/28/2018 4:29
48	1	9/28/2018 4:32
47	2	9/28/2018 4:20
48	2	9/28/2018 4:21
47	2	9/28/2018 4:25

48	1	9/28/2018 4:29
48	1	9/28/2018 4:32
48	3	9/28/2018 4:20
42	3	9/28/2018 4:21
42	3	9/28/2018 4:25
49	6	9/29/2018 2:00
49	7	9/29/2018 2:05

Table 10: Includes

catid	pid
1	1
1	2
1	3
1	4
1	5
1	6
1	7
1	9
2	1
2	2
2	3
2	4
2	5
2	6
2	7
2	8
2	9
3	10

3	11
3	12
3	13
4	14
4	15
4	16
1	17
2	17
1	18
2	18
3	19
4	20
5	21
6	22
4	23
4	24
5	25
5	26
8	27
8	28

7	29
7	31
7	30

Table 11: Products

eid	oid	datetime
6	1	9/27/2018 23:03
3	2	9/28/2018 1:03
7	4	9/28/2018 1:05
22	3	9/28/2018 1:07
27	5	9/28/2018 1:11
30	7	9/28/2018 1:13
27	6	9/28/2018 1:16
28	9	9/28/2018 1:20
16	8	9/28/2018 1:21
6	12	9/28/2018 1:26
3	10	9/28/2018 1:27
7	11	9/28/2018 1:28
22	13	9/28/2018 1:28

27	14	9/28/2018 1:31
30	16	9/28/2018 1:34
27	15	9/28/2018 1:35
28	17	9/28/2018 1:38
16	18	9/28/2018 1:38
6	20	9/28/2018 1:41
3	19	9/28/2018 1:42
7	21	9/28/2018 1:43
22	23	9/28/2018 1:46
27	22	9/28/2018 1:50
30	24	9/28/2018 1:52
27	25	9/28/2018 1:56
28	27	9/28/2018 2:00
16	26	9/28/2018 2:05

3	29	9/28/2018 2:06
7	28	9/28/2018 2:10
22	30	9/28/2018 2:11
27	31	9/28/2018 3:07
30	33	9/28/2018 3:11
27	32	9/28/2018 3:13
28	34	9/28/2018 3:16
16	36	9/28/2018 3:20
3	35	9/28/2018 3:21
7	37	9/28/2018 4:22
22	38	9/28/2018 4:28
27	39	9/28/2018 4:31
30	40	9/28/2018 4:34
27	42	9/29/2018 1:06

28	41	9/29/2018 1:07
16	43	9/29/2018 1:10
27	44	9/29/2018 1:21
30	46	9/29/2018 1:28
27	45	9/29/2018 1:30
28	47	9/29/2018 1:32
16	48	9/29/2018 1:33
3	49	9/29/2018 1:36

Table 12: Supplies

sid	pid	qty	datetime
1	2	2	09/28/2018 1:00:00
1	3	1	09/29/2018 2:03:00
1	4	2	09/29/2018 3:10:00
2	2	2	09/29/2018 2:12:00
2	1	2	10/01/2018 10:02:00
10	2	2	10/01/2018 12:33:00
9	6	2	10/02/2018 2:03:00
3	4	2	10/02/2018 8:40:00
6	5	1	10/03/2018 10:03:00
7	5	2	10/03/2018 4:20:00
5	2	4	10/03/2018 10:12:00
4	6	3	10/04/2018 12:02:00
10	3	2	10/04/2018 12:33:00

2	6	2	10/05/2018 2:03:00
4	4	2	10/05/2018 8:40:00
7	5	2	10/06/2018 4:20:00
5	2	4	10/06/2018 10:12:00
4	6	3	10/06/2018 12:02:00
10	3	2	10/07/2018 12:33:00
2	6	2	10/07/2018 2:03:00
4	4	2	10/07/2018 8:40:00
7	5	2	10/08/2018 4:20:00
5	2	4	10/09/2018 10:12:00
4	6	3	10/09/2018 12:02:00
10	3	4	10/10/2018 12:33:00
2	6	4	10/11/2018 2:03:00
1	4	3	10/11/2018 8:40:00

2	4	2	10/12/2018 8:40:00
6	5	2	10/12/2018 4:20:00
5	2	4	10/13/2018 10:12:00
4	6	3	10/14/2018 12:02:00
7	3	4	10/15/2018 12:33:00
3	6	4	10/16/2018 2:03:00
2	4	3	10/16/2018 8:40:00
6	20	1	10/17/2018 10:03:00
7	12	2	10/17/2018 4:20:00
5	10	4	10/18/2018 10:12:00
4	11	3	10/19/2018 12:02:00
10	9	2	10/19/2018 12:33:00
2	7	2	10/20/2018 2:03:00
4	19	2	10/20/2018 8:40:00

7	20	2	10/21/2018 4:20:00
5	1	4	10/22/2018 10:12:00
4	2	3	10/22/2018 12:02:00
10	4	2	10/23/2018 12:33:00
2	5	2	10/23/2018 2:03:00
4	15	2	10/23/2018 8:40:00
7	16	2	10/24/2018 4:20:00
5	12	4	10/25/2018 10:12:00
4	6	3	10/25/2018 12:02:00
10	14	4	10/26/2018 12:33:00
2	13	4	10/26/2018 2:03:00
1	14	3	10/26/2018 8:40:00
2	3	2	10/27/2018 8:40:00
6	15	2	10/27/2018 4:20:00

5	17	4	10/28/2018 10:12:00
4	6	3	10/29/2018 12:02:00
7	13	4	10/30/2018 12:33:00
3	16	4	10/31/2018 2:03:00
2	4	3	10/31/2018 8:40:00
6	20	1	11/01/2018 10:03:00
7	12	2	11/02/2018 4:20:00
5	10	4	11/02/2018 10:12:00
4	11	3	11/02/2018 12:02:00
10	9	2	11/03/2018 12:33:00
2	7	2	11/04/2018 2:03:00
4	19	2	11/04/2018 8:40:00
7	20	2	11/05/2018 4:20:00
5	1	4	11/05/2018 10:12:00

4	2	3	11/05/2018 12:02:00
10	4	2	11/06/2018 12:33:00
2	5	2	11/07/2018 2:03:00
4	15	2	11/07/2018 8:40:00
7	16	2	11/08/2018 4:20:00
5	12	4	11/09/2018 10:12:00
4	6	3	11/09/2018 12:02:00
10	14	4	11/10/2018 12:33:00
2	13	4	11/11/2018 2:03:00
1	14	3	11/11/2018 8:40:00
2	3	2	11/12/2018 8:40:00
6	15	2	11/13/2018 4:20:00
5	17	4	11/13/2018 10:12:00
4	6	3	11/13/2018 12:02:00

7	13	4	11/14/2018 12:33:00
3	16	4	11/15/2018 2:03:00
2	4	3	11/15/2018 8:40:00
6	20	1	11/16/2018 10:03:00
7	12	2	11/17/2018 4:20:00
5	10	4	11/17/2018 10:12:00
4	11	3	11/17/2018 12:02:00
10	9	2	11/18/2018 12:33:00
2	7	2	11/19/2018 2:03:00
4	19	2	11/19/2018 8:40:00
7	20	2	11/20/2018 4:20:00
5	1	4	11/21/2018 10:12:00
4	2	3	11/21/2018 12:02:00
10	4	2	11/22/2018 12:33:00

2	5	2	11/23/2018 2:03:00
4	15	2	11/23/2018 8:40:00
7	16	2	11/24/2018 4:20:00
5	12	4	11/25/2018 10:12:00
4	6	3	11/25/2018 12:02:00
10	14	4	11/26/2018 12:33:00
2	13	4	11/27/2018 2:03:00
1	14	3	11/27/2018 8:40:00
2	3	2	11/28/2018 8:40:00
6	15	2	11/29/2018 4:20:00
5	17	4	11/30/2018 10:12:00
4	6	3	11/30/2018 12:02:00
7	13	4	12/01/2018 12:33:00
3	16	4	12/02/2018 2:03:00
2	4	3	12/02/2018 8:40:00

2.4 Sample Queries to Your Database

The purpose of this section is to demonstrate sample queries from Kiwi's Coffee Shop database. The first subsection presents the design of the queries using words to express operations to allow the user to specify retrieval request of expressions. The queries consist ten questions providing additional operations such as division and difference, including qualifiers like for-all and not-exist for the purpose of adding complexity to the questions.

The second subsection contains the relational algebra expression providing a brief description of the expression, including presenting the query questions with the relational algebra expressions answers. The third subsection consists of tuple relational calculus expressions that provides a brief expression of the expression representing the queries in the tuple relational expressions. Similarly, the third subsection contains domain relational calculus expressions providing the description and representation of the queries with the domain relational expressions.

2.4.1 Design Of Queries

The following queries provides ten questions that are meant to be non-trivial consisting of division and difference operations to develop complex questions. The queries include few questions that provide for-all and non-existent qualifiers to the expressions.

2.4.2 Relational Algebra Expressions for Queries

Relational algebra (RA) represents relations through algebra whose operands are relations and operators are designed to perform the most common operations needed to do with relations in the database. Furthermore, relational algebra is considered to be a procedural query language that takes the operands of relations as input and returns instances of relations as an output. In addition, relational algebra can be performed recursively on a relation through the most common operators as 'select', 'project', 'union', 'set difference', and 'cartesian product'.

Relational Algebra Queries:

1.) List all orders containing product name 'Black Coffee' with the sale price less than 3.00.

$$Porder \leftarrow \pi_{id,payment}(\overset{o}{Orders})$$

$$Corder \leftarrow (Porder \underset{o.id\ = c.oid}{x} \overset{c}{Contains})$$

$$(\underset{p.id\ =\ c.pid\ \wedge\ p.name\ = 'Black\ Coffee'\ \wedge\ c.sprice\ <\ 3}{\sigma} (\overset{c}{Corder}\ x\ \overset{p}{Products}))$$

2.) List all the suppliers supplying less than 2 products.

$$\pi_{s.name,s2.datetime,s2.qty,p.name}(\underset{s.id\ =\ s2.sid\ \wedge\ p.id\ =\ s2.pid\ \wedge s2.qty\ <\ 2}{\sigma} (\overset{s}{Supplier}\ x\ \overset{s2}{Supplies}\ x\ \overset{p}{Products}))$$

3.) List the customer's first and last name with a complete status from the customer's placed order.

$$Cust \leftarrow \rho_{c}(cust\_fname, cust\_lname(\overset{c}{Customer}))$$

$$\leftarrow \pi_{c.fname,c.lname}(Customer)$$

$$\pi_{o.id,o.payment,o.datetime,o.total,s.status}(\underset{\substack{h.soid\ =\ o.id\ \wedge\ s.id\ =\ h.stid\ \wedge\ c.id = o.cid \\ h.datetime\ >\ '2018-09-28\ 00:00:00'\ \wedge \\ d.id \neq s.id\ \wedge\ d.id = h.stid\ \wedge \\ d.status\ =\ 'Done'}}{\sigma} (\overset{c}{Cust}\ x\ \overset{o}{Orders}\ x\ \overset{h}{Has}\ x\ \overset{s}{Status}\ x\ \overset{d}{Status}))$$

4.) List the orders placed by customers prepared by the employee 'Haleigh'.

$$Emp \leftarrow \rho_{e}(ename, cname(Employee)) \leftarrow \pi_{e.fname,c.fname,c.lname}(Employee^{e})$$

$$(\pi_{o.*}(\sigma_{\substack{p.oid = o.id \wedge o.cid = c.id \wedge \\ e.id = p.eid \wedge p.datetime >= '2018-09-28\ 00:00:00' \wedge \\ e2.id \neq e.id \wedge e2.fname = 'Haleigh'}} Orders^{o}\ x\ Prepares^{p}\ x\ Emp^{e}\ x\ Emp^{e2})$$

5.) List the customers who placed an order once.

$$Cust \leftarrow \rho_{c}(customer\_name, customer\_lname(Customer)) \leftarrow \pi_{c.id,c.fname,c.lname}(Customer^{c})$$

$$CustOrder \leftarrow \sigma_{o.cid = c.id}(Cust^{c}\ x\ Orders^{o})$$

$$(\sigma_{\substack{o2.id \neq o.id \wedge \\ o2.cid = c.id \wedge \\ o.datetime\ != o2.datetime}}(Cust^{c}\ x\ Orders^{o}\ x\ Orders^{o2}) - CustOrders)$$

6.) List the role with the least number of employees.

$$Emp \leftarrow \pi_{e.id,e.fname,e.lname,e.role}(Employee^{e})$$

$$(\sigma_{\substack{e.id \neq e2.id \wedge \\ e.role < e2.role}}(Employee^{e2}) - Emp^{e})$$

7.) List the last and first name of customers who have not placed an order.

$$Cust \leftarrow \pi_{c.lname,c.fname}\ (\overset{c}{Customer})$$

$$(\sigma_{c.id = o.cid}\ (\overset{o}{Orders}\ -\ \overset{c}{Cust}))$$

8.) List the first and last name of customers who at least more than 10.00 on their order.

$$Cust \leftarrow \pi_{c.fname,c.lname}\ (\overset{c}{Customer})$$

$$\sigma_{\substack{o.cid = c.id\ \wedge \\ o.total >= 10.00}}\ (\overset{o}{Orders}\ x\ \overset{c}{Cust})$$

9.) List the customer's first and last name who ordered products 'Cafe Americano' and 'Apple' or from the category 'Snacks'. cu

$$Cust \leftarrow \rho_{cu}(cust\_name, customer\_lname(Customer))$$
$$\leftarrow \pi_{cu.fname,cu.lname}\ (Customer)$$

$$Prod \leftarrow \rho_{p}(products\_name(Products)) \leftarrow \pi_{p.name}(\overset{p}{Products})$$

$$Cat \leftarrow \rho_{c}(category\_name(Category)) \leftarrow \pi_{c.name}(\overset{c}{Category})$$

$$\pi_{o.payment,o.datetime,o.total}(\sigma_{\substack{cu.id = o.cid\ \wedge \\ co.oid = o.id\ \wedge \\ p.id = co.pid\ \wedge \\ i.pid = p.id\ \wedge \\ c.oid = i.catid\ \wedge \\ p.name\ =\ 'Cafe\ Americano'\ \vee \\ c.name = 'Snacks'\ \wedge \\ p.name\ = 'Apple'\ \wedge \\ o.total\ <\ 8.00}}\ \overset{cu}{Cust}\ x\ \overset{co}{Contains}\ x\ \overset{o}{Orders}\ x\ \overset{p}{Prod}\ x\ \overset{i}{Includes}\ x\ \overset{c}{}$$

10.) List the suppliers who provided a less than one of products that cost more than 2.00 dollars.

$$Supp \leftarrow \rho_{s}(supplier\_name\ (Supplier)) \leftarrow \pi_{s.id,s.name} \overset{s}{(Supplier)}$$

$$Prod \leftarrow \rho_{p}(Product\_name\ (Products)) \leftarrow \pi_{p.name,p.hdp} \overset{p}{(Products)}$$

$$SuppLessThan \leftarrow \pi_{su.qty}(\sigma_{\substack{s.id = su.sid\ \wedge \\ su.pid = p.id\ \wedge \\ su.id\ \neq\ su2.id\ \wedge \\ p.id\ \neq p2.id}}\ \overset{s}{Supp}\ x\ \overset{p}{Prod}\ x\ \overset{su}{Supplies})$$

$$\pi_{.}^{\ *}\ (\sigma_{\substack{su.datetime\ != su.datetime\ \wedge \\ su2.qty < su..qty\ \wedge \\ p.hdp\ >\ 2.00\ \wedge \\ p2.hdp\ != \ p.hdp}}\ (\overset{su}{Supplies}\ x\ \overset{su2}{Supplies}\ x\ \overset{p}{Prod}\ x\ \overset{p2}{Prod}) - SuppLessThan)$$

2.4.3 Tuple Relational Calculus Expression for Queries

Tuple relational calculus (TRC) is a non-procedural query language and only provides the description of the query but does not contain the methods to solve it. Therefore, tuple relational calculus explains what to do, but not how to solve it. Tuple relational calculus selects the tuples with a range of values or tuples with certain attribute values. This results into a relation that can have one or multiple tuples.

1.) List all orders containing product name 'Black Coffee' with the sale price less than 3.00.

$$\begin{aligned}
&\{o.id, o.payment \mid Orders(o) \wedge (\exists c)(\exists p)\,(\\
&Contains(c) \wedge Products(p) \wedge (\\
&\quad p.id = c.pid \wedge \\
&\quad p.name = 'Black\ Coffee' \wedge \\
&\quad c.id = o.id \wedge \\
&\quad c.sprice < 3 \\
&))\}
\end{aligned}$$

2.) List all the suppliers supplying less than 2 products.

$$\begin{aligned}
&\{s.name \mid Supplier(s) \wedge (\exists s2)(\exists p)\,(\\
&Supplies(s2) \wedge Products(p) \wedge (\\
&\quad s.id = s2.sid \wedge \\
&\quad p.id = s2.pid \wedge \\
&\quad s2.qty < 2 \\
&))\}
\end{aligned}$$

3.) List the customer's first and last name with a complete status from the customer's placed order.

$$\begin{aligned}
&\{c.fname, c.lname \mid Customer(c) \wedge (\exists o)(\exists h)(\exists s)(\exists d)\,(\\
&Orders(o) \wedge Has(h) \wedge Status(s) \wedge Status(d) \wedge (\\
&\quad h.soid = o.id \wedge s.id = h.stid \wedge \\
&\quad c.id = o.cid \wedge h.datetime > '2018-09-28\ 00{:}00{:}00' \\
&\quad d.id \neq s.id \wedge d.id = h.stid \wedge \\
&\quad d.status = 'done'))\}
\end{aligned}$$

4.) List the orders placed by customers prepared by the employee 'Haleigh'.

$$\{e.fname, e.lname \mid Employee(e) \land (\exists o)\,(\exists p)(\exists e2) \land (\\
Orders(o) \land Prepares(p) \land Emp(e2) \land (\\
p.oid = o.id \land o.cid = c.id \land\\
e.id = p.eid \land datetime >= '2018-09-28\ 00{:}00{:}00' \land\\
e2.id \neq e.id \land e2.fname = 'Haleigh'\ ))\}$$

5.) List the customers who placed an order once.

$$\{c.id, c.fname, c.lname \mid Customer(c) \land (\exists o)\,(Orders(o) \land (\\
o.cid = c.id \land\\
(\neg(\exists o2)(Orders(o2) \land (\\
o2 \neq o \land\\
o2.cid = c.id \land\\
o.datetime\ != o2.datetime\ )))\}$$

6.) List the role with the least number of employees.

$$\{e.id, e.fname, e.lname, e.role \mid employee(e) \land \neg(\exists\, e2)(employee(e2) \land\\
e.id\ != e2.id \land\\
e.role < e2.role)\,\}$$

7.) List the last and first name of customers who have not placed an order.

$$\{c.lname, c.fname \mid Customer(c) \land \neg(\exists\, o)(Orders(o) \land\\
c.id = o.cid\,\}$$

8.) List the first and last name of customers who at least more than 10.00 on their order.

$$\{c.fname, c.lname \mid Customer(c) \land (\neg(\exists o)(Orders(o) \land (\\
o.cid = c.id \land o.total >= 10.00))\}$$

9.) List the customer's first and last name who ordered products 'Cafe Americano' and 'Apple' or from the category 'Snacks'.

$$\{cu.fname, cu.lname \mid Customer(cu) \land ((\forall p)(Products(p) \land (\\
(\exists co)(\exists o)(\exists i)(\exists c)(Contains(co) \land Orders(o) \land \\
(Includes(i) \land Category(c) \land (\\
cu.id = o.cid \land \\
co.oid = o.id \land \\
p.id = co.pid \land \\
i.pid = p.id \land \\
c.oid = i.catid \land \\
p.name = 'Cafe\ Americano' \lor \\
c.name = 'Snacks' \land \\
p.name = 'Apple' \land \\
o.total < 8.00))))\}$$

10.) List the suppliers who provided a less than one of products that cost more than 2.00 dollars.

$$\{s.id, s.name \mid Supplier(s) \land ((\exists p)(\exists su)(Products(p) \land Supplies(s) \land (\\
s.id = su.sid \land su.pid = p.id \land (\\
(\neg((\exists su2)(\exists p2)(Supplies(s2) \land Products(p2) \land (\\
su.id \neq su2.id \land p.id \neq p2.id \land \\
su.datetime\ != su.datetime \land \\
su2.qty < su.qty \land \\
p.hdp > 2.00 \land \\
p2.hdp\ != p.hdp)))))))\}$$

2.4.4 Domain Relational Calculus Expression for Queries

Domain Relational Calculus (DRC) is similar to tuple calculus and relational algebra. Like tuple relational calculus, DRC is a non-procedural language that filters variable uses for the domain of attributes. In addition, DRC uses similar operators as tuple calculus as well as quantifiers to bind the variables.

1.) List all orders containing product name 'Black Coffee' with the sale price less than 3.00.

$$\{< i,p > \mid Orders\,(i,p,\_,\_,\_) \wedge$$
$$(\exists c)(\exists p)(Contains(p,o,\_,s<3) \wedge$$
$$Products(i,n='Black\ Coffee',\_)$$
$$))\}$$

2.) List all the suppliers supplying less than 2 products.

$$\{< i,n > \mid Supplier\,(i,n,\_,\_,\_,\_,\_) \wedge$$
$$(\exists s)(\exists p)\,(Suppliers(s,p,q<2,d) \wedge$$
$$Products(i,n,\_)$$
$$)\}$$

3.) List the customer's first and last name with a complete status from the customer's placed order.

$$\{< f,l > \mid Customer(\_,f,\_,l,\_,\_,\_,\_,\_,\_,\_) \wedge$$
$$((\exists o)(\exists h)(\exists s)(Orders(i,\_,\_,\_,\_) \wedge ($$
$$Has(\_,\_,datetime > '2018-09-28\ 00{:}00{:}00') \wedge$$
$$Status(i,\_) \wedge$$
$$((\exists d)(Status(i,status='done') \wedge ($$
$$d.id \neq s.id$$
$$)))))\}$$

4.) List the orders placed by customers prepared by the employee 'Haleigh'.

$$\{< f, l > \mid Employee(\_, f, \_, l, \_, \_) \wedge$$
$$((\exists o)(\exists p)(\exists e2)(Orders(i, \_, \_, \_, \_) \wedge ($$
$$Prepares(\_, \_, datetime > '2018-09-28\ 00{:}00{:}00') \wedge$$
$$Employee(i, f = 'Haleigh', \_, l, \_, \_) \wedge$$
$$e2.id \neq e.id$$

5.) List the customers who placed an order once.

$$\{< i, f, l > \mid Customer(i, f, \_, l, \_, \_, \_, \_, \_, \_) \wedge$$
$$((\exists o)(Orders(i, \_, \_, \_, \_) \wedge ($$
$$\neg(\exists o2)(i, \_, ! = datetime, \_, \_) \wedge ($$
$$o2.id \neq o.id)))\}$$

6.) List the role with the least number of employees.

$$\{< i, f, l, r > \mid Employee(i, f, \_, l, r, \_)$$
$$\wedge \neg(\exists e2)(Employee\ (! = i, f, \_, l, < r, \_)\ )\}$$

7.) List the last and first name of customers who have not placed an order.

$$\{< l, f > \mid Customer(\_, f, \_, l, \_, \_, \_, \_, \_, \_) \wedge$$
$$(\neg(\exists o)(Orders(i, \_, \_, \_, \quad \_))\}$$

8.) List the first and last name of customers who at least more than 10.00 on their order.

$$\{< f, l > \mid Customer(\_, f, \_, l, \_, \_, \_, \_, \_, \_) \wedge$$
$$((\exists o)(Orders(i, \_, \_, \_, t >= 10.00))\}$$

9.) List the customer's first and last name who ordered products 'Cafe Americano' and 'Apple' or from the category 'Snacks'.

$$\{< f, l > \mid Customer(i, f, \_, l, \_, \_, \_, \_, \_, \_) \wedge$$
$$((\forall p)(\exists co)(\exists o)(\exists i)(\exists c)(Products(i, n = 'Cafe\ Americano', \_) \wedge$$
$$(Products(i, n = 'Apple', \_) \wedge$$
$$Contains(p, o, \_, \_) \wedge$$
$$Orders(i, c, \_, \_t < 8.00) \wedge$$
$$Includes(c, \quad p) \wedge$$
$$Category(o, n = 'Snacks'))\}$$

10.) List the suppliers who provided a less than one of products that cost more than 2.00 dollars.

$$\{< i, n > \mid Supplier(i, n, \_, \_, \_, \_, \_) \wedge$$
$$((\exists p)(\exists su)(Products(i, n, h > 2.00) \wedge$$
$$(Supplies(i, p, q, d) \wedge$$
$$(\neg(\exists su2)(\exists p2)(Supplies(i, \_, < q, ! = d) \wedge$$
$$Products(i, \_, ! = h) \wedge$$
$$su.id \neq su2.id \wedge$$
$$p.id \neq p2.id))))\}$$

Phase 3: Create Logical and Physical Database with Oracle DBMS

The purpose of Phase 3 is to use Postgresql to create, load, and query Kiwi's Coffee Shop database. The goal is to familiarize with the commands in the Postgresql psql interface by using Phase 2 queries by answering them in SQL language.

The first section of Phase 3 begins by defining normalization of the relations created in Phase 2. The section starts off with the first subsection of defining normalization, including defining the first, second, third, and Boyce-Codd Normal Forms. The next subsection checks and documents each of every relation to see if is in the third or BCNF.

The second section of Phase 3 describes the main purpose of psql and the functionality provided by the psql. The third section describes the schemas objects that is allowed in the postgres DBMS, expanding what they are and elaborating the purpose of the objects. The section includes to list the schema objects that were created in the project with the description of the syntax of statements for creating database schema objects by grouping the statements into subsections.

The fourth section of the phase contains for each relation to list its relation schema and its contents by using the DESC and SELECT commands to show the relation of the schema where the SELECT statement shows the contents of the relation. The fifth section of the phase includes writing the queries from Phase 2 and converting them to SQL language. The final section of the phase includes a data loader that describes the data loading methods such as INSERT INTO, VALUES, and so forth. The section includes a description of the Java Data Loader Program with additional features in the program to make it more user-friendly.

3.1 Normalization of Relations

The following section will define normalization and the rules of normalization. The rules of normalization includes the First Normal Form (1NF), Second Normal Form (2NF), Third Normal Form (3NF), and the Boyce-Codd Normal Form (BCNF). The normalization and normal forms are key aspects for the Kiwi's Coffee Shop database as a technique of organizing the data in the database.

Description of Normalization and Normal Forms

The normalization process is a systematic approach of breaking down tables to minimize data repetition and minimizing features such as Insertion, Update and Deletion Anomalies. The process is considered to be a filtering process to improve the design and is multipart which puts data into tabular form and removes replicated data from the relation tables. The overall purpose of normalization is by minimizing redundant data and ensuring data dependencies is logically stored. The consequence of a table not properly normalized comprises data redundancy that results for memory space to overload making it difficult to update the database without encountering data loss.

The normalization of data provides database designers of formal framework for analyzing relation schemas based on their keys and on the functional dependencies among their attributes. The second normalization procedure of database designers includes a series of normal form tests carried out on individual relation schemas where the relational database is normalized. The normal form of a relation contains the highest normal form condition that it meets which implies the degree that it has been normalized. The normal forms are divided from the normalization rules consisting of First Normal Form, Second Normal Form, Third Normal Form, and Boyce-Codd Normal Form.

3.1.1 First, Second, Third, and Boyce-Codd Normal Forms

First Normal Form

The first normal form (1NF) corresponds to a relation in the relational model and enforces rules to meet the 1NF criteria to designing the Kiwi's Coffee Shop database. The first rule of 1NF consists for a table to contain single (atomic) valued attributes and columns. The columns of a table should not contain multiple values where atomic values must not be divided. The second rule of 1NF comprises for the attribute domain to be unchanged. For each column, the values stored must be of the same kind. The third rule of 1NF requires for each column in a table to have unique names. In addition, this is to avoid further complexity when retrieval of stored data. The last rule of 1NF states that the order in which the data is stored does not matter. Furthermore, the first formal form (1NF) in a database is present when the conditions of containing only atomic values and no repeating groups is satisfied. In addition, if the table is not in 1NF, then the database design is substandard. The rules are self-explanatory that once they satisfy 1NF, the next step of the normalization process is progressed.

Second Normal Form(2NF)

The second normal form (2NF) comprises the approach of all attributes that are non-key columns, depend on a key. The 2NF is involved with embedded entities where one autonomous entity is embedded into the relation of another entity. The relation is in 2NF if the relation is in 1NF and for every non-key attribute to be fully functionally determined by every key of the relation. In other words, a non-key attribute that is functionally determined by only a proper subset of a key can not be found. The relation violates the 2NF criteria when there is a (H,I,J) keys in the relation and relation M is not part of any key. If there is a functional dependency of I → M, then M is functionally dependent on a subset of a key, thus making the criteria of the relation violate 2NF. The problems that stem from a relation that violates 2NF criteria requires solutions. For instance, if relation Employee is not in 2NF, the relation is decomposed into two or more relations. For a relation to be decomposed, the information from the original relation must be contained, including to prohibit introducing untrue information that were not present in the original relation. Furthermore, if a 2NF violation is determined of the form X → M, with X being a subset of a key, and M not being part of any key, then M is removed from the original relation to eliminate the 2NF violation. In addition, for a relation to be 2NF, the relation should not violate 2NF criteria. However, in the case of running into this issue it is resolved with the decomposition method.

Third Normal Form

The next normalization rule is the third normal form (3NF) with the inclusion of the second normal form (2NF). The 2NF helps identify a range of poorly designed relation but not all. The 3NF after 2NF is useful since 3NF deals with transitive dependencies with the goal of removing any non-key attributes from a new table that are more dependent on other non-key attributes than the table key. The relation R is in 3NF is R is in 2NF and if every non-key attribute is non-transitively dependent on all keys. A transitive functional dependency contains a function dependency such as B → C and is a transitive functional dependency in a relation R if there is a set of attributes Y such that B → Y and Y→ C, where Y is not a superkey. If Y is a superkey in relation R, then 3NF is not violated. The normalization rule faces 3NF violation where they could be detected when a functional dependency Y→ C such that C is a non-key attribute and C is not a superkey. The 3NF violation is resolved by using the same method previously used in 2NF, called decomposition. The exhibited violation of 3NF is removed by removing X^+ from the original relation. In addition, the primary key from the first and second relation of the decomposition must be passed down through all the steps.

Boyce-Codd Form

The Boyce-Codd Normal Form (BCNF) is derived from third normal form (3NF), where BCNF is considered a new 3NF. The relation R is in BCNF if for every functional dependency Y → B in R, where Y must be a superkey of R. The background of BCNF occurred when Boyce and Codd discovered a problem that develops when there are 2 or more composite keys in a relation. When the functional dependency of Student, Course → Instructor is checked in 3NF, the result is that the

Student and Course are the superkey. The functional dependency of Instructor -> Course, the instructor in 3NF is not the superkey and the Course is a key attribute, resulting for the relation to be in 3NF. The case of testing if the relation is in the BCNF differs from 3NF. The functional dependency of Student, Course → Instructor is checked, where the Student and Course are superkey, however when Instructor → Course is checked, the Instructor is not a superkey, therefore resulting it to be a violation. This result ends for the relation not being in the BCNF. The solution to terminate the problem is by using the decompose method to eliminate duplication of information.

Problems with Modification Anomalies

The modification anomalies host a set of problems in a database that does not allow for an SQL database be correctly structured. The problems underlying these modified anomalies are resolved by normalizing the database structure. The modified anomalies are bad properties; however, database designers may find these bad relations more efficient to use. The consequence of a bad relation in a database results in low performance where data inconsistency or data loss may occur within data stored in the database. The three set of modified anomalies consists of insert, delete, and update. The insert anomaly displays a normal behavior by inserting one item of information into the database by inserting one tuple in one or more tables without introducing any NULL values.

Insertion: The insert operation that inserts one item of information results to insert multiple tuple into some relation or use of NULL values.

Deletion: The delete anomaly normal behavior is exhibited by deleting one item of information from the database by deleting one tuple in one or more tables, and deleting only the intended information without the loss of additional information. The delete operation that deletes one item of information ends by deleting multiple tuples from some relation or causing unintended information loss.

Update: The last modified anomaly is the update anomaly with the normal behavior of updating one item of information from the database by updating one tuple in one or more tables. The update operation that updates one item of information ends by updating multiple tuples from some relation that causes the database to become inconsistent.

In addition, the normal forms introduced by Boyce and Codd were meant to resolved the problems of modified anomalies when they are not normalized. The normal forms have set of properties meant for the relations to meet. The normalize form involves functional dependency, but through the three normal forms the development of BCNF served to make 3NF stronger with less problems.

3.1.2 Normal Forms for the Database

The following will check and document each and every relations for Kiwi's Coffee Shop to see if the relations are in 3rd Normal Form or Boyce-Codd normal form. The original relations and updated relations will be listed if they were not in 3rd Normal Form or BCNF. In addition, the following includes a description of the functional dependencies, listing the candidate keys, and revising the table if they satisfy the 1NF and 2NF.

Category

Functional Dependencies

FD1. **catid** → {name}

Candidate Keys

catid

Normal Form:

1NF is satisfied since all each cell is atomic, entries in a column are same type, and rows are unique identified.
2NF is satisfied; All attributes (non-key columns) dependent on the key.
3NF is satisfied; All field (columns) can be determined only by the key in the table and no other column.
BCNF is satisfied; The left side of all functional dependencies are candidate keys.

Description

'Category' satisfies BCNF and therefore has no inherent modification abnormalities.

Customers

Functional Dependencies

FD1. **cid** → {fname, mname, lname,..,..,..., phone_no }

Candidate Keys

cid

Normal Form

1NF is satisfied since all each cell is atomic, entries in a column are same type, and rows are unique identified.
2NF is satisfied; All attributes (non-key columns) dependent on the key.
3NF is satisfied; All field (columns) can be determined only by the key in the table and no other column.
BCNF is satisfied; The left side of all functional dependencies are candidate keys.

Description

'Customers' satisfies BCNF and therefore has no inherent modification abnormalities.

Employee

Functional Dependencies

FD1. **eid** → {fname, mname,…,…,…, email}

Candidate Keys

eid

Normal Form:

1NF is satisfied since all each cell is atomic, entries in a column are same type, and rows are unique identified.
2NF is satisfied; All attributes (non-key columns) dependent on the key.
3NF is satisfied; All field (columns) can be determined only by the key in the table and no other column.
BCNF is satisfied; The left side of all functional dependencies are candidate keys.

Description

'Employee' satisfies BCNF and therefore has no inherent modification abnormalities.

Orders

Functional Dependencies

FD1. o**id, cid** → {payment, datetime, total}

Candidate Keys

oid, cid

Normal Form:

1NF is satisfied since all each cell is atomic, entries in a column are same type, and rows are unique identified.
2NF is satisfied; All attributes (non-key columns) dependent on the key.
3NF is satisfied; All field (columns) can be determined only by the key in the table and no other column.
BCNF is satisfied; The left side of all functional dependencies are candidate keys.

Description

'Orders' satisfies BCNF and therefore has no inherent modification abnormalities.

Products

Functional Dependencies

FD1. **pid** → {name, hdp}

Candidate Keys

pid

Normal Form:

1NF is satisfied since all each cell is atomic, entries in a column are same type, and rows are unique identified.
2NF is satisfied; All attributes (non-key columns) dependent on the key.
3NF is satisfied; All field (columns) can be determined only by the key in the table and no other column.
BCNF is satisfied; The left side of all functional dependencies are candidate keys.

Description

'Products' satisfies BCNF and therefore has no inherent modification abnormalities.

Status

Functional Dependencies

FD1. **stid** → {status}

Candidate Keys

stid

Normal Form:

1NF is satisfied since all each cell is atomic, entries in a column are same type, and rows are unique identified.
2NF is satisfied; All attributes (non-key columns) dependent on the key.
3NF is satisfied; All field (columns) can be determined only by the key in the table and no other column.
BCNF is satisfied; The left side of all functional dependencies are candidate keys.

Description

'Status' satisfies BCNF and therefore has no inherent modification abnormalities.

Supplier

Functional Dependencies

FD1. **sid** → {name, ...,...,..., zipcode}

Candidate Keys

sid

Normal Form:

1NF is satisfied since all each cell is atomic, entries in a column are same type, and rows are unique identified.
2NF is satisfied; All attributes (non-key columns) dependent on the key.
3NF is satisfied; All field (columns) can be determined only by the key in the table and no other column.
BCNF is satisfied; The left side of all functional dependencies are candidate keys.

Description

'Supplier' satisfies BCNF and therefore has no inherent modification abnormalities.

Contains

Functional Dependencies

FD1. **pid, oid** → {qty, sprice}

Candidate Keys

pid, oid

Normal Form:

1NF is satisfied since all each cell is atomic, entries in a column are same type, and rows are unique identified.
2NF is satisfied; All attributes (non-key columns) dependent on the key.
3NF is satisfied; All field (columns) can be determined only by the key in the table and no other column.
BCNF is satisfied; The left side of all functional dependencies are candidate keys.

Description

'Contains' satisfies BCNF and therefore has no inherent modification abnormalities.

Has

Functional Dependencies

FD1. **soid, stid** → {datetime}

Candidate Keys

soid, stid

Normal Form:

1NF is satisfied since all each cell is atomic, entries in a column are same type, and rows are unique identified.
2NF is satisfied; All attributes (non-key columns) dependent on the key.
3NF is satisfied; All field (columns) can be determined only by the key in the table and no other column.
BCNF is satisfied; The left side of all functional dependencies are candidate keys.

Description

'Has' satisfies BCNF and therefore has no inherent modification abnormalities.

Includes

Functional Dependencies

None

Candidate Keys

catid, pid

Normal Form:

1NF is satisfied since all each cell is atomic, entries in a column are same type, and rows are unique identified.
2NF is satisfied; All attributes (non-key columns) dependent on the key.
3NF is satisfied; All field (columns) can be determined only by the key in the table and no other column.
BCNF is satisfied; The left side of all functional dependencies are candidate keys.

Description

'Includes' satisfies BCNF and therefore has no inherent modification abnormalities.

Prepares

Functional Dependencies

FD1. **eid, oid** → {datetime}

Candidate Keys

eid, oid

Normal Form:

1NF is satisfied since all each cell is atomic, entries in a column are same type, and rows are unique identified.
2NF is satisfied; All attributes (non-key columns) dependent on the key.
3NF is satisfied; All field (columns) can be determined only by the key in the table and no other column.
BCNF is satisfied; The left side of all functional dependencies are candidate keys.

Description

‘Prepares’ satisfies BCNF and therefore has no inherent modification abnormalities.

Supplies

Functional Dependencies

FD1. **sid, pid** → {qty, datetime}

Candidate Keys

sid, pid

Normal Form:

1NF is satisfied since all each cell is atomic, entries in a column are same type, and rows are unique identified.
2NF is satisfied; All attributes (non-key columns) dependent on the key.
3NF is satisfied; All field (columns) can be determined only by the key in the table and no other column.
BCNF is satisfied; The left side of all functional dependencies are candidate keys.

Description

‘Supplies’ satisfies BCNF and therefore has no inherent modification abnormalities.

3.2 Main Purpose of PSQL

In the previous section, the analyzation and inspection of the entities and relations were reviewed. For the following, the process of the physical implementation will be explained. The physical implementation for the database will be PostgreSQL, using the shell psql to demonstrate sample data. The psql is a terminal-based front-end to PostgreSQL and enables to type in queries where they can be issued in PostgreSQL to view the query outputs. This includes to perform meta-commands and multiple shell features to write scripts that perform various functions. In addition, PostgreSQL has graphical user-interface that offers advanced features such as foreign key referential integrity, views, rules, subqueries, user-defined types, and more.

3.3 Schema Objects allowed Postgres DBMS

The following section describes the schema objects allowed in the Postgres database management system. This includes listing the schema objects created for Kiwi's Coffee Shop and going over the syntax of statement for creating the database schema object. These statements are grouped into subsections presented in the following as sample.

Functions

Functions in PostgreSQL are stored procedures that simplifies queries into a single function within the database. The PostgreSQL function is used to offer services composed of a set of declarations, expressions, and statements. Furthermore, PostgreSQL contains built-in functions for existing data types. In many cases, functions can perform complex logic that difficult to perform on SQL. In dynamic SQL, a function argument can be used to pass a table and view names through the 'EXECUTE' statement. In addition, functions allows for database reuse that interacts with the application with the stored procedures instead of duplicating code.

Syntax:

```
CREATE [ OR REPLACE ] FUNCTION
          name ( [ [ argmode ] [ argname ] argtype [, ...] ] )
         [ RETURNS rettype ]
{         LANGUAGE langname
         | IMMUTABLE | STABLE | VOLATILE
         | CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT | STRICT
         | [ EXTERNAL ] SECURITY INVOKER | [ EXTERNAL ] SECURITY DEFINER
          | AS 'definition'
```

```
      | AS 'obj_file', 'link_symbol'
 } ...
     [ WITH ( attribute [, ...] ) ]
```

Indexes

Indexes are physical database objects that is defined on a table column or a list of columns. In other words, they are special lookup tables that the database search engine can use to quickly retrieve data. Indexes can be used to optimize performance and validate constraints instead of checking constraints. In addition, indexes contain many index types consisting of B-tree index, hash index, generalized inverted index (GIN), generalized search tree (GiST), and block range index (BRIN) that uses different algorithms that is suited to the different types of queries. Creating an index results by default in the B-tree index that is suited to fit most cases. In addition, indexes contains unique index for data integrity, partial indexes to build over a subset of a table, multicolumn indexes to define more than one column of a table, and implicit indexes automatically created by the database server when an object is created.

Syntax:

```
CREATE [ UNIQUE ] INDEX name ON table [ USING method ]
  ( { column | ( expression ) } [ opclass ] [, ...] )
  [ TABLESPACE tablespace ]
  [ WHERE predicate ]
```

Examples in this implementation by the implicit index:

- ❖ Category_id_key
- ❖ Customer_id_key
- ❖ Employee_id_key
- ❖ Orders_id_key
- ❖ Product_id_key
- ❖ Status_id_key
- ❖ Supplier_id_key

Sequence

A ***sequence*** involves creating new sequence number generator and is a feature in the database where multiple users can generate unique integers. The sequence generator helps generate unique primary keys automatically and arranges keys across multiple rows. A sequence is created through the

CREATE SEQUENCE statement that creates and initializes a new special single-row table with the same name and is owned by the user who issues the command. In addition, sequence names must be different from other sequences, tables, index, views, and foreign keys belonging to the same schema.

Syntax:

CREATE [TEMPORARY | TEMP] SEQUENCE ***name*** [INCREMENT [BY] ***increment***]
[MINVALUE ***minvalue*** | NO MINVALUE] [MAXVALUE ***maxvalue*** | NO MAXVALUE]
[START [WITH] ***start***] [CACHE ***cache***] [[NO] CYCLE]

Examples in this implementation:

- ❖ Category_id_seq
- ❖ Customer_id_seq
- ❖ Employee_id_seq
- ❖ Orders_id_seq
- ❖ Products_id_seq
- ❖ Status_id_seq
- ❖ Supplier_id_seq

Tables

In PostgreSQL, ***tables*** is the major building block where they are used internally to implement views and sequences. Creating tables comprises of creating new empty tables in the current database, and is owned by the user issuing the commands. Tables in a specified schema is important since that is where all of the data is stored. Creating a table can be of different types where a table can be permanent or temporary by where temporary tables exist in special schemas in the user session, and the permanent table starts at creation and ends when the table is dropped. Before creating a table, users must decide the appropriate data types since changing the data type of a column is a costly operation since tables have to be locked or rewritten. Some of the factors to pay close attention to is the extensibility an data type size.

Syntax:

CREATE [[GLOBAL | LOCAL] { TEMPORARY | TEMP }] TABLE ***table_name*** ([
{ ***column_name data_type*** [DEFAULT ***default_expr***] [***column_constraint*** [...]]

```
    | table_constraint
    | LIKE parent_table [ { INCLUDING | EXCLUDING } DEFAULTS ] }
    [, ... ]
] )
[ INHERITS ( parent_table [, ... ] ) ]
[ WITH OIDS | WITHOUT OIDS ]
[ ON COMMIT { PRESERVE ROWS | DELETE ROWS | DROP } ]
[ TABLESPACE tablespace ]
```

where ***column_constraint*** is:

```
[ CONSTRAINT constraint_name ]
{ NOT NULL |
  NULL |
  UNIQUE [ USING INDEX TABLESPACE tablespace ] |
  PRIMARY KEY [ USING INDEX TABLESPACE tablespace ] |
  CHECK (expression) |
  REFERENCES reftable [ ( refcolumn ) ] [ MATCH FULL | MATCH PARTIAL | MATCH
SIMPLE ]
    [ ON DELETE action ] [ ON UPDATE action ] }
[ DEFERRABLE | NOT DEFERRABLE ] [ INITIALLY DEFERRED | INITIALLY
IMMEDIATE ]
```

and ***table_constraint*** is:

```
[ CONSTRAINT constraint_name ]
{ UNIQUE ( column_name [, ... ] ) [ USING INDEX TABLESPACE tablespace ] |
  PRIMARY KEY ( column_name [, ... ] ) [ USING INDEX TABLESPACE tablespace ] |
  CHECK ( expression ) |
  FOREIGN KEY ( column_name [, ... ] ) REFERENCES reftable [ ( refcolumn [, ... ] ) ]
    [ MATCH FULL | MATCH PARTIAL | MATCH SIMPLE ] [ ON DELETE action ] [ ON
UPDATE action ] }
```

[DEFERRABLE | NOT DEFERRABLE] [INITIALLY DEFERRED | INITIALLY IMMEDIATE]

Triggers

Triggers are database callback functions that are invoked automatically when a specific database event happens. A trigger can be cited to fire before an operation is attempted on a row and after the operation is completed. These operations can be INSERT, UPDATE, or DELETE. In addition, if a trigger fires before or instead of an event, then the trigger can skip the operation of the current row or change being inserted. However, if the trigger fires after the event, then all changes are visible to the trigger.

Syntax:

CREATE TRIGGER ***name*** { BEFORE | AFTER } { ***event*** [OR ...] }
   ON ***table*** [FOR [EACH] { ROW | STATEMENT }]
   EXECUTE PROCEDURE ***funcname*** (***arguments***)

Views

A ***view*** is a pseudo-table that are not really tables, but appear as tables when the user SELECTS the table. In other words, the pseudo-tables do not store any data in the database, but can represents a subset of a real table by selecting certain columns or joining tables. Views allows for users to structure data in a way that users find intuitive, assign permissions that restricts access to the data that other users may be limited to view, and summarize data from various table to generate reports. Moreover, since views do not create real tables, certain operations such as DELETE, INSERT, or UPDATE cannot be executed. However, a rule can be created to correct the issues to execute the operations on view if needed.

Syntax:

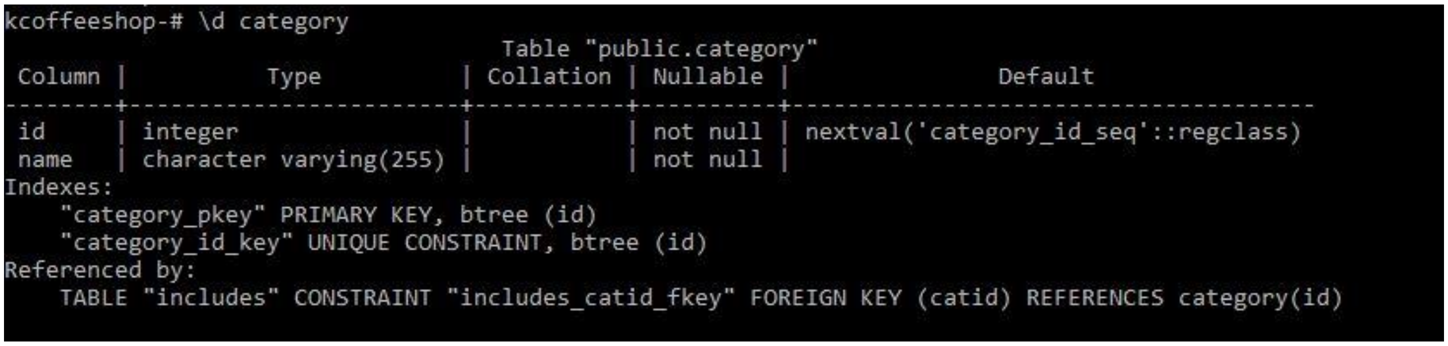
CREATE [OR REPLACE] [TEMP | TEMPORARY] VIEW ***name*** [(***column_name*** [, ...])]
   AS ***query***

3.4 List of Relations with PostgreSQL Commands

The following section contains each relation that list out the relation schema and it's information. The commands used to access the information in SQL Shell(psql) is \d+ to describe the table and select * from <table name> to show the relation schema contents.

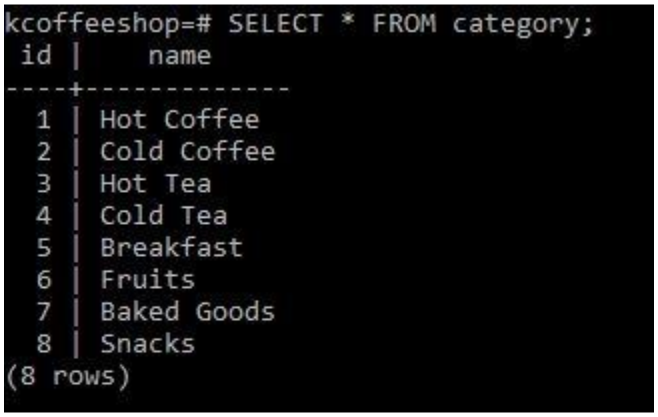
Relation- Category

\d category



```
kcoffeeshop-# \d category
                                  Table "public.category"
 Column |          Type          | Collation | Nullable |                Default
--------+------------------------+-----------+----------+---------------------------------------
 id     | integer                |           | not null | nextval('category_id_seq'::regclass)
 name   | character varying(255) |           | not null |
Indexes:
    "category_pkey" PRIMARY KEY, btree (id)
    "category_id_key" UNIQUE CONSTRAINT, btree (id)
Referenced by:
    TABLE "includes" CONSTRAINT "includes_catid_fkey" FOREIGN KEY (catid) REFERENCES category(id)
```

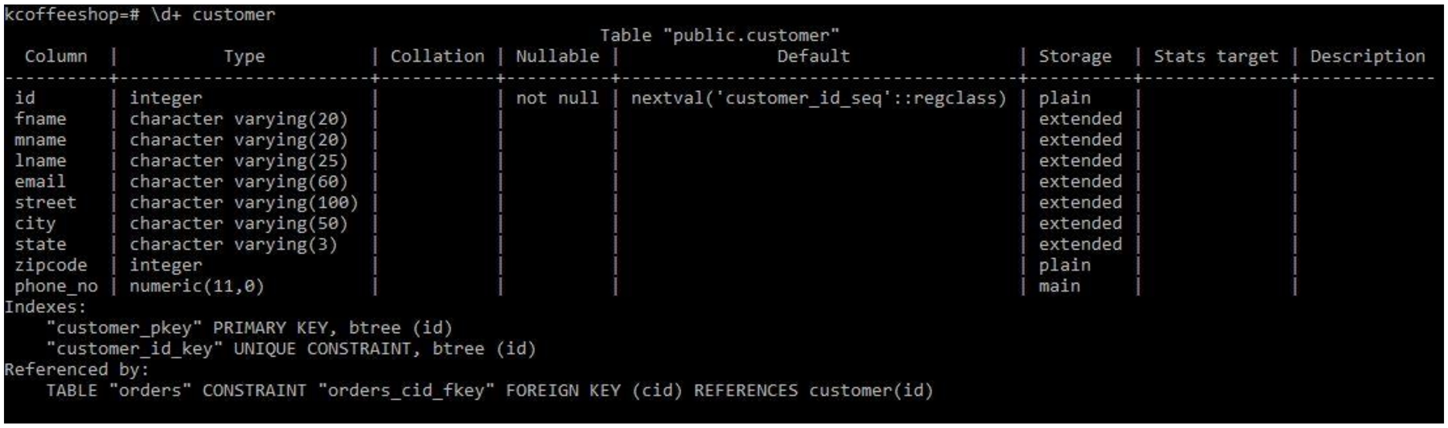
SELECT * FROM category;



```
kcoffeeshop=# SELECT * FROM category;
 id |     name
----+-------------
  1 | Hot Coffee
  2 | Cold Coffee
  3 | Hot Tea
  4 | Cold Tea
  5 | Breakfast
  6 | Fruits
  7 | Baked Goods
  8 | Snacks
(8 rows)
```

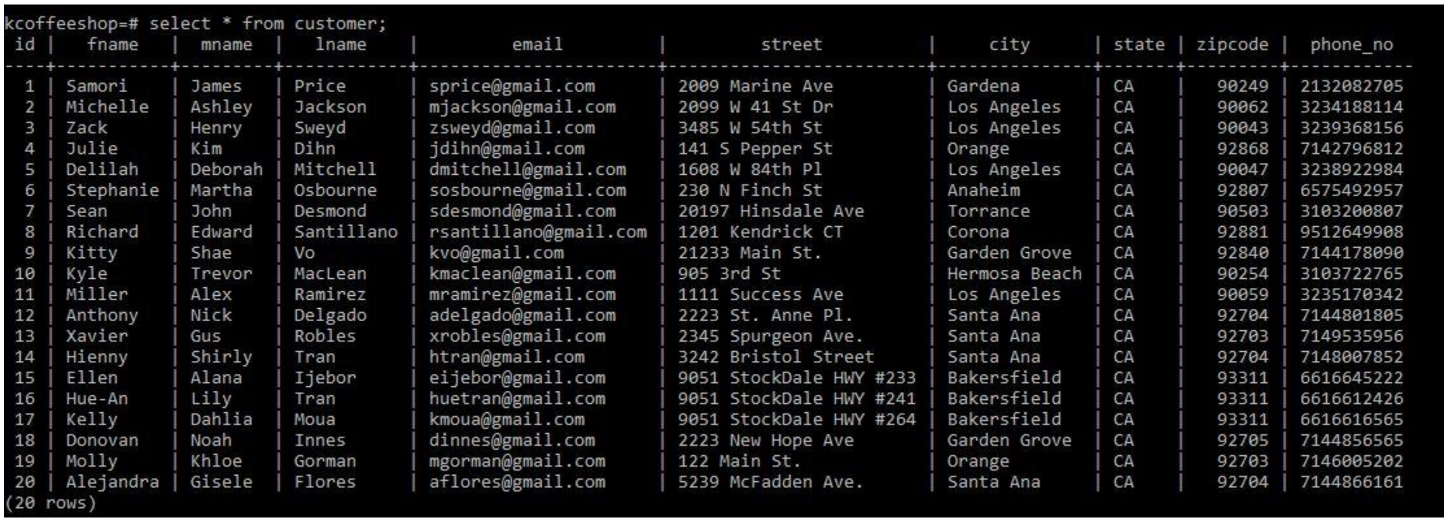
Relation- Customer

\d+ Customer



```
kcoffeeshop=# \d+ customer
                                                        Table "public.customer"
  Column  |          Type          | Collation | Nullable |                Default                | Storage  | Stats target | Description
----------+------------------------+-----------+----------+---------------------------------------+----------+--------------+-------------
 id       | integer                |           | not null | nextval('customer_id_seq'::regclass) | plain    |              |
 fname    | character varying(20)  |           |          |                                       | extended |              |
 mname    | character varying(20)  |           |          |                                       | extended |              |
 lname    | character varying(25)  |           |          |                                       | extended |              |
 email    | character varying(60)  |           |          |                                       | extended |              |
 street   | character varying(100) |           |          |                                       | extended |              |
 city     | character varying(50)  |           |          |                                       | extended |              |
 state    | character varying(3)   |           |          |                                       | extended |              |
 zipcode  | integer                |           |          |                                       | plain    |              |
 phone_no | numeric(11,0)          |           |          |                                       | main     |              |
Indexes:
    "customer_pkey" PRIMARY KEY, btree (id)
    "customer_id_key" UNIQUE CONSTRAINT, btree (id)
Referenced by:
    TABLE "orders" CONSTRAINT "orders_cid_fkey" FOREIGN KEY (cid) REFERENCES customer(id)
```

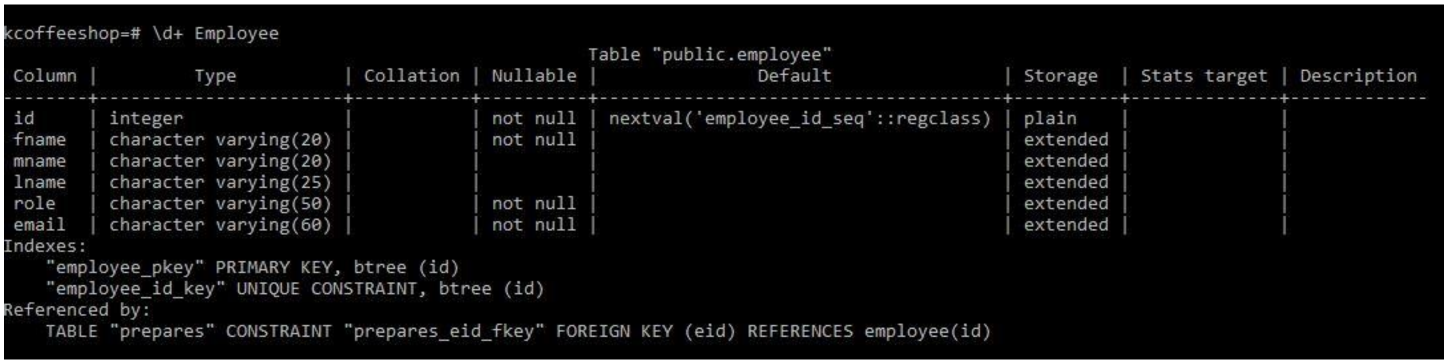
SELECT * FROM customer;



```
kcoffeeshop=# select * from customer;
 id |   fname    |  mname  |   lname    |         email          |         street          |     city      | state | zipcode |  phone_no
----+------------+---------+------------+------------------------+-------------------------+---------------+-------+---------+------------
  1 | Samori     | James   | Price      | sprice@gmail.com       | 2009 Marine Ave         | Gardena       | CA    |   90249 | 2132082705
  2 | Michelle   | Ashley  | Jackson    | mjackson@gmail.com     | 2099 W 41 St Dr         | Los Angeles   | CA    |   90062 | 3234188114
  3 | Zack       | Henry   | Sweyd      | zsweyd@gmail.com       | 3485 W 54th St          | Los Angeles   | CA    |   90043 | 3239368156
  4 | Julie      | Kim     | Dihn       | jdihn@gmail.com        | 141 S Pepper St         | Orange        | CA    |   92868 | 7142796812
  5 | Delilah    | Deborah | Mitchell   | dmitchell@gmail.com    | 1608 W 84th Pl          | Los Angeles   | CA    |   90047 | 3238922984
  6 | Stephanie  | Martha  | Osbourne   | sosbourne@gmail.com    | 230 N Finch St          | Anaheim       | CA    |   92807 | 6575492957
  7 | Sean       | John    | Desmond    | sdesmond@gmail.com     | 20197 Hinsdale Ave      | Torrance      | CA    |   90503 | 3103200807
  8 | Richard    | Edward  | Santillano | rsantillano@gmail.com  | 1201 Kendrick CT        | Corona        | CA    |   92881 | 9512649908
  9 | Kitty      | Shae    | Vo         | kvo@gmail.com          | 21233 Main St.          | Garden Grove  | CA    |   92840 | 7144178090
 10 | Kyle       | Trevor  | MacLean    | kmaclean@gmail.com     | 905 3rd St              | Hermosa Beach | CA    |   90254 | 3103722765
 11 | Miller     | Alex    | Ramirez    | mramirez@gmail.com     | 1111 Success Ave        | Los Angeles   | CA    |   90059 | 3235170342
 12 | Anthony    | Nick    | Delgado    | adelgado@gmail.com     | 2223 St. Anne Pl.       | Santa Ana     | CA    |   92704 | 7144801805
 13 | Xavier     | Gus     | Robles     | xrobles@gmail.com      | 2345 Spurgeon Ave.      | Santa Ana     | CA    |   92703 | 7149535956
 14 | Hienny     | Shirly  | Tran       | htran@gmail.com        | 3242 Bristol Street     | Santa Ana     | CA    |   92704 | 7148007852
 15 | Ellen      | Alana   | Ijebor     | eijebor@gmail.com      | 9051 StockDale HWY #233 | Bakersfield   | CA    |   93311 | 6616645222
 16 | Hue-An     | Lily    | Tran       | huetran@gmail.com      | 9051 StockDale HWY #241 | Bakersfield   | CA    |   93311 | 6616612426
 17 | Kelly      | Dahlia  | Moua       | kmoua@gmail.com        | 9051 StockDale HWY #264 | Bakersfield   | CA    |   93311 | 6616616565
 18 | Donovan    | Noah    | Innes      | dinnes@gmail.com       | 2223 New Hope Ave       | Garden Grove  | CA    |   92705 | 7144856565
 19 | Molly      | Khloe   | Gorman     | mgorman@gmail.com      | 122 Main St.            | Orange        | CA    |   92703 | 7146005202
 20 | Alejandra  | Gisele  | Flores     | aflores@gmail.com      | 5239 McFadden Ave.      | Santa Ana     | CA    |   92704 | 7144866161
(20 rows)
```

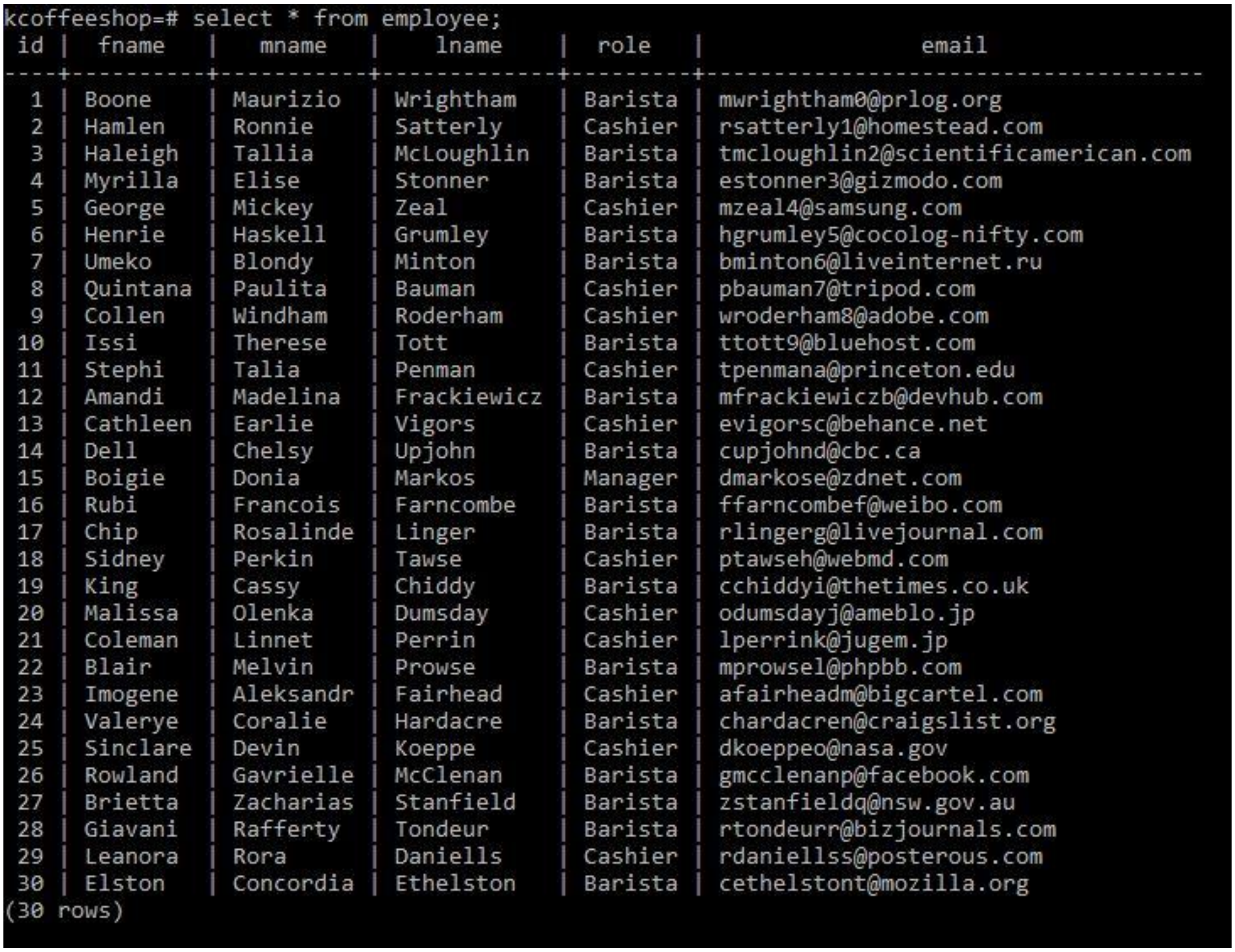
Relation- Employee

\d+ Employee



```
kcoffeeshop=# \d+ Employee
                                               Table "public.employee"
 Column |          Type          | Collation | Nullable |                 Default                  | Storage  | Stats target | Description
--------+------------------------+-----------+----------+------------------------------------------+----------+--------------+-------------
 id     | integer                |           | not null | nextval('employee_id_seq'::regclass)     | plain    |              |
 fname  | character varying(20)  |           | not null |                                          | extended |              |
 mname  | character varying(20)  |           |          |                                          | extended |              |
 lname  | character varying(25)  |           |          |                                          | extended |              |
 role   | character varying(50)  |           | not null |                                          | extended |              |
 email  | character varying(60)  |           | not null |                                          | extended |              |
Indexes:
    "employee_pkey" PRIMARY KEY, btree (id)
    "employee_id_key" UNIQUE CONSTRAINT, btree (id)
Referenced by:
    TABLE "prepares" CONSTRAINT "prepares_eid_fkey" FOREIGN KEY (eid) REFERENCES employee(id)
```

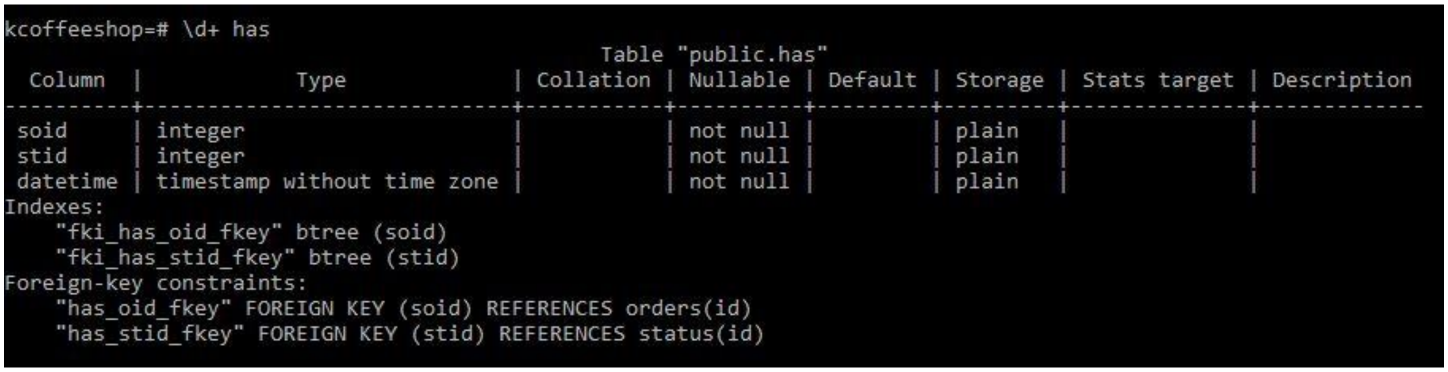
SELECT * FROM employee;



```
kcoffeeshop=# select * from employee;
 id |  fname   |   mname   |   lname    |  role   |                email
----+----------+-----------+------------+---------+------------------------------------
  1 | Boone    | Maurizio  | Wrightham  | Barista | mwrightham0@prlog.org
  2 | Hamlen   | Ronnie    | Satterly   | Cashier | rsatterly1@homestead.com
  3 | Haleigh  | Tallia    | McLoughlin | Barista | tmcloughlin2@scientificamerican.com
  4 | Myrilla  | Elise     | Stonner    | Barista | estonner3@gizmodo.com
  5 | George   | Mickey    | Zeal       | Cashier | mzeal4@samsung.com
  6 | Henrie   | Haskell   | Grumley    | Barista | hgrumley5@cocolog-nifty.com
  7 | Umeko    | Blondy    | Minton     | Barista | bminton6@liveinternet.ru
  8 | Quintana | Paulita   | Bauman     | Cashier | pbauman7@tripod.com
  9 | Collen   | Windham   | Roderham   | Cashier | wroderham8@adobe.com
 10 | Issi     | Therese   | Tott       | Barista | ttott9@bluehost.com
 11 | Stephi   | Talia     | Penman     | Cashier | tpenmana@princeton.edu
 12 | Amandi   | Madelina  | Frackiewicz | Barista | mfrackiewiczb@devhub.com
 13 | Cathleen | Earlie    | Vigors     | Cashier | evigorsc@behance.net
 14 | Dell     | Chelsy    | Upjohn     | Barista | cupjohnd@cbc.ca
 15 | Boigie   | Donia     | Markos     | Manager | dmarkose@zdnet.com
 16 | Rubi     | Francois  | Farncombe  | Barista | ffarncombef@weibo.com
 17 | Chip     | Rosalinde | Linger     | Barista | rlingerg@livejournal.com
 18 | Sidney   | Perkin    | Tawse      | Cashier | ptawseh@webmd.com
 19 | King     | Cassy     | Chiddy     | Barista | cchiddyi@thetimes.co.uk
 20 | Malissa  | Olenka    | Dumsday    | Cashier | odumsdayj@ameblo.jp
 21 | Coleman  | Linnet    | Perrin     | Cashier | lperrink@jugem.jp
 22 | Blair    | Melvin    | Prowse     | Barista | mprowsel@phpbb.com
 23 | Imogene  | Aleksandr | Fairhead   | Cashier | afairheadm@bigcartel.com
 24 | Valerye  | Coralie   | Hardacre   | Barista | chardacren@craigslist.org
 25 | Sinclare | Devin     | Koeppe     | Cashier | dkoeppeo@nasa.gov
 26 | Rowland  | Gavrielle | McClenan   | Barista | gmcclenanp@facebook.com
 27 | Brietta  | Zacharias | Stanfield  | Barista | zstanfieldq@nsw.gov.au
 28 | Giavani  | Rafferty  | Tondeur    | Barista | rtondeurr@bizjournals.com
 29 | Leanora  | Rora      | Daniells   | Cashier | rdaniellss@posterous.com
 30 | Elston   | Concordia | Ethelston  | Barista | cethelstont@mozilla.org
(30 rows)
```

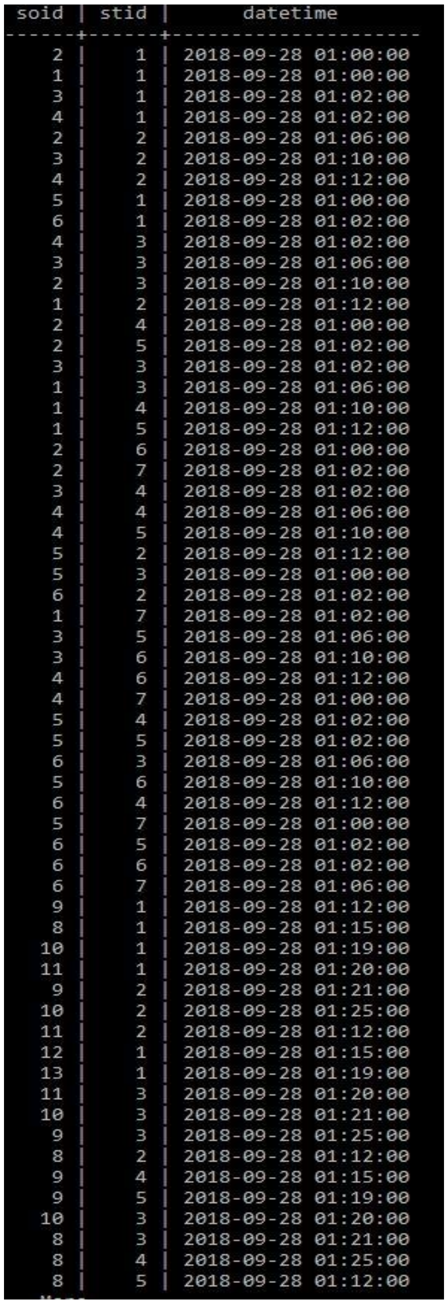
Relation- Has

\d+ Has

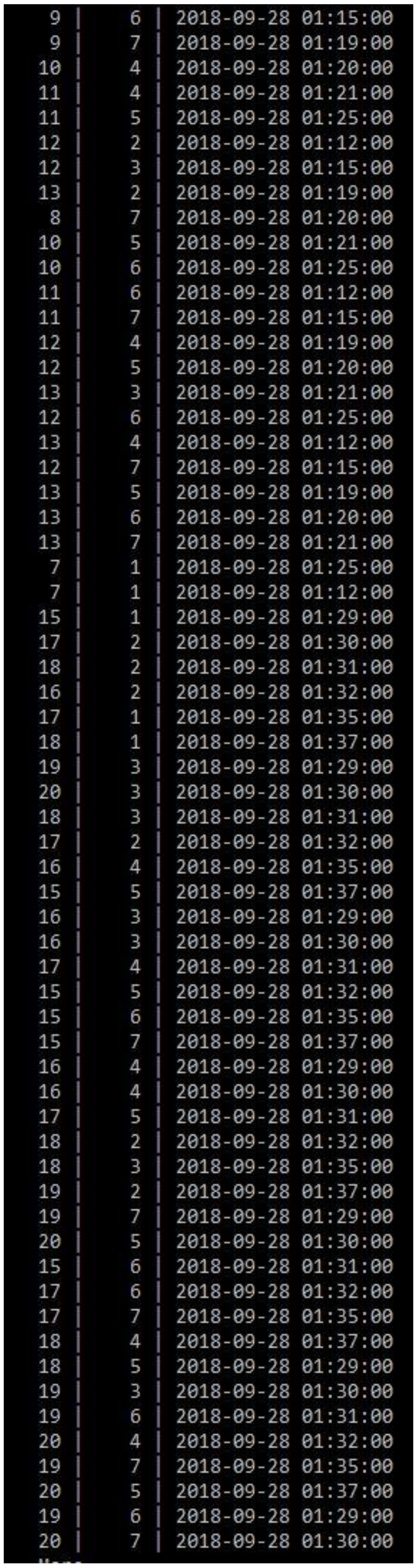


```
kcoffeeshop=# \d+ has
                                              Table "public.has"
  Column  |            Type             | Collation | Nullable | Default | Storage | Stats target | Description
----------+-----------------------------+-----------+----------+---------+---------+--------------+-------------
 soid     | integer                     |           | not null |         | plain   |              |
 stid     | integer                     |           | not null |         | plain   |              |
 datetime | timestamp without time zone |           | not null |         | plain   |              |
Indexes:
    "fki_has_oid_fkey" btree (soid)
    "fki_has_stid_fkey" btree (stid)
Foreign-key constraints:
    "has_oid_fkey" FOREIGN KEY (soid) REFERENCES orders(id)
    "has_stid_fkey" FOREIGN KEY (stid) REFERENCES status(id)
```

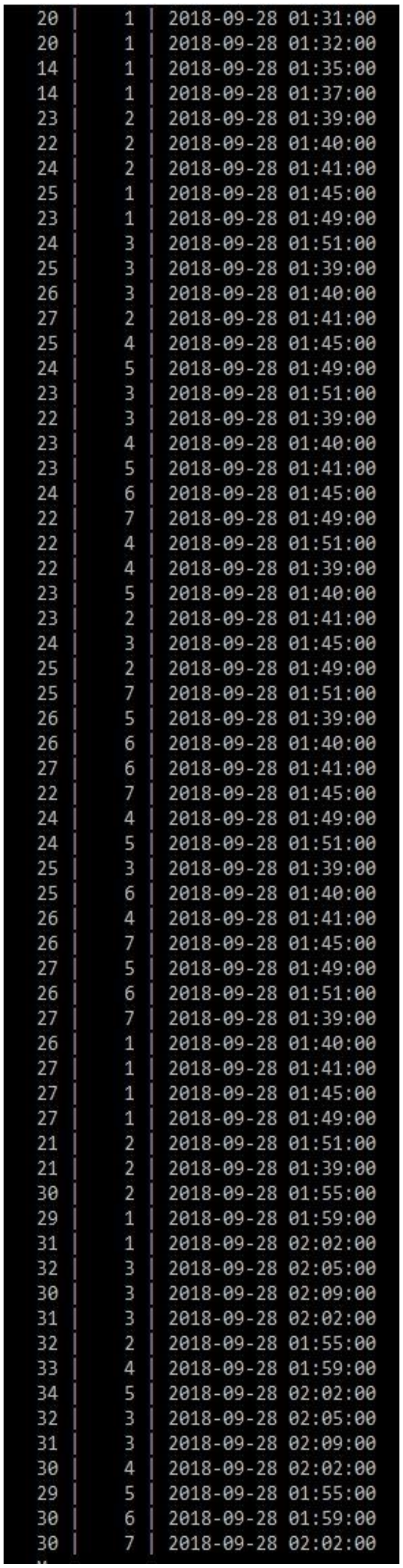
SELECT * FROM has;



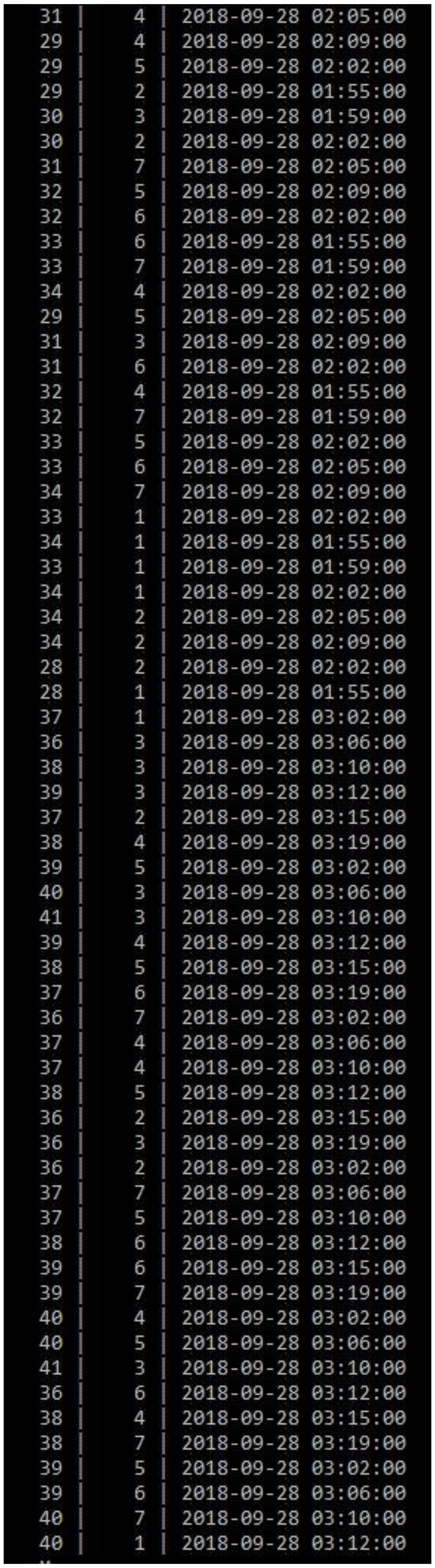
```
 soid | stid |      datetime
------+------+---------------------
    2 |    1 | 2018-09-28 01:00:00
    1 |    1 | 2018-09-28 01:00:00
    3 |    1 | 2018-09-28 01:02:00
    4 |    1 | 2018-09-28 01:02:00
    2 |    2 | 2018-09-28 01:06:00
    3 |    2 | 2018-09-28 01:10:00
    4 |    2 | 2018-09-28 01:12:00
    5 |    1 | 2018-09-28 01:00:00
    6 |    1 | 2018-09-28 01:02:00
    4 |    3 | 2018-09-28 01:02:00
    3 |    3 | 2018-09-28 01:06:00
    2 |    3 | 2018-09-28 01:10:00
    1 |    2 | 2018-09-28 01:12:00
    2 |    4 | 2018-09-28 01:00:00
    2 |    5 | 2018-09-28 01:02:00
    3 |    3 | 2018-09-28 01:02:00
    1 |    3 | 2018-09-28 01:06:00
    1 |    4 | 2018-09-28 01:10:00
    1 |    5 | 2018-09-28 01:12:00
    2 |    6 | 2018-09-28 01:00:00
    2 |    7 | 2018-09-28 01:02:00
    3 |    4 | 2018-09-28 01:02:00
    4 |    4 | 2018-09-28 01:06:00
    4 |    5 | 2018-09-28 01:10:00
    5 |    2 | 2018-09-28 01:12:00
    5 |    3 | 2018-09-28 01:00:00
    6 |    2 | 2018-09-28 01:02:00
    1 |    7 | 2018-09-28 01:02:00
    3 |    5 | 2018-09-28 01:06:00
    3 |    6 | 2018-09-28 01:10:00
    4 |    6 | 2018-09-28 01:12:00
    4 |    7 | 2018-09-28 01:00:00
    5 |    4 | 2018-09-28 01:02:00
    5 |    5 | 2018-09-28 01:02:00
    6 |    3 | 2018-09-28 01:06:00
    5 |    6 | 2018-09-28 01:10:00
    6 |    4 | 2018-09-28 01:12:00
    5 |    7 | 2018-09-28 01:00:00
    6 |    5 | 2018-09-28 01:02:00
    6 |    6 | 2018-09-28 01:02:00
    6 |    7 | 2018-09-28 01:06:00
    9 |    1 | 2018-09-28 01:12:00
    8 |    1 | 2018-09-28 01:15:00
   10 |    1 | 2018-09-28 01:19:00
   11 |    1 | 2018-09-28 01:20:00
    9 |    2 | 2018-09-28 01:21:00
   10 |    2 | 2018-09-28 01:25:00
   11 |    2 | 2018-09-28 01:12:00
   12 |    1 | 2018-09-28 01:15:00
   13 |    1 | 2018-09-28 01:19:00
   11 |    3 | 2018-09-28 01:20:00
   10 |    3 | 2018-09-28 01:21:00
    9 |    3 | 2018-09-28 01:25:00
    8 |    2 | 2018-09-28 01:12:00
    9 |    4 | 2018-09-28 01:15:00
    9 |    5 | 2018-09-28 01:19:00
   10 |    3 | 2018-09-28 01:20:00
    8 |    3 | 2018-09-28 01:21:00
    8 |    4 | 2018-09-28 01:25:00
    8 |    5 | 2018-09-28 01:12:00
```



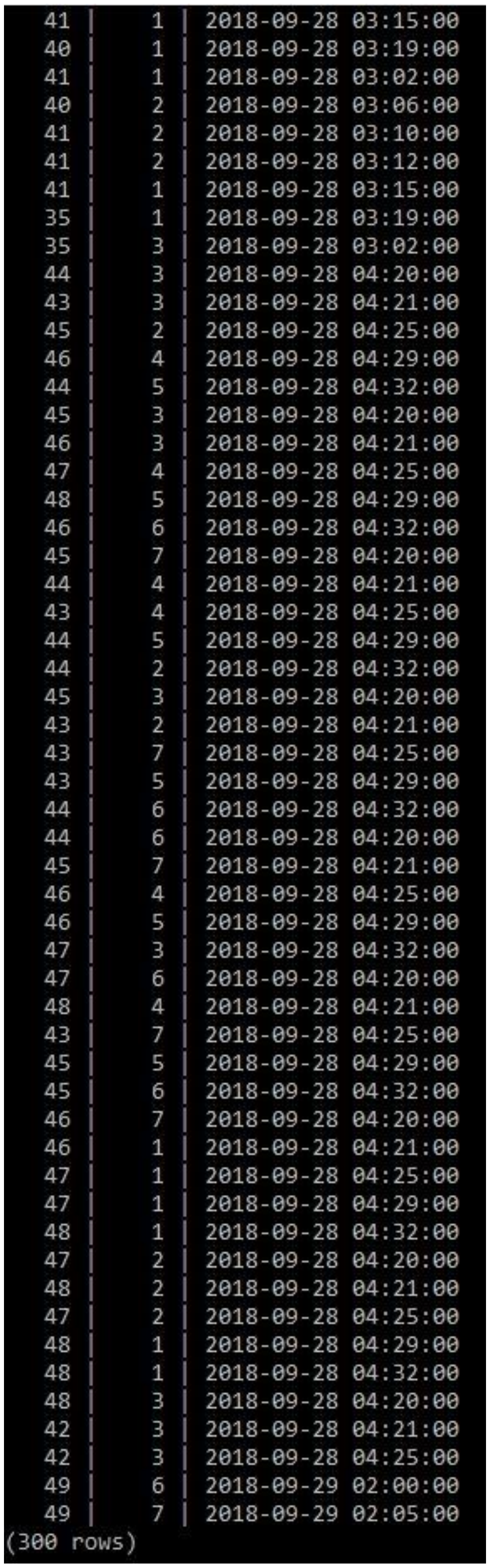
```
    9 |     6 | 2018-09-28 01:15:00
    9 |     7 | 2018-09-28 01:19:00
   10 |     4 | 2018-09-28 01:20:00
   11 |     4 | 2018-09-28 01:21:00
   11 |     5 | 2018-09-28 01:25:00
   12 |     2 | 2018-09-28 01:12:00
   12 |     3 | 2018-09-28 01:15:00
   13 |     2 | 2018-09-28 01:19:00
    8 |     7 | 2018-09-28 01:20:00
   10 |     5 | 2018-09-28 01:21:00
   10 |     6 | 2018-09-28 01:25:00
   11 |     6 | 2018-09-28 01:12:00
   11 |     7 | 2018-09-28 01:15:00
   12 |     4 | 2018-09-28 01:19:00
   12 |     5 | 2018-09-28 01:20:00
   13 |     3 | 2018-09-28 01:21:00
   12 |     6 | 2018-09-28 01:25:00
   13 |     4 | 2018-09-28 01:12:00
   12 |     7 | 2018-09-28 01:15:00
   13 |     5 | 2018-09-28 01:19:00
   13 |     6 | 2018-09-28 01:20:00
   13 |     7 | 2018-09-28 01:21:00
    7 |     1 | 2018-09-28 01:25:00
    7 |     1 | 2018-09-28 01:12:00
   15 |     1 | 2018-09-28 01:29:00
   17 |     2 | 2018-09-28 01:30:00
   18 |     2 | 2018-09-28 01:31:00
   16 |     2 | 2018-09-28 01:32:00
   17 |     1 | 2018-09-28 01:35:00
   18 |     1 | 2018-09-28 01:37:00
   19 |     3 | 2018-09-28 01:29:00
   20 |     3 | 2018-09-28 01:30:00
   18 |     3 | 2018-09-28 01:31:00
   17 |     2 | 2018-09-28 01:32:00
   16 |     4 | 2018-09-28 01:35:00
   15 |     5 | 2018-09-28 01:37:00
   16 |     3 | 2018-09-28 01:29:00
   16 |     3 | 2018-09-28 01:30:00
   17 |     4 | 2018-09-28 01:31:00
   15 |     5 | 2018-09-28 01:32:00
   15 |     6 | 2018-09-28 01:35:00
   15 |     7 | 2018-09-28 01:37:00
   16 |     4 | 2018-09-28 01:29:00
   16 |     4 | 2018-09-28 01:30:00
   17 |     5 | 2018-09-28 01:31:00
   18 |     2 | 2018-09-28 01:32:00
   18 |     3 | 2018-09-28 01:35:00
   19 |     2 | 2018-09-28 01:37:00
   19 |     7 | 2018-09-28 01:29:00
   20 |     5 | 2018-09-28 01:30:00
   15 |     6 | 2018-09-28 01:31:00
   17 |     6 | 2018-09-28 01:32:00
   17 |     7 | 2018-09-28 01:35:00
   18 |     4 | 2018-09-28 01:37:00
   18 |     5 | 2018-09-28 01:29:00
   19 |     3 | 2018-09-28 01:30:00
   19 |     6 | 2018-09-28 01:31:00
   20 |     4 | 2018-09-28 01:32:00
   19 |     7 | 2018-09-28 01:35:00
   20 |     5 | 2018-09-28 01:37:00
   19 |     6 | 2018-09-28 01:29:00
   20 |     7 | 2018-09-28 01:30:00
```



```
     20 |       1 | 2018-09-28 01:31:00
     20 |       1 | 2018-09-28 01:32:00
     14 |       1 | 2018-09-28 01:35:00
     14 |       1 | 2018-09-28 01:37:00
     23 |       2 | 2018-09-28 01:39:00
     22 |       2 | 2018-09-28 01:40:00
     24 |       2 | 2018-09-28 01:41:00
     25 |       1 | 2018-09-28 01:45:00
     23 |       1 | 2018-09-28 01:49:00
     24 |       3 | 2018-09-28 01:51:00
     25 |       3 | 2018-09-28 01:39:00
     26 |       3 | 2018-09-28 01:40:00
     27 |       2 | 2018-09-28 01:41:00
     25 |       4 | 2018-09-28 01:45:00
     24 |       5 | 2018-09-28 01:49:00
     23 |       3 | 2018-09-28 01:51:00
     22 |       3 | 2018-09-28 01:39:00
     23 |       4 | 2018-09-28 01:40:00
     23 |       5 | 2018-09-28 01:41:00
     24 |       6 | 2018-09-28 01:45:00
     22 |       7 | 2018-09-28 01:49:00
     22 |       4 | 2018-09-28 01:51:00
     22 |       4 | 2018-09-28 01:39:00
     23 |       5 | 2018-09-28 01:40:00
     23 |       2 | 2018-09-28 01:41:00
     24 |       3 | 2018-09-28 01:45:00
     25 |       2 | 2018-09-28 01:49:00
     25 |       7 | 2018-09-28 01:51:00
     26 |       5 | 2018-09-28 01:39:00
     26 |       6 | 2018-09-28 01:40:00
     27 |       6 | 2018-09-28 01:41:00
     22 |       7 | 2018-09-28 01:45:00
     24 |       4 | 2018-09-28 01:49:00
     24 |       5 | 2018-09-28 01:51:00
     25 |       3 | 2018-09-28 01:39:00
     25 |       6 | 2018-09-28 01:40:00
     26 |       4 | 2018-09-28 01:41:00
     26 |       7 | 2018-09-28 01:45:00
     27 |       5 | 2018-09-28 01:49:00
     26 |       6 | 2018-09-28 01:51:00
     27 |       7 | 2018-09-28 01:39:00
     26 |       1 | 2018-09-28 01:40:00
     27 |       1 | 2018-09-28 01:41:00
     27 |       1 | 2018-09-28 01:45:00
     27 |       1 | 2018-09-28 01:49:00
     21 |       2 | 2018-09-28 01:51:00
     21 |       2 | 2018-09-28 01:39:00
     30 |       2 | 2018-09-28 01:55:00
     29 |       1 | 2018-09-28 01:59:00
     31 |       1 | 2018-09-28 02:02:00
     32 |       3 | 2018-09-28 02:05:00
     30 |       3 | 2018-09-28 02:09:00
     31 |       3 | 2018-09-28 02:02:00
     32 |       2 | 2018-09-28 01:55:00
     33 |       4 | 2018-09-28 01:59:00
     34 |       5 | 2018-09-28 02:02:00
     32 |       3 | 2018-09-28 02:05:00
     31 |       3 | 2018-09-28 02:09:00
     30 |       4 | 2018-09-28 02:02:00
     29 |       5 | 2018-09-28 01:55:00
     30 |       6 | 2018-09-28 01:59:00
     30 |       7 | 2018-09-28 02:02:00
```



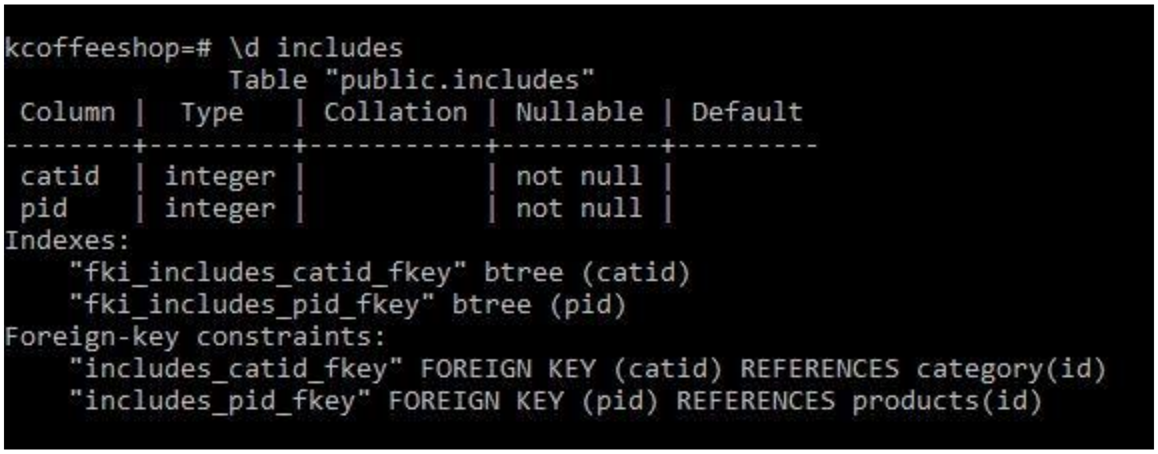
```
   31 |      4 | 2018-09-28 02:05:00
   29 |      4 | 2018-09-28 02:09:00
   29 |      5 | 2018-09-28 02:02:00
   29 |      2 | 2018-09-28 01:55:00
   30 |      3 | 2018-09-28 01:59:00
   30 |      2 | 2018-09-28 02:02:00
   31 |      7 | 2018-09-28 02:05:00
   32 |      5 | 2018-09-28 02:09:00
   32 |      6 | 2018-09-28 02:02:00
   33 |      6 | 2018-09-28 01:55:00
   33 |      7 | 2018-09-28 01:59:00
   34 |      4 | 2018-09-28 02:02:00
   29 |      5 | 2018-09-28 02:05:00
   31 |      3 | 2018-09-28 02:09:00
   31 |      6 | 2018-09-28 02:02:00
   32 |      4 | 2018-09-28 01:55:00
   32 |      7 | 2018-09-28 01:59:00
   33 |      5 | 2018-09-28 02:02:00
   33 |      6 | 2018-09-28 02:05:00
   34 |      7 | 2018-09-28 02:09:00
   33 |      1 | 2018-09-28 02:02:00
   34 |      1 | 2018-09-28 01:55:00
   33 |      1 | 2018-09-28 01:59:00
   34 |      1 | 2018-09-28 02:02:00
   34 |      2 | 2018-09-28 02:05:00
   34 |      2 | 2018-09-28 02:09:00
   28 |      2 | 2018-09-28 02:02:00
   28 |      1 | 2018-09-28 01:55:00
   37 |      1 | 2018-09-28 03:02:00
   36 |      3 | 2018-09-28 03:06:00
   38 |      3 | 2018-09-28 03:10:00
   39 |      3 | 2018-09-28 03:12:00
   37 |      2 | 2018-09-28 03:15:00
   38 |      4 | 2018-09-28 03:19:00
   39 |      5 | 2018-09-28 03:02:00
   40 |      3 | 2018-09-28 03:06:00
   41 |      3 | 2018-09-28 03:10:00
   39 |      4 | 2018-09-28 03:12:00
   38 |      5 | 2018-09-28 03:15:00
   37 |      6 | 2018-09-28 03:19:00
   36 |      7 | 2018-09-28 03:02:00
   37 |      4 | 2018-09-28 03:06:00
   37 |      4 | 2018-09-28 03:10:00
   38 |      5 | 2018-09-28 03:12:00
   36 |      2 | 2018-09-28 03:15:00
   36 |      3 | 2018-09-28 03:19:00
   36 |      2 | 2018-09-28 03:02:00
   37 |      7 | 2018-09-28 03:06:00
   37 |      5 | 2018-09-28 03:10:00
   38 |      6 | 2018-09-28 03:12:00
   39 |      6 | 2018-09-28 03:15:00
   39 |      7 | 2018-09-28 03:19:00
   40 |      4 | 2018-09-28 03:02:00
   40 |      5 | 2018-09-28 03:06:00
   41 |      3 | 2018-09-28 03:10:00
   36 |      6 | 2018-09-28 03:12:00
   38 |      4 | 2018-09-28 03:15:00
   38 |      7 | 2018-09-28 03:19:00
   39 |      5 | 2018-09-28 03:02:00
   39 |      6 | 2018-09-28 03:06:00
   40 |      7 | 2018-09-28 03:10:00
   40 |      1 | 2018-09-28 03:12:00
```



```
    41 |     1 | 2018-09-28 03:15:00
    40 |     1 | 2018-09-28 03:19:00
    41 |     1 | 2018-09-28 03:02:00
    40 |     2 | 2018-09-28 03:06:00
    41 |     2 | 2018-09-28 03:10:00
    41 |     2 | 2018-09-28 03:12:00
    41 |     1 | 2018-09-28 03:15:00
    35 |     1 | 2018-09-28 03:19:00
    35 |     3 | 2018-09-28 03:02:00
    44 |     3 | 2018-09-28 04:20:00
    43 |     3 | 2018-09-28 04:21:00
    45 |     2 | 2018-09-28 04:25:00
    46 |     4 | 2018-09-28 04:29:00
    44 |     5 | 2018-09-28 04:32:00
    45 |     3 | 2018-09-28 04:20:00
    46 |     3 | 2018-09-28 04:21:00
    47 |     4 | 2018-09-28 04:25:00
    48 |     5 | 2018-09-28 04:29:00
    46 |     6 | 2018-09-28 04:32:00
    45 |     7 | 2018-09-28 04:20:00
    44 |     4 | 2018-09-28 04:21:00
    43 |     4 | 2018-09-28 04:25:00
    44 |     5 | 2018-09-28 04:29:00
    44 |     2 | 2018-09-28 04:32:00
    45 |     3 | 2018-09-28 04:20:00
    43 |     2 | 2018-09-28 04:21:00
    43 |     7 | 2018-09-28 04:25:00
    43 |     5 | 2018-09-28 04:29:00
    44 |     6 | 2018-09-28 04:32:00
    44 |     6 | 2018-09-28 04:20:00
    45 |     7 | 2018-09-28 04:21:00
    46 |     4 | 2018-09-28 04:25:00
    46 |     5 | 2018-09-28 04:29:00
    47 |     3 | 2018-09-28 04:32:00
    47 |     6 | 2018-09-28 04:20:00
    48 |     4 | 2018-09-28 04:21:00
    43 |     7 | 2018-09-28 04:25:00
    45 |     5 | 2018-09-28 04:29:00
    45 |     6 | 2018-09-28 04:32:00
    46 |     7 | 2018-09-28 04:20:00
    46 |     1 | 2018-09-28 04:21:00
    47 |     1 | 2018-09-28 04:25:00
    47 |     1 | 2018-09-28 04:29:00
    48 |     1 | 2018-09-28 04:32:00
    47 |     2 | 2018-09-28 04:20:00
    48 |     2 | 2018-09-28 04:21:00
    47 |     2 | 2018-09-28 04:25:00
    48 |     1 | 2018-09-28 04:29:00
    48 |     1 | 2018-09-28 04:32:00
    48 |     3 | 2018-09-28 04:20:00
    42 |     3 | 2018-09-28 04:21:00
    42 |     3 | 2018-09-28 04:25:00
    49 |     6 | 2018-09-29 02:00:00
    49 |     7 | 2018-09-29 02:05:00
(300 rows)
```

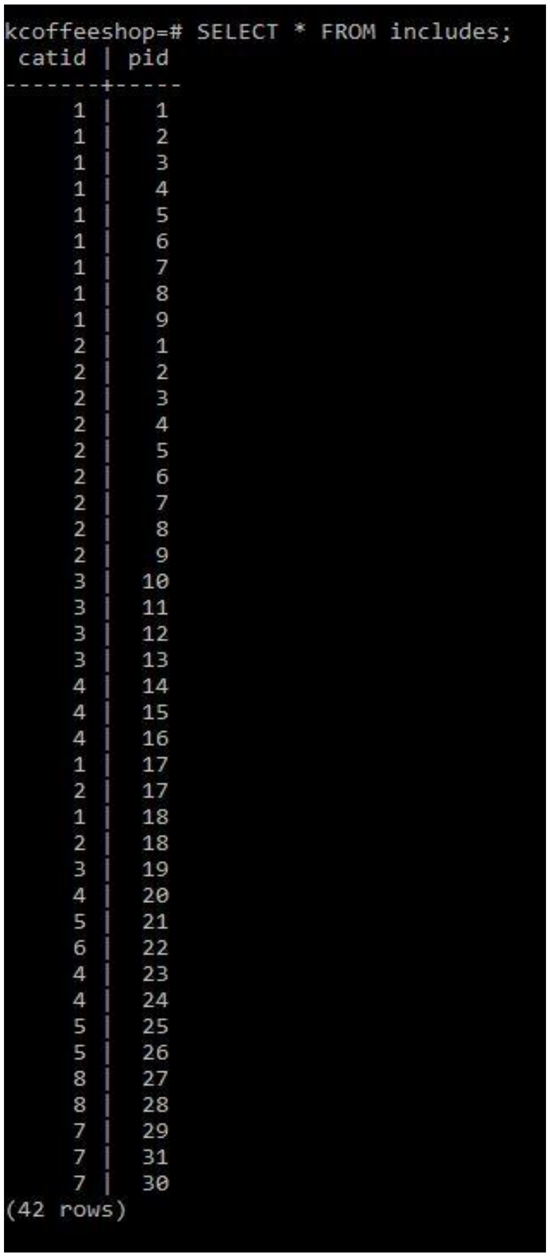
Relation- Includes

\d Includes



```
kcoffeeshop=# \d includes
              Table "public.includes"
 Column |  Type   | Collation | Nullable | Default
--------+---------+-----------+----------+---------
 catid  | integer |           | not null |
 pid    | integer |           | not null |
Indexes:
    "fki_includes_catid_fkey" btree (catid)
    "fki_includes_pid_fkey" btree (pid)
Foreign-key constraints:
    "includes_catid_fkey" FOREIGN KEY (catid) REFERENCES category(id)
    "includes_pid_fkey" FOREIGN KEY (pid) REFERENCES products(id)
```

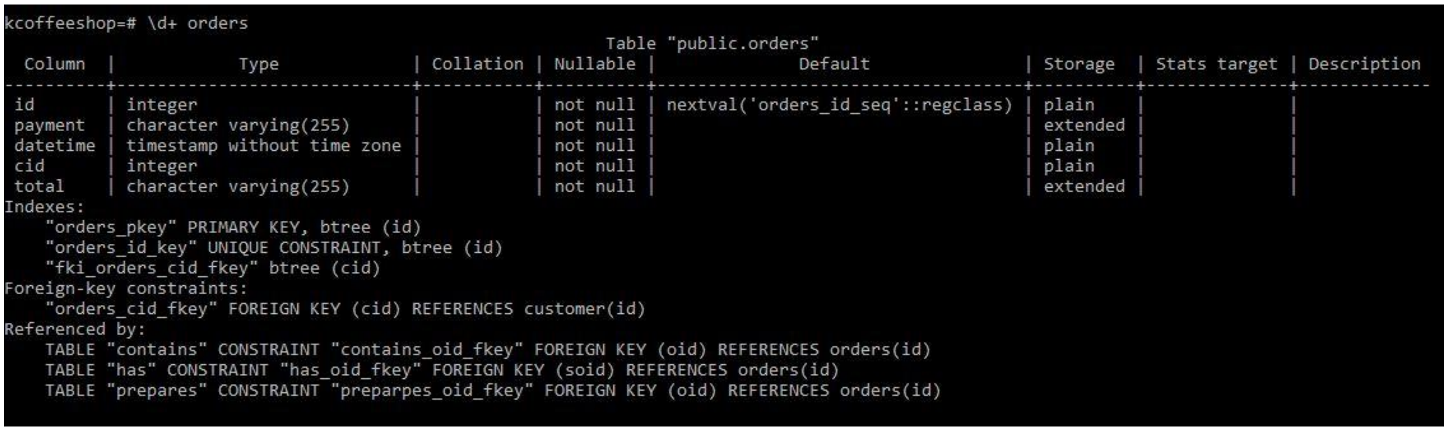
SELECT * FROM includes;



```
kcoffeeshop=# SELECT * FROM includes;
 catid | pid
-------+-----
     1 |   1
     1 |   2
     1 |   3
     1 |   4
     1 |   5
     1 |   6
     1 |   7
     1 |   8
     1 |   9
     2 |   1
     2 |   2
     2 |   3
     2 |   4
     2 |   5
     2 |   6
     2 |   7
     2 |   8
     2 |   9
     3 |  10
     3 |  11
     3 |  12
     3 |  13
     4 |  14
     4 |  15
     4 |  16
     1 |  17
     2 |  17
     1 |  18
     2 |  18
     3 |  19
     4 |  20
     5 |  21
     6 |  22
     4 |  23
     4 |  24
     5 |  25
     5 |  26
     8 |  27
     8 |  28
     7 |  29
     7 |  31
     7 |  30
(42 rows)
```

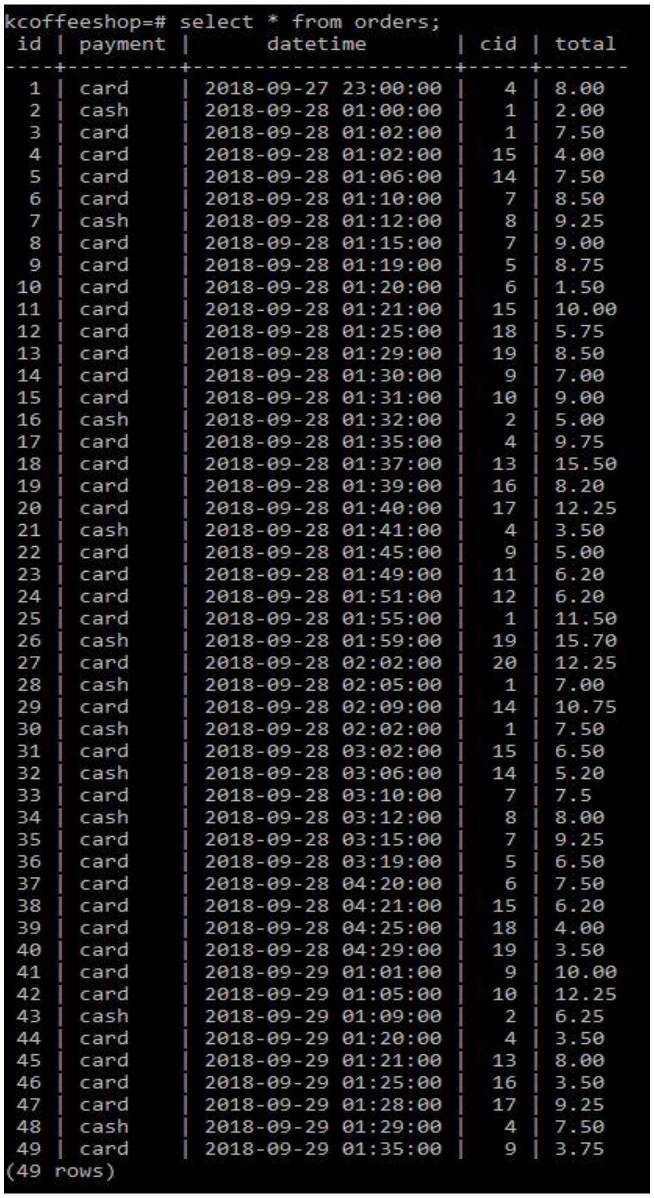
Relation- Orders

\d+ Orders



```
kcoffeeshop=# \d+ orders
                                                         Table "public.orders"
  Column  |            Type             | Collation | Nullable |               Default               | Storage  | Stats target | Description
----------+-----------------------------+-----------+----------+-------------------------------------+----------+--------------+-------------
 id       | integer                     |           | not null | nextval('orders_id_seq'::regclass) | plain    |              |
 payment  | character varying(255)      |           | not null |                                     | extended |              |
 datetime | timestamp without time zone |           | not null |                                     | plain    |              |
 cid      | integer                     |           | not null |                                     | plain    |              |
 total    | character varying(255)      |           | not null |                                     | extended |              |
Indexes:
    "orders_pkey" PRIMARY KEY, btree (id)
    "orders_id_key" UNIQUE CONSTRAINT, btree (id)
    "fki_orders_cid_fkey" btree (cid)
Foreign-key constraints:
    "orders_cid_fkey" FOREIGN KEY (cid) REFERENCES customer(id)
Referenced by:
    TABLE "contains" CONSTRAINT "contains_oid_fkey" FOREIGN KEY (oid) REFERENCES orders(id)
    TABLE "has" CONSTRAINT "has_oid_fkey" FOREIGN KEY (soid) REFERENCES orders(id)
    TABLE "prepares" CONSTRAINT "preparpes_oid_fkey" FOREIGN KEY (oid) REFERENCES orders(id)
```

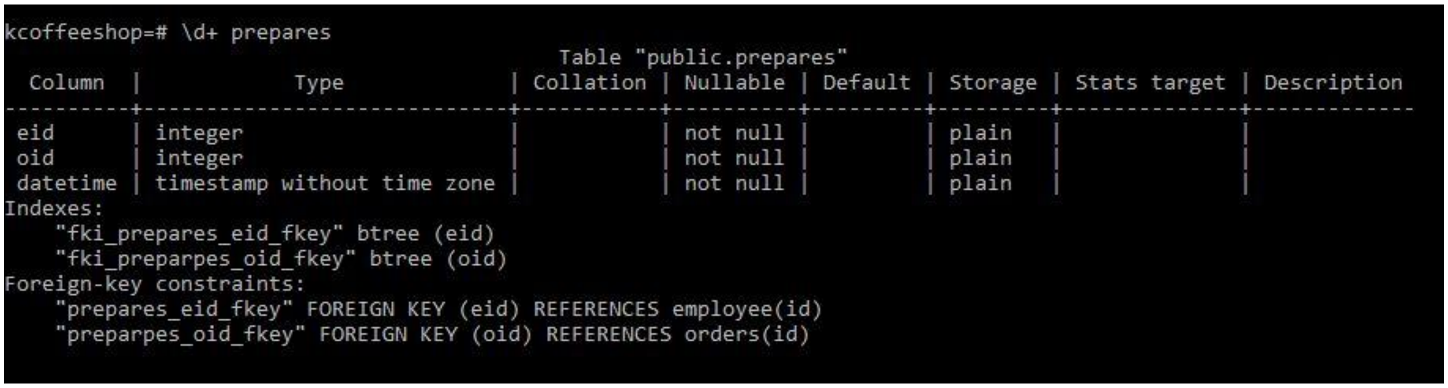
SELECT * FROM orders;



```
kcoffeeshop=# select * from orders;
 id | payment |      datetime       | cid | total
----+---------+---------------------+-----+-------
  1 | card    | 2018-09-27 23:00:00 |   4 | 8.00
  2 | cash    | 2018-09-28 01:00:00 |   1 | 2.00
  3 | card    | 2018-09-28 01:02:00 |   1 | 7.50
  4 | card    | 2018-09-28 01:02:00 |  15 | 4.00
  5 | card    | 2018-09-28 01:06:00 |  14 | 7.50
  6 | card    | 2018-09-28 01:10:00 |   7 | 8.50
  7 | cash    | 2018-09-28 01:12:00 |   8 | 9.25
  8 | card    | 2018-09-28 01:15:00 |   7 | 9.00
  9 | card    | 2018-09-28 01:19:00 |   5 | 8.75
 10 | card    | 2018-09-28 01:20:00 |   6 | 1.50
 11 | card    | 2018-09-28 01:21:00 |  15 | 10.00
 12 | card    | 2018-09-28 01:25:00 |  18 | 5.75
 13 | card    | 2018-09-28 01:29:00 |  19 | 8.50
 14 | card    | 2018-09-28 01:30:00 |   9 | 7.00
 15 | card    | 2018-09-28 01:31:00 |  10 | 9.00
 16 | cash    | 2018-09-28 01:32:00 |   2 | 5.00
 17 | card    | 2018-09-28 01:35:00 |   4 | 9.75
 18 | card    | 2018-09-28 01:37:00 |  13 | 15.50
 19 | card    | 2018-09-28 01:39:00 |  16 | 8.20
 20 | card    | 2018-09-28 01:40:00 |  17 | 12.25
 21 | cash    | 2018-09-28 01:41:00 |   4 | 3.50
 22 | card    | 2018-09-28 01:45:00 |   9 | 5.00
 23 | card    | 2018-09-28 01:49:00 |  11 | 6.20
 24 | card    | 2018-09-28 01:51:00 |  12 | 6.20
 25 | card    | 2018-09-28 01:55:00 |   1 | 11.50
 26 | cash    | 2018-09-28 01:59:00 |  19 | 15.70
 27 | card    | 2018-09-28 02:02:00 |  20 | 12.25
 28 | cash    | 2018-09-28 02:05:00 |   1 | 7.00
 29 | card    | 2018-09-28 02:09:00 |  14 | 10.75
 30 | cash    | 2018-09-28 02:02:00 |   1 | 7.50
 31 | card    | 2018-09-28 03:02:00 |  15 | 6.50
 32 | cash    | 2018-09-28 03:06:00 |  14 | 5.20
 33 | card    | 2018-09-28 03:10:00 |   7 | 7.5
 34 | cash    | 2018-09-28 03:12:00 |   8 | 8.00
 35 | card    | 2018-09-28 03:15:00 |   7 | 9.25
 36 | card    | 2018-09-28 03:19:00 |   5 | 6.50
 37 | card    | 2018-09-28 04:20:00 |   6 | 7.50
 38 | card    | 2018-09-28 04:21:00 |  15 | 6.20
 39 | card    | 2018-09-28 04:25:00 |  18 | 4.00
 40 | card    | 2018-09-28 04:29:00 |  19 | 3.50
 41 | card    | 2018-09-29 01:01:00 |   9 | 10.00
 42 | card    | 2018-09-29 01:05:00 |  10 | 12.25
 43 | cash    | 2018-09-29 01:09:00 |   2 | 6.25
 44 | card    | 2018-09-29 01:20:00 |   4 | 3.50
 45 | card    | 2018-09-29 01:21:00 |  13 | 8.00
 46 | card    | 2018-09-29 01:25:00 |  16 | 3.50
 47 | card    | 2018-09-29 01:28:00 |  17 | 9.25
 48 | cash    | 2018-09-29 01:29:00 |   4 | 7.50
 49 | card    | 2018-09-29 01:35:00 |   9 | 3.75
(49 rows)
```

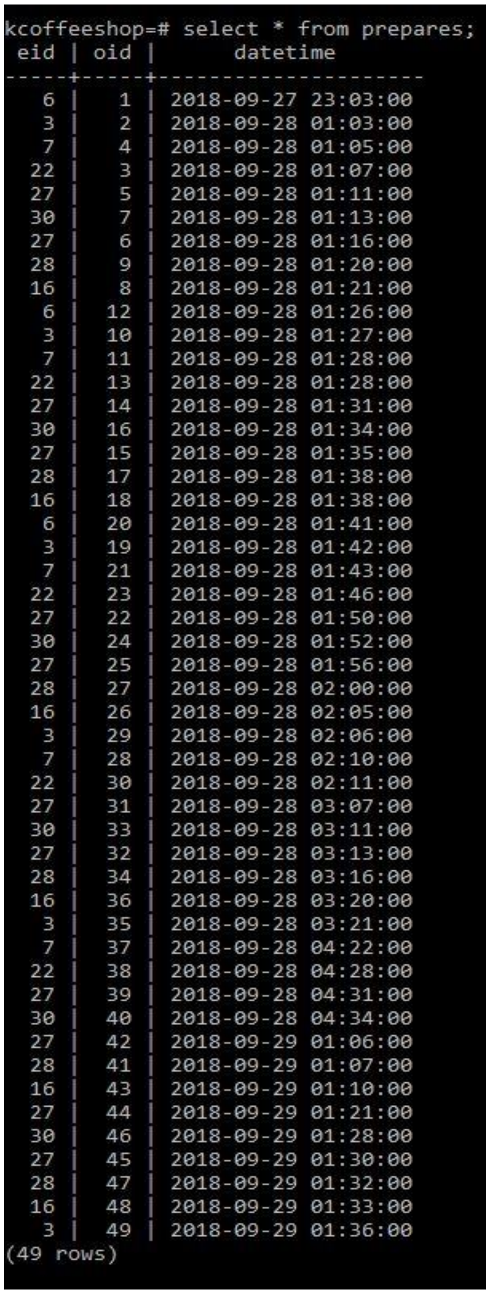
Relation- Prepares

\d+ Prepapres



```
kcoffeeshop=# \d+ prepares
                                          Table "public.prepares"
  Column  |            Type             | Collation | Nullable | Default | Storage | Stats target | Description
----------+-----------------------------+-----------+----------+---------+---------+--------------+-------------
 eid      | integer                     |           | not null |         | plain   |              |
 oid      | integer                     |           | not null |         | plain   |              |
 datetime | timestamp without time zone |           | not null |         | plain   |              |
Indexes:
    "fki_prepares_eid_fkey" btree (eid)
    "fki_preparpes_oid_fkey" btree (oid)
Foreign-key constraints:
    "prepares_eid_fkey" FOREIGN KEY (eid) REFERENCES employee(id)
    "preparpes_oid_fkey" FOREIGN KEY (oid) REFERENCES orders(id)
```

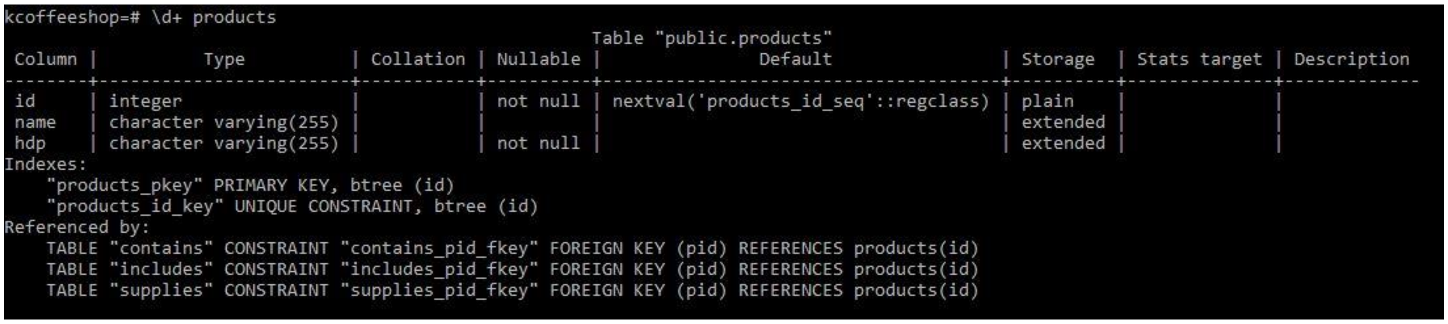
SELECT * FROM prepares;



```
kcoffeeshop=# select * from prepares;
 eid | oid |      datetime
-----+-----+---------------------
   6 |   1 | 2018-09-27 23:03:00
   3 |   2 | 2018-09-28 01:03:00
   7 |   4 | 2018-09-28 01:05:00
  22 |   3 | 2018-09-28 01:07:00
  27 |   5 | 2018-09-28 01:11:00
  30 |   7 | 2018-09-28 01:13:00
  27 |   6 | 2018-09-28 01:16:00
  28 |   9 | 2018-09-28 01:20:00
  16 |   8 | 2018-09-28 01:21:00
   6 |  12 | 2018-09-28 01:26:00
   3 |  10 | 2018-09-28 01:27:00
   7 |  11 | 2018-09-28 01:28:00
  22 |  13 | 2018-09-28 01:28:00
  27 |  14 | 2018-09-28 01:31:00
  30 |  16 | 2018-09-28 01:34:00
  27 |  15 | 2018-09-28 01:35:00
  28 |  17 | 2018-09-28 01:38:00
  16 |  18 | 2018-09-28 01:38:00
   6 |  20 | 2018-09-28 01:41:00
   3 |  19 | 2018-09-28 01:42:00
   7 |  21 | 2018-09-28 01:43:00
  22 |  23 | 2018-09-28 01:46:00
  27 |  22 | 2018-09-28 01:50:00
  30 |  24 | 2018-09-28 01:52:00
  27 |  25 | 2018-09-28 01:56:00
  28 |  27 | 2018-09-28 02:00:00
  16 |  26 | 2018-09-28 02:05:00
   3 |  29 | 2018-09-28 02:06:00
   7 |  28 | 2018-09-28 02:10:00
  22 |  30 | 2018-09-28 02:11:00
  27 |  31 | 2018-09-28 03:07:00
  30 |  33 | 2018-09-28 03:11:00
  27 |  32 | 2018-09-28 03:13:00
  28 |  34 | 2018-09-28 03:16:00
  16 |  36 | 2018-09-28 03:20:00
   3 |  35 | 2018-09-28 03:21:00
   7 |  37 | 2018-09-28 04:22:00
  22 |  38 | 2018-09-28 04:28:00
  27 |  39 | 2018-09-28 04:31:00
  30 |  40 | 2018-09-28 04:34:00
  27 |  42 | 2018-09-29 01:06:00
  28 |  41 | 2018-09-29 01:07:00
  16 |  43 | 2018-09-29 01:10:00
  27 |  44 | 2018-09-29 01:21:00
  30 |  46 | 2018-09-29 01:28:00
  27 |  45 | 2018-09-29 01:30:00
  28 |  47 | 2018-09-29 01:32:00
  16 |  48 | 2018-09-29 01:33:00
   3 |  49 | 2018-09-29 01:36:00
(49 rows)
```

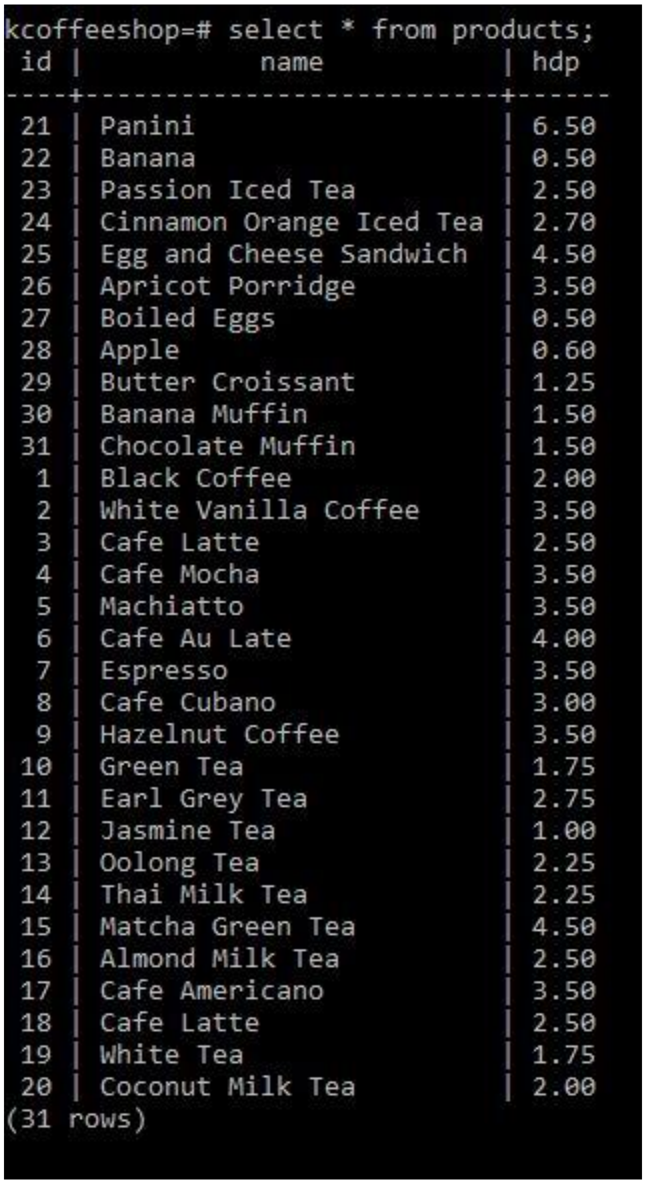
Relation- Products

\d+ Products



```
kcoffeeshop=# \d+ products
                                                           Table "public.products"
 Column |          Type          | Collation | Nullable |                 Default                  | Storage  | Stats target | Description
--------+------------------------+-----------+----------+------------------------------------------+----------+--------------+-------------
 id     | integer                |           | not null | nextval('products_id_seq'::regclass) | plain    |              |
 name   | character varying(255) |           |          |                                          | extended |              |
 hdp    | character varying(255) |           | not null |                                          | extended |              |
Indexes:
    "products_pkey" PRIMARY KEY, btree (id)
    "products_id_key" UNIQUE CONSTRAINT, btree (id)
Referenced by:
    TABLE "contains" CONSTRAINT "contains_pid_fkey" FOREIGN KEY (pid) REFERENCES products(id)
    TABLE "includes" CONSTRAINT "includes_pid_fkey" FOREIGN KEY (pid) REFERENCES products(id)
    TABLE "supplies" CONSTRAINT "supplies_pid_fkey" FOREIGN KEY (pid) REFERENCES products(id)
```

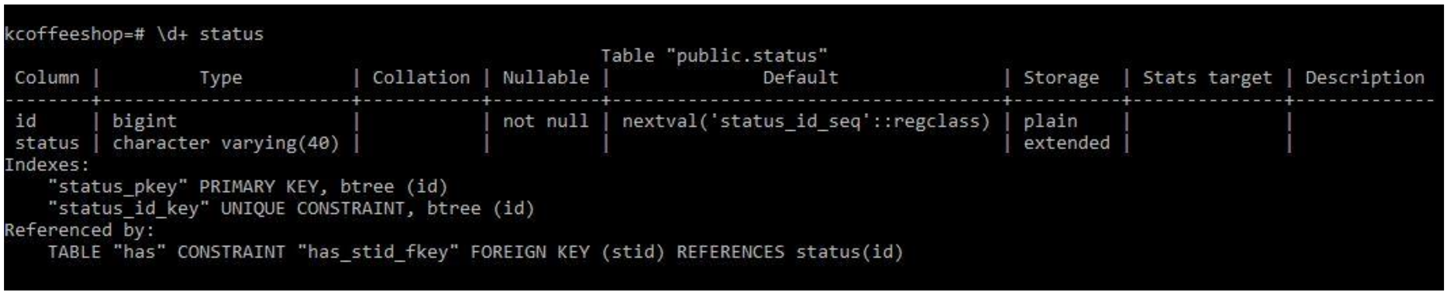
SELECT * FROM products;



```
kcoffeeshop=# select * from products;
 id |            name            | hdp
----+----------------------------+------
 21 | Panini                     | 6.50
 22 | Banana                     | 0.50
 23 | Passion Iced Tea           | 2.50
 24 | Cinnamon Orange Iced Tea   | 2.70
 25 | Egg and Cheese Sandwich    | 4.50
 26 | Apricot Porridge           | 3.50
 27 | Boiled Eggs                | 0.50
 28 | Apple                      | 0.60
 29 | Butter Croissant           | 1.25
 30 | Banana Muffin              | 1.50
 31 | Chocolate Muffin           | 1.50
  1 | Black Coffee               | 2.00
  2 | White Vanilla Coffee       | 3.50
  3 | Cafe Latte                 | 2.50
  4 | Cafe Mocha                 | 3.50
  5 | Machiatto                  | 3.50
  6 | Cafe Au Late               | 4.00
  7 | Espresso                   | 3.50
  8 | Cafe Cubano                | 3.00
  9 | Hazelnut Coffee            | 3.50
 10 | Green Tea                  | 1.75
 11 | Earl Grey Tea              | 2.75
 12 | Jasmine Tea                | 1.00
 13 | Oolong Tea                 | 2.25
 14 | Thai Milk Tea              | 2.25
 15 | Matcha Green Tea           | 4.50
 16 | Almond Milk Tea            | 2.50
 17 | Cafe Americano             | 3.50
 18 | Cafe Latte                 | 2.50
 19 | White Tea                  | 1.75
 20 | Coconut Milk Tea           | 2.00
(31 rows)
```

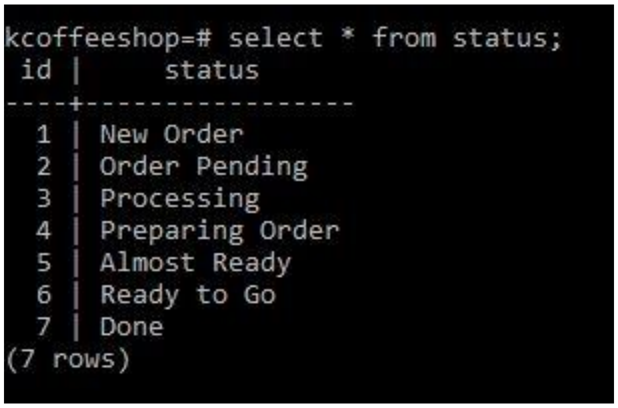
Relation- Status

\d+ Status



```
kcoffeeshop=# \d+ status
                                                    Table "public.status"
 Column |          Type          | Collation | Nullable |                Default                 | Storage  | Stats target | Description
--------+------------------------+-----------+----------+----------------------------------------+----------+--------------+-------------
 id     | bigint                 |           | not null | nextval('status_id_seq'::regclass)     | plain    |              |
 status | character varying(40)  |           |          |                                        | extended |              |
Indexes:
    "status_pkey" PRIMARY KEY, btree (id)
    "status_id_key" UNIQUE CONSTRAINT, btree (id)
Referenced by:
    TABLE "has" CONSTRAINT "has_stid_fkey" FOREIGN KEY (stid) REFERENCES status(id)
```

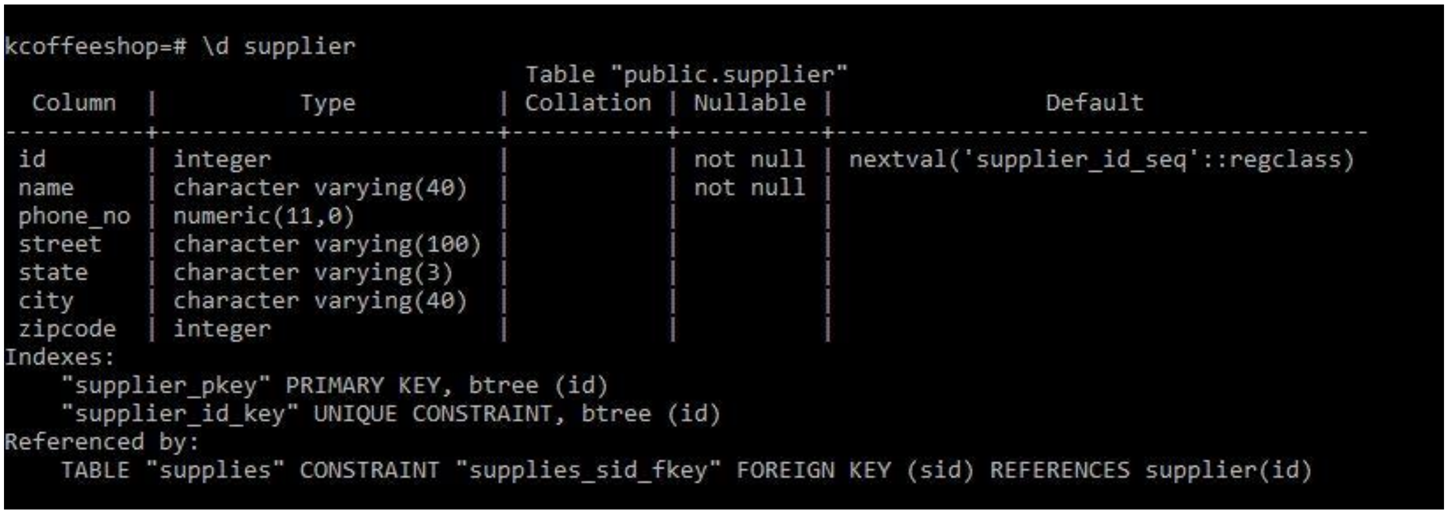
SELECT * FROM status;



```
kcoffeeshop=# select * from status;
 id |      status
----+------------------
  1 | New Order
  2 | Order Pending
  3 | Processing
  4 | Preparing Order
  5 | Almost Ready
  6 | Ready to Go
  7 | Done
(7 rows)
```

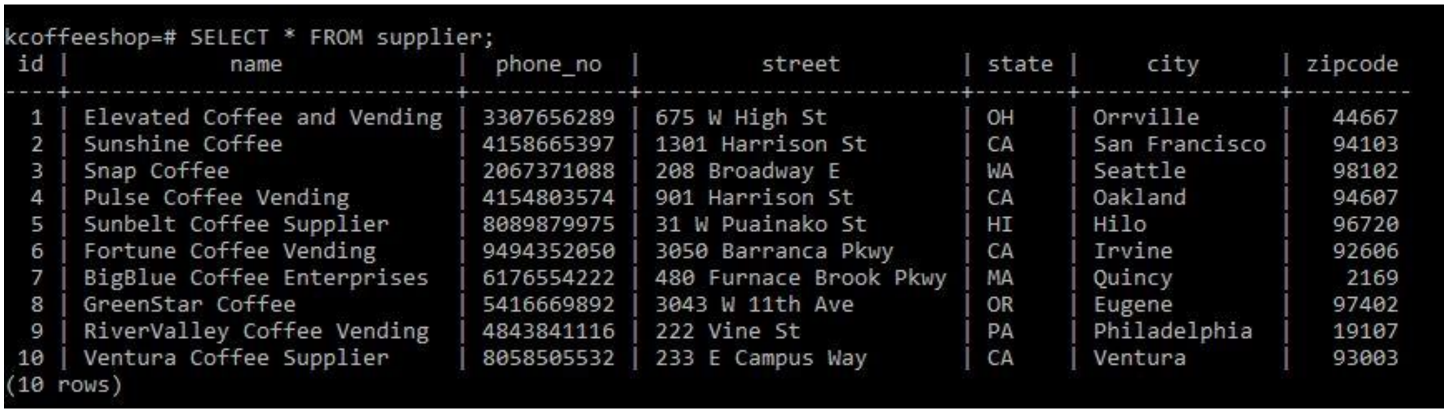
Relation- Supplier

\d supplier



```
kcoffeeshop=# \d supplier
                                      Table "public.supplier"
  Column  |          Type          | Collation | Nullable |                Default
----------+------------------------+-----------+----------+---------------------------------------
 id       | integer                |           | not null | nextval('supplier_id_seq'::regclass)
 name     | character varying(40)  |           | not null |
 phone_no | numeric(11,0)          |           |          |
 street   | character varying(100) |           |          |
 state    | character varying(3)   |           |          |
 city     | character varying(40)  |           |          |
 zipcode  | integer                |           |          |
Indexes:
    "supplier_pkey" PRIMARY KEY, btree (id)
    "supplier_id_key" UNIQUE CONSTRAINT, btree (id)
Referenced by:
    TABLE "supplies" CONSTRAINT "supplies_sid_fkey" FOREIGN KEY (sid) REFERENCES supplier(id)
```

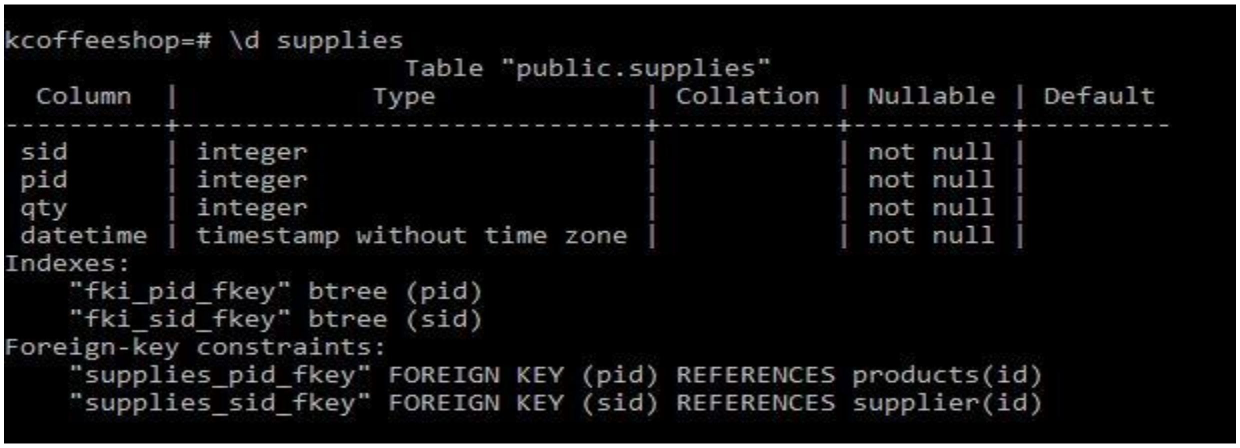
SELECT * FROM supplier;



```
kcoffeeshop=# SELECT * FROM supplier;
 id |             name             |  phone_no  |          street          | state |      city       | zipcode
----+------------------------------+------------+--------------------------+-------+-----------------+---------
  1 | Elevated Coffee and Vending  | 3307656289 | 675 W High St            | OH    | Orrville        |   44667
  2 | Sunshine Coffee              | 4158665397 | 1301 Harrison St         | CA    | San Francisco   |   94103
  3 | Snap Coffee                  | 2067371088 | 208 Broadway E           | WA    | Seattle         |   98102
  4 | Pulse Coffee Vending         | 4154803574 | 901 Harrison St          | CA    | Oakland         |   94607
  5 | Sunbelt Coffee Supplier      | 8089879975 | 31 W Puainako St         | HI    | Hilo            |   96720
  6 | Fortune Coffee Vending       | 9494352050 | 3050 Barranca Pkwy       | CA    | Irvine          |   92606
  7 | BigBlue Coffee Enterprises   | 6176554222 | 480 Furnace Brook Pkwy   | MA    | Quincy          |    2169
  8 | GreenStar Coffee             | 5416669892 | 3043 W 11th Ave          | OR    | Eugene          |   97402
  9 | RiverValley Coffee Vending   | 4843841116 | 222 Vine St              | PA    | Philadelphia    |   19107
 10 | Ventura Coffee Supplier      | 8058505532 | 233 E Campus Way         | CA    | Ventura         |   93003
(10 rows)
```

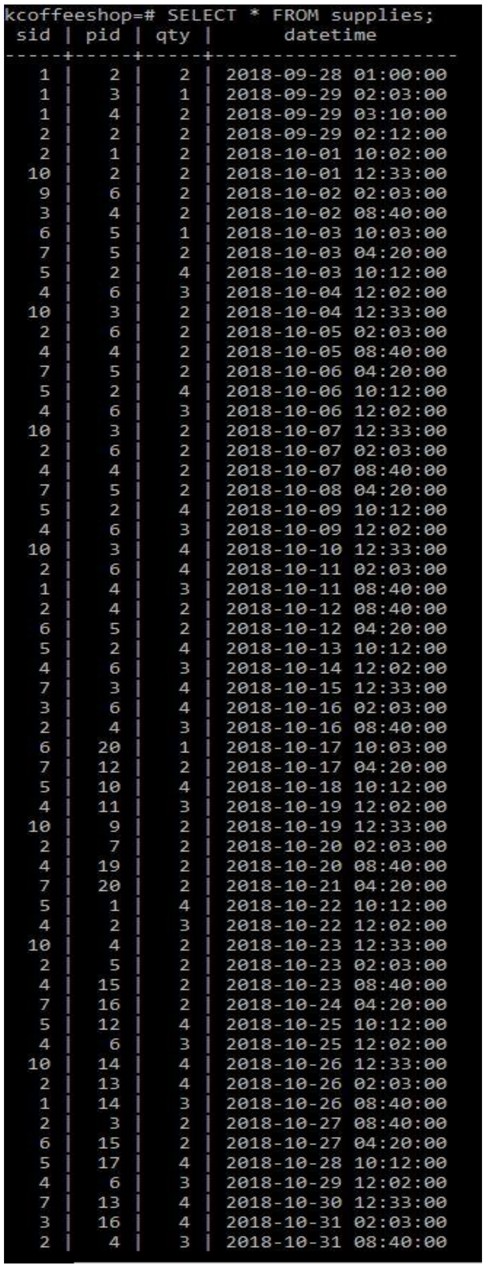
Relation- Supplies

\d supplies

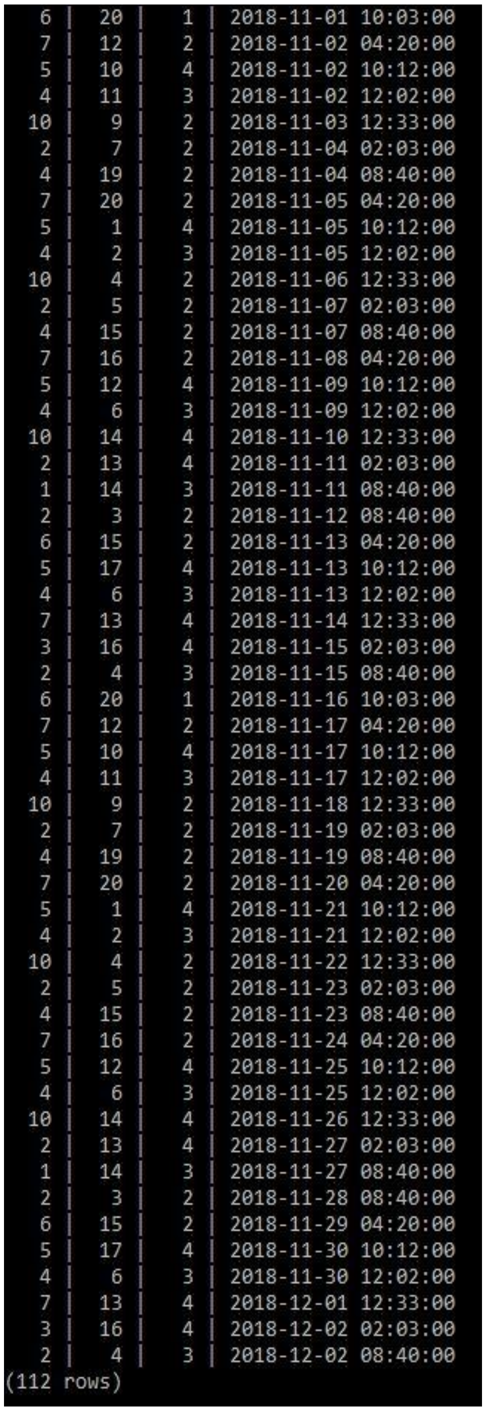


```
kcoffeeshop=# \d supplies
                          Table "public.supplies"
  Column  |            Type             | Collation | Nullable | Default
----------+-----------------------------+-----------+----------+---------
 sid      | integer                     |           | not null |
 pid      | integer                     |           | not null |
 qty      | integer                     |           | not null |
 datetime | timestamp without time zone |           | not null |
Indexes:
    "fki_pid_fkey" btree (pid)
    "fki_sid_fkey" btree (sid)
Foreign-key constraints:
    "supplies_pid_fkey" FOREIGN KEY (pid) REFERENCES products(id)
    "supplies_sid_fkey" FOREIGN KEY (sid) REFERENCES supplier(id)
```

SELECT * FROM supplies;



```
kcoffeeshop=# SELECT * FROM supplies;
 sid | pid | qty |      datetime
-----+-----+-----+---------------------
   1 |   2 |   2 | 2018-09-28 01:00:00
   1 |   3 |   1 | 2018-09-29 02:03:00
   1 |   4 |   2 | 2018-09-29 03:10:00
   2 |   2 |   2 | 2018-09-29 02:12:00
   2 |   1 |   2 | 2018-10-01 10:02:00
  10 |   2 |   2 | 2018-10-01 12:33:00
   9 |   6 |   2 | 2018-10-02 02:03:00
   3 |   4 |   2 | 2018-10-02 08:40:00
   6 |   5 |   1 | 2018-10-03 10:03:00
   7 |   5 |   2 | 2018-10-03 04:20:00
   5 |   2 |   4 | 2018-10-03 10:12:00
   4 |   6 |   3 | 2018-10-04 12:02:00
  10 |   3 |   2 | 2018-10-04 12:33:00
   2 |   6 |   2 | 2018-10-05 02:03:00
   4 |   4 |   2 | 2018-10-05 08:40:00
   7 |   5 |   2 | 2018-10-06 04:20:00
   5 |   2 |   4 | 2018-10-06 10:12:00
   4 |   6 |   3 | 2018-10-06 12:02:00
  10 |   3 |   2 | 2018-10-07 12:33:00
   2 |   6 |   2 | 2018-10-07 02:03:00
   4 |   4 |   2 | 2018-10-07 08:40:00
   7 |   5 |   2 | 2018-10-08 04:20:00
   5 |   2 |   4 | 2018-10-09 10:12:00
   4 |   6 |   3 | 2018-10-09 12:02:00
  10 |   3 |   4 | 2018-10-10 12:33:00
   2 |   6 |   4 | 2018-10-11 02:03:00
   1 |   4 |   3 | 2018-10-11 08:40:00
   2 |   4 |   2 | 2018-10-12 08:40:00
   6 |   5 |   2 | 2018-10-12 04:20:00
   5 |   2 |   4 | 2018-10-13 10:12:00
   4 |   6 |   3 | 2018-10-14 12:02:00
   7 |   3 |   4 | 2018-10-15 12:33:00
   3 |   6 |   4 | 2018-10-16 02:03:00
   2 |   4 |   3 | 2018-10-16 08:40:00
   6 |  20 |   1 | 2018-10-17 10:03:00
   7 |  12 |   2 | 2018-10-17 04:20:00
   5 |  10 |   4 | 2018-10-18 10:12:00
   4 |  11 |   3 | 2018-10-19 12:02:00
  10 |   9 |   2 | 2018-10-19 12:33:00
   2 |   7 |   2 | 2018-10-20 02:03:00
   4 |  19 |   2 | 2018-10-20 08:40:00
   7 |  20 |   2 | 2018-10-21 04:20:00
   5 |   1 |   4 | 2018-10-22 10:12:00
   4 |   2 |   3 | 2018-10-22 12:02:00
  10 |   4 |   2 | 2018-10-23 12:33:00
   2 |   5 |   2 | 2018-10-23 02:03:00
   4 |  15 |   2 | 2018-10-23 08:40:00
   7 |  16 |   2 | 2018-10-24 04:20:00
   5 |  12 |   4 | 2018-10-25 10:12:00
   4 |   6 |   3 | 2018-10-25 12:02:00
  10 |  14 |   4 | 2018-10-26 12:33:00
   2 |  13 |   4 | 2018-10-26 02:03:00
   1 |  14 |   3 | 2018-10-26 08:40:00
   2 |   3 |   2 | 2018-10-27 08:40:00
   6 |  15 |   2 | 2018-10-27 04:20:00
   5 |  17 |   4 | 2018-10-28 10:12:00
   4 |   6 |   3 | 2018-10-29 12:02:00
   7 |  13 |   4 | 2018-10-30 12:33:00
   3 |  16 |   4 | 2018-10-31 02:03:00
   2 |   4 |   3 | 2018-10-31 08:40:00
```



```
   6 |  20 |   1 | 2018-11-01 10:03:00
   7 |  12 |   2 | 2018-11-02 04:20:00
   5 |  10 |   4 | 2018-11-02 10:12:00
   4 |  11 |   3 | 2018-11-02 12:02:00
  10 |   9 |   2 | 2018-11-03 12:33:00
   2 |   7 |   2 | 2018-11-04 02:03:00
   4 |  19 |   2 | 2018-11-04 08:40:00
   7 |  20 |   2 | 2018-11-05 04:20:00
   5 |   1 |   4 | 2018-11-05 10:12:00
   4 |   2 |   3 | 2018-11-05 12:02:00
  10 |   4 |   2 | 2018-11-06 12:33:00
   2 |   5 |   2 | 2018-11-07 02:03:00
   4 |  15 |   2 | 2018-11-07 08:40:00
   7 |  16 |   2 | 2018-11-08 04:20:00
   5 |  12 |   4 | 2018-11-09 10:12:00
   4 |   6 |   3 | 2018-11-09 12:02:00
  10 |  14 |   4 | 2018-11-10 12:33:00
   2 |  13 |   4 | 2018-11-11 02:03:00
   1 |  14 |   3 | 2018-11-11 08:40:00
   2 |   3 |   2 | 2018-11-12 08:40:00
   6 |  15 |   2 | 2018-11-13 04:20:00
   5 |  17 |   4 | 2018-11-13 10:12:00
   4 |   6 |   3 | 2018-11-13 12:02:00
   7 |  13 |   4 | 2018-11-14 12:33:00
   3 |  16 |   4 | 2018-11-15 02:03:00
   2 |   4 |   3 | 2018-11-15 08:40:00
   6 |  20 |   1 | 2018-11-16 10:03:00
   7 |  12 |   2 | 2018-11-17 04:20:00
   5 |  10 |   4 | 2018-11-17 10:12:00
   4 |  11 |   3 | 2018-11-17 12:02:00
  10 |   9 |   2 | 2018-11-18 12:33:00
   2 |   7 |   2 | 2018-11-19 02:03:00
   4 |  19 |   2 | 2018-11-19 08:40:00
   7 |  20 |   2 | 2018-11-20 04:20:00
   5 |   1 |   4 | 2018-11-21 10:12:00
   4 |   2 |   3 | 2018-11-21 12:02:00
  10 |   4 |   2 | 2018-11-22 12:33:00
   2 |   5 |   2 | 2018-11-23 02:03:00
   4 |  15 |   2 | 2018-11-23 08:40:00
   7 |  16 |   2 | 2018-11-24 04:20:00
   5 |  12 |   4 | 2018-11-25 10:12:00
   4 |   6 |   3 | 2018-11-25 12:02:00
  10 |  14 |   4 | 2018-11-26 12:33:00
   2 |  13 |   4 | 2018-11-27 02:03:00
   1 |  14 |   3 | 2018-11-27 08:40:00
   2 |   3 |   2 | 2018-11-28 08:40:00
   6 |  15 |   2 | 2018-11-29 04:20:00
   5 |  17 |   4 | 2018-11-30 10:12:00
   4 |   6 |   3 | 2018-11-30 12:02:00
   7 |  13 |   4 | 2018-12-01 12:33:00
   3 |  16 |   4 | 2018-12-02 02:03:00
   2 |   4 |   3 | 2018-12-02 08:40:00
(112 rows)
```

3.5 Examples of Queries written in SQL Language

1. **List all orders to fetch the product name ‘Black Coffee’ with the total price more than 3.00.**

```
CREATE OR REPLACE VIEW q1 AS
        SELECT DISTINCT o.id, o.payment
        FROM orders as o
        INNER JOIN contains as c
        ON c.oid=o.id
        INNER JOIN products as p
        ON p.id = c.pid
        AND p.name = 'Black Coffee'
        AND c.sprice > 3.00;
```

Output:

```
 kcoffeeshop=# select * from q1;

 id | payment
----+---------
  2 | cash
(1 row)
```

2. **List all orders to fetch the product name 'Black Coffee' with the sale price less than 3.00.**

```
CREATE OR REPLACE VIEW q2 AS
        SELECT DISTINCT supplier.name, supplies.datetime, supplies.qty, products.name
        AS products
        FROM supplier
        INNER JOIN supplies ON supplier.id = supplies.sid
        INNER JOIN products ON products.id = supplies.pid
        WHERE supplies.qty < 2
        LIMIT 5;
```

Output:

```
kcoffeeshop=# select * from q2;


          name                  |      datetime        | qty |   products
--------------------------------+----------------------+-----+------------------
 Elevated Coffee and Vending    | 2018-09-29 02:03:00  |   1 | Cafe Latte
 Fortune Coffee Vending         | 2018-10-03 10:03:00  |   1 | Macchiato
 Fortune Coffee Vending         | 2018-10-17 10:03:00  |   1 | Coconut Milk Tea
 Fortune Coffee Vending         | 2018-11-01 10:03:00  |   1 | Coconut Milk Tea
 Fortune Coffee Vending         | 2018-11-16 10:03:00  |   1 | Coconut Milk Tea
```

3. List the customer's first and last name with a complete status from the customer's placed order.

```
        CREATE OR REPLACE VIEW q4 AS
                        SELECT DISTINCT o.id, c.fname AS cust_fname, c.lname AS cust_lname, o.payment,
                o.datetime, o.total, s.status
                        FROM orders o
                        INNER JOIN has h
                        ON h.soid=o.id
                        INNER JOIN status s
                        ON s.id=h.stid
                        INNER JOIN customer c
                        ON c.id=o.cid
                        WHERE h.datetime > '2018-09-28 00:00:00'
                        AND EXISTS (
                            SELECT d.status
                            FROM status d
                            WHERE d.id = h.stid
                            AND d.status = 'Done')
                        GROUP BY o.id, s.id, c.fname, c.lname
                        ORDER BY o.id, c.fname
                        LIMIT 10;
```

Output:

```
 kcoffeeshop=# select * from q4;
 Id  | cust_fname | cust_lname | payment |      datetime       | total | status
----+-----------------+-----------------+------------+---------------------------+-------+--------
  1  | Julie       | Dihn        | card    | 2018-09-27 23:00:00 |  8.00 | Done
  2  | Samori      | Price       | cash    | 2018-09-28 01:00:00 |  2.00 | Done
  4  | Ellen       | Ijebor      | card    | 2018-09-28 01:02:00 |  4.00 | Done
  5  | Hienny      | Tran        | card    | 2018-09-28 01:06:00 |  7.50 | Done
  6  | Sean        | Desmond     | card    | 2018-09-28 01:10:00 |  8.50 | Done
  8  | Sean        | Desmond     | card    | 2018-09-28 01:15:00 |  9.00 | Done
  9  | Delilah     | Mitchell    | card    | 2018-09-28 01:19:00 |  8.75 | Done
 11  | Ellen       | Ijebor      | card    | 2018-09-28 01:21:00 | 10.00 | Done
 12  | Donovan     | Innes       | card    | 2018-09-28 01:25:00 |  5.75 | Done
 13  | Molly       | Gorman      | card    | 2018-09-28 01:29:00 |  8.50 | Done
(10 rows)
```

4. List the orders placed by customers prepared by the employee 'Haleigh'.

```
CREATE OR REPLACE VIEW q5 AS
        SELECT DISTINCT o.*, e.fname AS ename, c.fname AS cname, c.lname
                FROM orders o
                INNER JOIN prepares p
                ON p.oid=o.id
                INNER JOIN customer c
                ON o.cid=c.id
                INNER JOIN employee e
                ON e.id=p.eid
                WHERE p.datetime >= '2018-09-28 00:00:00'
                AND EXISTS (
                        SELECT *
                        FROM employee e2
                        WHERE e2.id = e.id
                        AND e2.fname = 'Haleigh')
        GROUP BY o.id, e.id, c.id
        ORDER BY o.id;
```

Output:

```
 kcoffeeshop=# select * from q5;

 id  | payment |     datetime           | cid | total  | ename  |  cname     | lname
----+-------------+---------------------------+-----+--------+-----------+-------------+----------
  2  | cash     | 2018-09-28 01:00:00 |  1  |  2.00  | Haleigh | Samori     | Price
 10 | card     | 2018-09-28 01:20:00 |  6  |  1.50  | Haleigh | Stephanie | Osbourne
 19 | card     | 2018-09-28 01:39:00 | 16  |  8.20  | Haleigh | Hue-An    | Tran
 29 | card     | 2018-09-28 02:09:00 | 14  | 10.75 | Haleigh | Hienny    | Tran
 35 | card     | 2018-09-28 03:15:00 |  7  |  9.25  | Haleigh | Sean       | Desmond
 49 | card     | 2018-09-29 01:35:00 |  9  |  3.75  | Haleigh | Kitty       | Vo
(6 rows)
```

5. List the customers who placed an order once.

```
CREATE OR REPLACE VIEW q6 AS
    SELECT DISTINCT c.id, c.fname AS customer_name, c.lname AS customer_lname
    FROM customer c
        WHERE EXISTS (
        SELECT *
        FROM orders o
        WHERE o.cid = c.id AND
            NOT EXISTS (
                SELECT *
                FROM orders o2
                WHERE o2.cid = c.id
                AND o.datetime != o2.datetime));
```

Output:

```
kcoffeeshop=# select * from q6;
 Id  | customer_name | customer_lname
----+----------------------+---------------
 11 | Miller               | Ramirez
 12 | Anthony              | Delgado
 20 | Alejandra            | Flores
(3 rows)
```

6. List the role with the least number of employees.

```
CREATE OR REPLACE VIEW q7 AS
    SELECT DISTINCT e.id, e.fname, e.lname, e.role
        From employee e
        WHERE NOT EXISTS(
            SELECT * FROM employee e2
            WHERE
            e.id != e2.id AND e.role < e2.role);
```

Output:

```
kcoffeeshop=# select * from q7;
 id | fname  | lname  |  role
----+---------+-----------+---------
 15 | Boigie | Markos | Manager
(1 row)
```

7. List the last and first name of customers who have not placed an order.

```
CREATE OR REPLACE VIEW q12 AS
    SELECT c.lname, c.fname
        FROM customer c
        WHERE NOT EXISTS (
            SELECT *
            FROM orders o
            WHERE c.id = o.cid);
```

Output:

```
kcoffeeshop=# select * from q12;
 lname | fname
---------+-------
 Sweyd | Zack
(1 row)
```

8. List the first and last name of customers who at least more than 10.00 on their order.

```
        CREATE OR REPLACE VIEW q13 AS
            SELECT c.fname, c.lname
                FROM customer c
                WHERE EXISTS (
                    SELECT *
                    FROM orders o
                    WHERE o.cid=c.id
                    AND total >= 10.00)
            ORDER BY c.fname, c.lname;
```

Output:

```
kcoffeeshop=# select * from q13;
  fname   | lname
------------+---------
 Alejandra | Flores
 Ellen     | Ijebor
 Hienny    | Tran
 Kelly     | Moua
 Kitty     | Vo
 Kyle      | MacLean
 Molly     | Gorman
 Samori    | Price
 Xavier    | Robles
(9 rows)
```

9. List the customer's first and last name who ordered products 'Cafe Americano' and 'Apple' or from the category 'Snacks'.

```
        CREATE OR REPLACE VIEW q10 AS
                SELECT DISTINCT cu.fname AS cust_fname, cu.lname AS cust_lname, o.payment,
        o.datetime, p.name AS product_name , c.name AS category_name, o.total
                FROM customer cu
                INNER JOIN orders o
                ON cu.id=o.cid
                 INNER JOIN contains co
                ON co.oid=o.id
                 INNER JOIN products p
                ON p.id=co.pid
                INNER JOIN includes i
                ON i.pid=p.id
                INNER JOIN category c
                 ON c.id=i.catid
                 WHERE(p.name='Cafe Americano' OR c.name='Snacks' AND p.name='Apple')
                AND o.total < 8.00
        ORDER BY c.name;
```

Output:

```
kcoffeeshop=# select * from q10;
 cust_fname | cust_lname | payment |      datetime       | product_name   | category_name | total
------------+------------+---------+---------------------+----------------+---------------+-------
 Julie      | Dihn       | card    | 2018-09-29 01:20:00 | Cafe Americano | Cold Coffee   |  3.50
 Sean       | Desmond    | card    | 2018-09-28 03:10:00 | Cafe Americano | Cold Coffee   |  7.50
 Julie      | Dihn       | card    | 2018-09-29 01:20:00 | Cafe Americano | Hot Coffee    |  3.50
 Sean       | Desmond    | card    | 2018-09-28 03:10:00 | Cafe Americano | Hot Coffee    |  7.50
 Ellen      | Ijebor     | card    | 2018-09-28 01:02:00 | Apple          | Snacks        |  4.00
 Samori     | Price      | card    | 2018-09-28 01:02:00 | Apple          | Snacks        |  7.50
(6 rows)
```

10. List the suppliers who provided a less than one of products that cost more than 2.00 dollars.

```
        CREATE OR REPLACE VIEW q11 AS
                SELECT DISTINCT s.id, s.name AS supplier_name, su.qty, p.hdp, p.name AS
        product_name
                 FROM supplier s
                INNER JOIN supplies su
                ON s.id=su.sid
                INNER JOIN products p
                ON su.pid=p.id
                        AND NOT EXISTS(
                                SELECT su2.datetime
                                FROM supplies su2
                                WHERE su2.datetime!=su.datetime
                                AND su2.qty<su.qty)
                                        AND p.hdp > 2.00
                                        AND EXISTS (
                                                SELECT p.hdp
                                                FROM products p2
                                                WHERE p2.hdp != p.hdp)
                ORDER BY s.id, su.qty, p.name;
```

Output:

```
kcoffeeshop=# select * from q11;
 id |       supplier_name                   | qty | hdp  | product_name
----+-----------------------------------------+-----+------+-------------
  1 | Elevated Coffee and Vending      |  1   | 2.50 | Cafe Latte
  6 | Fortune Coffee Vending              |  1   | 3.50 | Macchiato
(2 rows)
```

<u>Additional Queries:</u>

11. For all orders, retrieve the total number of orders from '2018-09-28 04:00:00' to now.

```
CREATE OR REPLACE VIEW q3 AS
        SELECT DATE(datetime) AS date,
        COUNT(total) AS count
        FROM orders
       WHERE datetime
       BETWEEN '2018-09-28 00:00:00'
       AND NOW()
       GROUP BY date
       ORDER BY date;
```

Output:

```
 kcoffeeshop=# select * from q3;

   date       | count
----------------+-------
 2018-09-28 |   39
 2018-09-29 |    9
(2 rows)
```

12. List the sum of prices from the orders.

```
CREATE OR REPLACE VIEW q8 AS
    SELECT o.*, SUM(c.sprice) AS "Sum of Prices"
     FROM contains c
    INNER JOIN orders o
    ON o.id=c.oid
    WHERE id=1
    GROUP BY c.sprice, o.id
    ORDER BY c.sprice, o.id;
```

Output:

```
kcoffeeshop=# select * from q8;
 id | payment |      datetime          | cid | total  | Sum of Prices
----+------------+---------------------------+-----+-------+--------------
  1 | card      | 2018-09-27 23:00:00 |   4 |  8.00 |         0.50
  1 | card      | 2018-09-27 23:00:00 |   4 |  8.00 |         3.00
  1 | card      | 2018-09-27 23:00:00 |   4 |  8.00 |         4.50
(3 rows)
```

13. List the customer's details in order number 5 with the order total of products.

```
    CREATE OR REPLACE VIEW q9 AS
        SELECT DISTINCT SUM(total), pay, custf, custl
            FROM (
            SELECT o.payment AS pay, cu.fname AS custf, cu.lname AS custl,
        SUM(c.sprice) AS total
            FROM contains c
             INNER JOIN orders o
            ON o.id=c.oid
            INNER JOIN customer cu
            ON cu.id=o.cid
            INNER JOIN products p
             ON c.pid=p.id
            WHERE c.oid=5
             GROUP BY c.sprice, o.id, p.id, cu.id
            ORDER BY c.sprice, o.id, p.id) AS totals
        GROUP BY pay, custf, custl;
```

Output:

```
kcoffeeshop=# select * from q9;
 sum  | pay  | custf  | custl
------+------+--------+-------
 7.50 | card | Hienny | Tran
(1 row)
```

3.6 Data Loader

The following section goes over the data loading methods in the PostgreSQL database server using different formats to import and export data from the database. The section includes a description of the Data Loader Program and additional features added into the program to make it more user friendly. This consisted of using different methods to insert data.

The first method of data loading is using the pgAdmin tool. After creating a table, the table can be restored by providing a path to the database file.

Sample using pgAdmin tool for data loading:

```
C:\temp\table.tar
```

After providing the path, the restore button is clicked and seconds later they give the option to click on the done button to restore the database. Then, a verification of the loaded sample database is required where the table is opened in the object browser panel and a a list of tables are located in the schema, including other database objects.

The other method to data loading is using COPY to load all of the rows in one command instead of using a series of INSERT commands. The command is optimized for loading a large amount of rows and is less adjustable than INSERT, but contains less overhead when it comes to loading large amount of data. Therefore, COPY is the fastest method for data loading.

Sample of using COPY for data loading:

```
COPY myTable FROM '/path/to/file/on/server' ( FORMAT CSV, DELIMITER('|') );
```

PgLoader on the other hand can load data from the CSV and is written as the following configuration file as:

Sample of using PgLoader for data loading:

```
LOAD CSV
  FROM 'path/to/file.csv' (x, y, a, b, c, d)
  INTO postgresql:///pgloader?csv (a, b, d, c)
  …
```

Further methods of data loading includes SQL statements where they can be manually inputted to insert data into the database. In the following, the SQL statement to load data is INSERT.

Insert Statements:
The postgreSQL INSERT INTO statement is utilized to insert new data into a table. The INSERT INTO statement can be expressed in two forms.

Example 1:
The first form allows to specify only the values where the data will be inserted, not their column names.

INSERT INTO *<table_name>*
VALUES *(<expression_1>, <expression_2>, … <expression_n>);*

Example II:
The second form specifies both the columns and the corresponding values that are inserted in the column.

INSERT INTO table_name (<column_1>, <column_2>, … <column_n>)
VALUES (<expression_1>, <expression_2>, … <expression_n>);

Phase 4: DBMS Procedural Language & Stored Procedures and Triggers

The previous phases consisted of building a conceptual database model for the purpose of converting it as a logical model. The last phase consisted of implementing the logical model and normalizing the relational database. The implementation of the logical model created relational schemas that can be displayed, retrieved and queried on psql. The following phase involves the DBMS procedural language of PostgreSQL with the stored procedures and triggers.

The first section of the phase involves gaining a depth understanding of the Postgres PL/pgSQL language and stored procedures, functions, triggers and packages, including their purpose and benefits. The second section of the phase consists of coding requirements by creating three stored procedures and triggers to implement and test in the Postgres PL/pgSQL subprograms.

The last section of the phase contains a description of PL/pgSQL-like languages in Microsoft SQL Server, MySQL, and Oracle DBMS. The gathering of information is achieved by using the internet to find further details over procedural languages for the three DBMS. The last step of the section is comparing and contrasting of the three DBMS containing the syntax of each language comprising of selective statements, repetitive statements, and statements for creating stored procedures/functions/triggers.

4.1 Postgres PL/ pgSQL

The Postgres PL/pgSQL is a loadable procedural language for the PostgreSQL database system. The first objective of PL/pgSQL consists of creating functions and trigger procedures. The second objective of PL/pgSQL is allowing more procedural control than SQL that gives it the ability to use loops and additional control structures. The third objective of PL/pgSQL is allow users to perform complex operations and computations than SQL, aimed to be easy to use. The procedural language differs from SQL functions since PL/pgSQL includes features such as variables, conditional evaluation, and looping allowing more procedural control than SQL functions allow only argument substitution. The user-defined functions developed in PL/PgSQL can be utilized like any built-in function. In addition, PL/pgSQL is safe and is trusted by the server.

The PostgreSQL stored procedures classifies functions for creating triggers and aggregate functions. Stored procedures contains additional procedural features consisting of control structures and complex calculations that allows the user to develop custom functions that are both easy and effective. The advantages using stored procedures is due to its reusability, separation of duties, data integrity, and event handling. The store produce's reusability avoids rewriting subqueries while improving readability. For instance, if a user is unable to store a query for a library where all applications are accessed, the query can be in stored procedure. The separation of duties serves to secure and grants minimal privileges for users and administrators by restricting full authority to a non-DBA to prevent internal threats and fraud. The data integrity uses triggers and constraints to prevent defective data from entering, including running several interdependent queries in a transaction in a single stored procedure. The event handling contains log changes and notifying other system of new incoming data.

The disadvantages of stored procedures consists of slowing down software development process, software rollouts that require more database changes, challenging to version control stored procedures. The stored should be used since views may be all that the user needs, and object-relational mappers (ORM) helps write queries safely. In comparison of using stored procedures and SQL query, stored procedures can improve performance, reduce network traffic, and avoids implementing statements into every application that communicates with the database. In addition, another advantage of stored procedures compared to SQL consists of allowing users to embed business login as an API in the database, which simplifies data management and minimizes encoding logic in client programs. This results to faster, reliable, and less corrupted of data in frontend/client software development and ensures data integrity and consistency when implementing stored procedures.

4.1.1 PL/pgSQL Program Structure, Control Statements and Cursors

The following subsection includes the PL/pgSQL program structure, control statement, and cursors. The structures and statements is briefly described and includes their general syntax.

PL/pgSQL Program Structure

The PL/pgSQL procedural language is a block-structured language where each statement is terminated by a semicolon. The blocks consists of sequence of statements containing DECLARE/BEGIN and END statements. Block can be nested and must end with END, including a semi-color after END.

Block Syntax:

```
[<<label>>]
[DECLARE
        declarations]
BEGIN
```

```
        statements
END [ label ];
```

PL/pgSQL Control Structures

The PL/pgSQL control structures helps manipulate PostgreSQL data and is the most useful part of PL/pgSQL. The control structures consists of conditional statements and additional types of functions for loops, while loops, and basic loops. The following will provide the syntax of the statements.

Conditional Statements

The conditional statement involves the "IF", case, and loop statements. The "IF" statement executes a command conditionally while a case statement allows to execute a block of code conditionally and the loops occasionally need to execute a block of statements until a condition turns out true. There exist three forms of the "IF" statements, two forms of case statements, and various forms of loops. The following will provide examples of the "IF", case, and loop statement syntax.

Three Forms of IF Statements

IF-THEN-END IF Syntax:

```
IF boolean-expression THEN
        statements
END IF;
```

IF-THEN-ELSE-END IF Syntax:

```
IF boolean-expression THEN
        statements
ELSE
        statements
END IF;
```

IF-THEN-ELSIF Syntax:

```
IF boolean-expression THEN
   statements
[ ELSIF boolean-expression THEN
   statements
[ ELSIF boolean-expression THEN
   statements
```

```
  ...]]
[ ELSE
  statements ]
END IF;
```

Two forms of Case Statements

Simple Case Syntax:

```
CASE search-expression
  WHEN expression [, expression [ ... ]] THEN
   statements
 [ WHEN expression [, expression [ ... ]] THEN
   statements
  ... ]
 [ ELSE
   statements ]
END CASE;
```

Searched Case Syntax:

```
CASE
  WHEN boolean-expression THEN
   statements
 [ WHEN boolean-expression THEN
   statements
  ... ]
 [ ELSE
   statements ]
END CASE;
```

Forms of Loop Statements

Simple Loop:

```
[ <<label>> ]
LOOP
  statements
END LOOP [ label ];
```

While Loop:

```
[ <<label>> ]
WHILE boolean-expression LOOP
   statements
END LOOP [ label ];
```

For Loop [Integer Variant]:

```
[ <<label>> ]
FOR name IN [ REVERSE ] expression .. expression [ BY expression ] LOOP
   statements
END LOOP [ label ];
```

Cursors

Cursors encapsulates queries and then the query result is generated a few rows at a time. This is to avoid memory overrun when the result contains a large number of rows. The cursors are used to divide a large row set into parts and process each part individually. Processing it all at once runs into the problem of memory overflow error. Furthermore, a function can be developed to return a reference to a cursor to provide efficiency in returning a large row set from a function where the called of the function can process the result set based on the cursor reference. The steps of using a cursor in PL/pgSQL starts with declaring a cursor, opening a cursor, fetching rows from the result set into a target, checking if there is more row left to fetch, and closing the cursor. The following displays the syntax in declaring, opening, fetching, and closing a cursor.

Declaring Cursor Syntax :

```
name [ [ NO ] SCROLL ] CURSOR [ ( arguments ) ] FOR query;
```

Opening Cursor Syntax:

```
OPEN unbound_cursorvar [ [ NO ] SCROLL ] FOR query;
```

Fetching Cursor Syntax:

```
FETCH [ direction { FROM | IN } ] cursor INTO target;
```

Closing Cursor Syntax:

```
CLOSE cursor;
```

4.1.2 Stored Procedures

The PL/pgSQL stored procedures and stored functions are used interchangeably since both are generally the same. The stored procedures are defined as functions for creating triggers and customer aggregate functions. Similarly to stored functions, stored procedures add procedural features such as control structure and complex calculations. Since stored procedures and stored functions are used interchangeably, a broader scope of stored procedures is explained in stored functions.

Stored Procedure/Function Syntax:

```
CREATE OR REPLACE FUNCTION function_name(p1 type, p2 type)
        RETURNS type AS $$
                BEGIN
                -- logic
                END;
        $$ LANGUAGE language_name;
```

4.1.3 Stored Functions

The PL/pgSQL support stored functions but does not support stored procedures. The stored functions and stored procedures are used interchangeably where functions are created with the FUNCTION keyword and procedures are created with the PROCEDURE keyword. The stored procedures do not return a value whereas a stored function returns a single value that can be used in the SELECT statements. Since PL/pgSQL supports stored functions, the syntax allows for the functions to act procedure-like. Furthermore, in PL/pgSQL the stored functions creates triggers or custom aggregate functions and adds many procedural features such as control structures and complex calculations which makes the functions easier and effective. The advantage of PL/pgSQL stored functions prevails on reducing the number of round trips between application and database servers, as well as increasing the application performance because of the user-defined function that are pre-compiled and stored in the PL/pgSQL database server, including that they can be reused in many applications. The disadvantages of using PL/pgSQL stored functions consists of slowing the software developed since it requires specialized skills that many developers do not have, as well as making it difficult to manage versions and hard to debug, and it may not be portable to other DBMS. The following will display the basic stored functions syntax.

Stored Procedure/Function Syntax:

```
CREATE OR REPLACE FUNCTION function_name(p1 type, p2 type)
```

```
RETURNS type AS $$
          BEGIN
          -- logic
          END;
$$ LANGUAGE language_name;
```

4.1.4 Packages

The packages in PL/pgSQL do not exist, instead uses schemas to organize the functions into groups. Considering that packages are non-existent, package-level variables are also non-existent. Though there may not be packages or package-level, schemas are used to organize functions into groups.

4.1.5 Triggers

The PL/pgSQL trigger is a user-defined function binded to a table where trigger functions are created and defined to bind with the table when it is activated by a certain event . The event consists of the INSERT, UPDATE, DELETE and TRUNCATE where trigger functions can invoke before or after an event. The trigger implemented before an event can skip the operation for the current row and can change the row that is updated or inserted. Furthermore, triggers are practical in databases for the purpose of enforcing business rules, validating input data, generating unique values in new inserted rows of a different file, containing audit trails, querying from different files, duplicating data to other files, and obtaining access to system functions. The benefits of using triggers in business involves of quicker app development, large-squale enforcement of business rules, simpler maintenance, and improved performance of client/server environment. The trigger is created through the CREATE TRIGGER statement with the trigger function syntax.

Trigger Function Syntax:

```
CREATE [ CONSTRAINT ] TRIGGER name { BEFORE | AFTER | INSTEAD OF } { event [
OR ... ] }
   ON table_name
   [ FROM referenced_table_name ]
   [ NOT DEFERRABLE | [ DEFERRABLE ] { INITIALLY IMMEDIATE | INITIALLY
DEFERRED } ]
   [ FOR [ EACH ] { ROW | STATEMENT } ]
   [ WHEN ( condition ) ]
   EXECUTE PROCEDURE function_name ( arguments )
```

4.2 PostgreSQL PL/pgSQL Subprogram Examples

The following section presents subprograms in PL/pgSQL. The method used for group functions is through the schema in which focuses on implementing procedures, functions, and triggers. The section does not include packages since in PostgreSQL, packages do not exist.

Postgres Stored Function-INSERT Description:

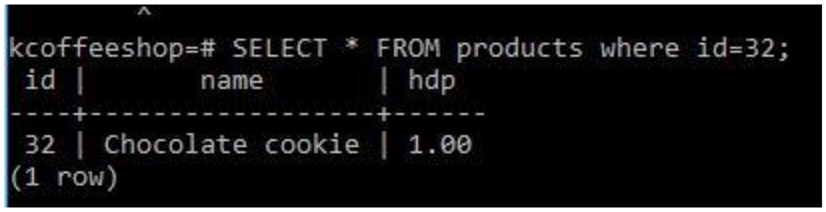
The INSERT stored procedure is utilized to create new products and insert data in the parameters. The following creates the 'insert_product' stored procedure.

```
CREATE OR REPLACE FUNCTION insert_product(id integer, name character varying(255), hpd numeric(10,2))
            RETURNS void AS $$
                BEGIN
                INSERT INTO products VALUES (id, name, hpd);
            END
$$ LANGUAGE 'plpgsql';
```

Postgres Stored Function-INSERT Usage:

```
SELECT public.insert_product(32, 'Chocolate cookie', 1.00)
```

Postgres Stored Function-INSERT Output:



```
kcoffeeshop=# SELECT * FROM products where id=32;
 id |       name       | hdp
----+------------------+------
 32 | Chocolate cookie | 1.00
(1 row)
```

Postgres Trigger Function:

Before creating the trigger, a separate table had to be created to store the event (insert, delete, or update) that occurs in the employee table called *backup_emp_tbl*. The following trigger invokes 'before insert, delete or update operation' by inserting the old data into the backup_emp_tbl.

```
CREATE OR REPLACE FUNCTION ins_function() RETURNS trigger AS '
BEGIN
  IF tg_op = ''DELETE'' THEN
     INSERT INTO backup_emp_tbl(id, fname, mname, lname, role, email, operation)
     VALUES (old.id, old.fname, old.mname, old.lname, old.role, old.email, tg_op);
     RETURN old;
  END IF;
  IF tg_op = ''INSERT'' THEN
     INSERT INTO backup_emp_tbl(id, fname, mname, lname, role, email, operation)
     VALUES (new.id, new.fname, new.mname, new.lname, new.role, new.email, tg_op);
     RETURN new;
  END IF;
  IF tg_op = ''UPDATE'' THEN
     INSERT INTO backup_emp_tbl(id, fname, mname, lname, role, email, operation)
     VALUES (old.id, old.fname, old.mname, old.lname, old.role, old.email, tg_op);
     RETURN new;
  END IF;
END;
' LANGUAGE plpgsql;
```

Postgres Trigger:

The following trigger invokes the function 'ins_function' after insert, delete, or update operation.

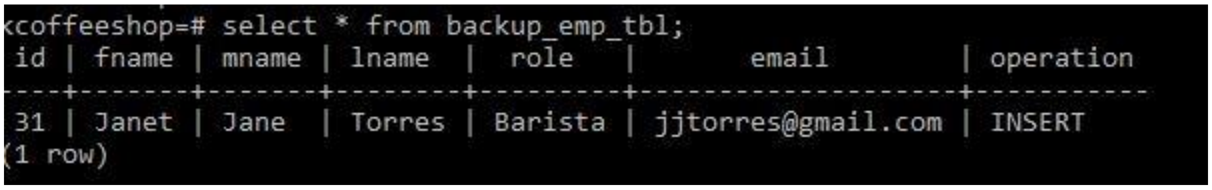
```
CREATE TRIGGER audit_ins AFTER INSERT OR DELETE OR UPDATE
        ON employee FOR each ROW
        EXECUTE PROCEDURE ins_function();
```

Postgres Trigger-INSERT Usage:

Inserting data will be stored to the backup_emp_tbl where the trigger logic is applied every time the insert function is invoked.

```
INSERT INTO employee (id, fname, mname, lname, role, email) values (31, 'Janet', 'Jane', 'Torres', 'Barista', 'jjtorres@gmail.com');
```

Postgres Trigger-INSERT Output:



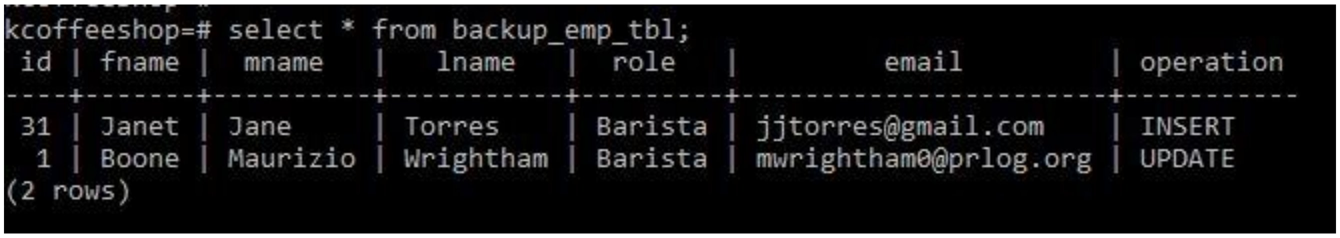
```
kcoffeeshop=# select * from backup_emp_tbl;
 id | fname | mname | lname  |  role   |        email         | operation
----+-------+-------+--------+---------+----------------------+-----------
 31 | Janet | Jane  | Torres | Barista | jjtorres@gmail.com   | INSERT
(1 row)
```

Postgres Trigger-UPDATE Usage:

Updating data in the employee table will insert the old data to the backup_emp_tbl as per trigger logic.

```
UPDATE employee SET role= 'Cashier' where id=1;
```

Postgres Trigger-UPDATE Output:



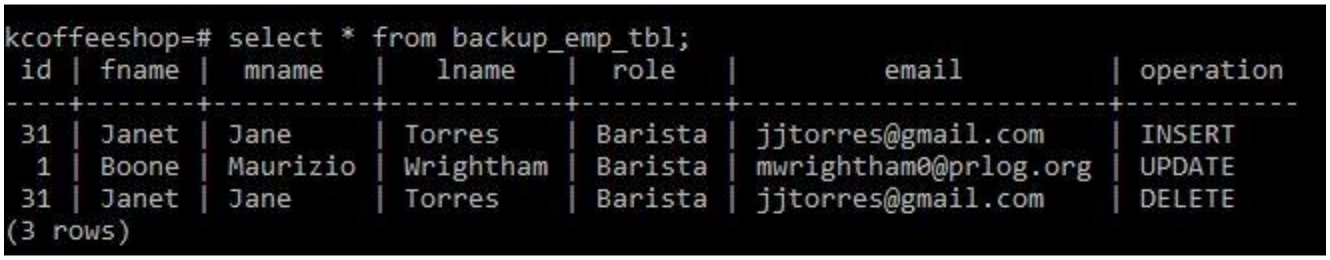
```
kcoffeeshop=# select * from backup_emp_tbl;
 id | fname |  mname   |   lname   |  role   |         email          | operation
----+-------+----------+-----------+---------+------------------------+-----------
 31 | Janet | Jane     | Torres    | Barista | jjtorres@gmail.com     | INSERT
  1 | Boone | Maurizio | Wrightham | Barista | mwrightham0@prlog.org  | UPDATE
(2 rows)
```

Postgres Trigger-DELETE Usage:

Deleting data in the employee table will insert the old data to the backup_emp_tbl as per trigger logic.

```
DELETE FROM employee WHERE id='31';
```

Postgres Trigger-DELETE Output:



```
kcoffeeshop=# select * from backup_emp_tbl;
 id | fname |  mname   |   lname   |  role   |         email          | operation
----+-------+----------+-----------+---------+------------------------+-----------
 31 | Janet | Jane     | Torres    | Barista | jjtorres@gmail.com     | INSERT
  1 | Boone | Maurizio | Wrightham | Barista | mwrightham0@prlog.org  | UPDATE
 31 | Janet | Jane     | Torres    | Barista | jjtorres@gmail.com     | DELETE
(3 rows)
```

4.3 PL/pgSQL Like Tools (Oracle, Microsoft SQL Server, and MySQL)

The PL/pgSQL is utilized in Kiwi's Coffee Shop to implement the physical database. Other database management systems exist that offer similar features to PL/pgSQL written in their own unique syntax. The following will explore and display a comparison between the PL/pgSQL like tools of Oracle PL/SQL, Microsoft SQL Server T-SQL, and MySQL and an overview to their syntax consisting selective statements, repetitive statements, and the syntax of creating stored procedures, functions, and triggers.

Microsoft SQL Server T-SQL

Comparison to other commercial DBMS Languages:

The Microsoft T-SQL (Transact-SQL) is a proprietary procedural language extension from Microsoft that adds features to SQL, including transaction control, error handling and exception, row processing and declared variables. PL/SQL and T-SQL share many similarities, however there are differences between the two RDBMS since they use different languages that comprises of differences within their syntax. The second difference between the two languages is how they handle variables, stored procedures, and built-in functions.

While PL/SQL can group procedures together into packages, T-SQL does not. The third difference between the two languages is transaction control where they are defined as a group of operations treated as a single unit. The drawback is T-SQL Server executes and commits each command individually making it difficult to roll back changes if errors were to occur during the process, while PL/SQL has changes made only in the memory and nothing is committed until an explicit commit statement is prompted that helps with error control. The last difference between PL/SQL and T-SQL is the organization of RDBMS database objects, where T-SQL organizes all objects such as table, view, and procedures by their database names. PL/SQL organizes the database objects by grouped schemas that are subset collections of database objects to share within the all database schemas.

In comparison with T-SQL and MySQL, MySQL does not support common table expression with "with" statements. T-SQL is case sensitive with schema names, whereas MySQL is not. The next comparison comprises of multiple storage making MySQL more flexible than T-SQL. In query execution, T-SQL allows for database queries shorten during execution without killing the entire process while MySQL does not allow users to kill or cancel a query when it is running and requires to kill the entire process to stop execution. Moreover, the following provides the syntax commonly used in Microsoft SQL Server T-SQL.

IF Statement Syntax:

```
IF Boolean_expression
    { sql_statement | statement_block }
[ ELSE
    { sql_statement | statement_block } ]
```

Switch Statement Syntax:

```
CASE input_expression
    WHEN when_expression THEN result_expression [ ...n ]
    [ ELSE else_result_expression ]
END
Searched CASE expression:
CASE
    WHEN Boolean_expression THEN result_expression [ ...n ]
    [ ELSE else_result_expression ]
END
```

While Statement Syntax:

```
WHILE Boolean_expression
    { sql_statement | statement_block | BREAK | CONTINUE }
```

Creating a Stored Procedure Syntax:

```
CREATE [ OR ALTER ] { PROC | PROCEDURE }
   [schema_name.] procedure_name [ ; number ]
   [ { @parameter [ type_schema_name. ] data_type }
      [ VARYING ] [ = default ] [ OUT | OUTPUT | [READONLY]
   ] [ ,...n ]
[ WITH <procedure_option> [ ,...n ] ]
[ FOR REPLICATION ]
AS { [ BEGIN ] sql_statement [;] [ ...n ] [ END ] }
[;]
```

```
<procedure_option> ::=
   [ ENCRYPTION ]
   [ RECOMPILE ]
   [ EXECUTE AS Clause ]
```

Creating Functions Syntax:

```
CREATE [ OR ALTER ] FUNCTION [ schema_name. ] function_name
( [ { @parameter_name [ AS ][ type_schema_name. ] parameter_data_type
   [ = default ] [ READONLY ] }
   [ ,...n ]
  ]
)
RETURNS return_data_type
   [ WITH <function_option> [ ,...n ] ]
   [ AS ]
   BEGIN
      function_body
      RETURN scalar_expression
   END
[ ; ]
```

Creating a Trigger Syntax:

```
CREATE [ OR ALTER ] TRIGGER [ schema_name . ]trigger_name
ON { table | view }
[ WITH <dml_trigger_option> [ ,...n ] ]
{ FOR | AFTER | INSTEAD OF }
{ [ INSERT ] [ , ] [ UPDATE ] [ , ] [ DELETE ] }
[ WITH APPEND ]
[ NOT FOR REPLICATION ]
AS { sql_statement  [ ; ] [ ,...n ] | EXTERNAL NAME <method specifier [ ; ] > }

<dml_trigger_option> ::=
   [ ENCRYPTION ]
   [ EXECUTE AS Clause ]
```

```
<method_specifier> ::=
  assembly_name.class_name.method_name
```

MySQL

Comparison to other commercial DBMS Languages:

MySQL is an open-source RDBMS (relational database management system) and is the most popular RDBMS used to develop web-based software applications. In comparison, T-SQL and MySQL are similar since both are relational database management systems and are programmed in both C++ and C. The differences between MySQL and T-SQL involves that MySQL is only available in English language while T-SQL is available in multiple languages. MySQL supports many platforms whereas T-SQL supports only Linux and Windows platforms only. Furthermore, MySQL is open source while T-SQL is commercial. The syntax in MySQL is more complex than T-SQL, since T-SQL's syntax has a simple ease of use. MySQL is used mainly in PHP applications where T-SQL is primarily used in .Net applications or Windows applications. In addition, MySQL contains multiple storage engines that gives developers more flexibility to use the engine for tables based on performance whereas T-SQL only uses a single storage engine. Once a query is executed on MySQL, the process of querying cannot be cancelled immediately, whereas T-SQL has the flexibility of having it's query process cancelled midway.

Moreover, Oracle is an object-relational database management system (ORDBMS) whereas MySQL is an open-source relational database management system. In terms of cost, Oracle is licensed but can get the express edition for free and comes with very limited functionalities and it only used for educational and testing purposes, whereas MySQL is free and licensed under the General Public License. The scalability in Oracle is used in large scale deployments, whereas in MySQL is used for small and big businesses. Oracle support stored procedure that is embedded within the database, whereas stored procedures within MySQL was not supported until version 5. The customizability in MySQL is that it can be easily modified by the programmer to suit their environment based on requirements, whereas in Oracle it is not customizable. Oracle supports data partitioning, whereas in MySQL does not support data partitions and requires a server for each set of data files. In security, Oracle requires for a username, password to login whereas MySQL requires only the username, password, and host. In the end, MySQL, Oracle, and T-SQL hold their differences, but choosing the language depends on certain specifications.

IF Statement Syntax:

```
IF search_condition THEN statement_list
   [ELSEIF search_condition THEN statement_list] ...
   [ELSE statement_list]
END IF
```

Switch Statement Syntax:

```
CASE case_value
   WHEN when_value THEN statement_list
   [WHEN when_value THEN statement_list] ...
   [ELSE statement_list]
END CASE
```

While Loop Syntax:

```
[begin_label:] WHILE search_condition DO
   statement_list
END WHILE [end_label]
```

Simple Loop Syntax:

```
[begin_label:] LOOP
   statement_list
END LOOP [end_label]
```

Creating Stored Procedures Syntax:

```
CREATE
   [DEFINER = { user | CURRENT_USER }]
   PROCEDURE sp_name ([proc_parameter[,...]])
   [characteristic ...] routine_body
```

Creating a Trigger Syntax:

```
CREATE
   [DEFINER = { user | CURRENT_USER }]
   TRIGGER trigger_name
   trigger_time trigger_event
   ON tbl_name FOR EACH ROW
```

```
    [trigger_order]
    trigger_body

trigger_time: { BEFORE | AFTER }

trigger_event: { INSERT | UPDATE | DELETE }

trigger_order: { FOLLOWS | PRECEDES } other_trigger_name
```

Create a Function Syntax:

```
CREATE
    [DEFINER = { user | CURRENT_USER }]
    FUNCTION sp_name ([func_parameter[,...]])
    RETURNS type
    [characteristic ...] routine_body
```

Oracle PL/SQL

Comparison to other commercial DBMS Languages:

The Oracle PL/SQL is a procedural language extension of Structure Query Language (SQL) use in Oracle. The PL/SQL is partially integrated with SQL with programming constructs that are not inherent of SQL. Furthermore, Oracle is a widely used RDBMS, whereas T-SQL is limitedly used in Linux and Windows. The languages that support Oracle extends to many languages whereas T-SQL is limitedly supported by few languages. In addition, T-SQL and Oracle are convenient for creating full applications. In both RDBMS, migrating code from one dialect to the other is nontrivial. In Oracle, each database connection is treated as a new transaction, whereas changes and rollbacks are allowed as values and do not change before being inputted, in comparison cannot be done in T-SQL. Moreover, T-SQL executes each command separately and makes it difficult to make changes if errors are encountered during the process.

Moreover, Oracle and MySQL was previously compared and distinguished. However, they both have their advantages. One of these advantages deals that MySQL is free to use, performs well, is user-friendly, scalable, works with many operating systems, and supports many development interfaces. On the other hand, Oracle's advantages is very feature rich, highly reliable, and has flashback technologies.

IF Syntax:

```
IF condition THEN
  {...statements to execute when condition is TRUE...}
END IF;
```

Switch (CASE) Statement Syntax:

```
CASE [ expression ]

  WHEN condition_1 THEN result_1
  WHEN condition_2 THEN result_2
  ...
  WHEN condition_n THEN result_n
  ELSE result
END
```

While Loop Syntax:

```
WHILE condition
LOOP
  {...statements...}
END LOOP;
```

For Loop Syntax:

```
FOR loop_counter IN [REVERSE] lowest_number..highest_number
LOOP
  {...statements...}
END LOOP;
```

Creating a Stored Procedure Syntax:

```
CREATE OR REPLACE PROCEDURE
<procedure_name>
        (
        <parameterl IN/OUT <datatype>
        ..
        .
        )
[ IS | AS ]
        <declaration_part>
BEGIN
        <execution part>
EXCEPTION
        <exception handling part>
END;
```

Creating a Function Syntax:

```
CREATE OR REPLACE FUNCTION
        <procedure_name>
        (
        <parameter IN/OUT <datatype>
        )
        RETURN <datatype>
        [ IS | AS ]
        <declaration_part>
        BEGIN
        <execution part>
        EXCEPTION
        <exception handling part>
        END;
```

Creating a Trigger Syntax:

```
CREATE [ OR REPLACE ] TRIGGER <trigger_name>

        [BEFORE | AFTER | INSTEAD OF ]

        [INSERT | UPDATE | DELETE......]

        ON<name of underlying object>
```

```
[FOR EACH ROW]

[WHEN<condition for trigger to get execute> ]

DECLARE
<Declaration part>
BEGIN
<Execution part>
EXCEPTION
<Exception handling part>
END;
```

Phase 5: Graphics User Interface Design and Implementation

The final phase implements a graphical user interface (GUI) application by integrating the previous phases developed for Kiwi's Coffee Shop enterprise. The phase reviews the functionalities and content for the GUI interface by displaying an overview of the application. In addition, the phase revisits the server-side programs that consists of views, stored subprograms, and triggers that the user can promptly invoke.

The second section of the phase inspects the backend, middle-tier, and front-end programming. The backend of the application uses the PostgreSQL software to store data in the database created on a specific server utilized in the enterprise. The middle-tier purpose is to integrate the back-end and front-end by displaying the data to the user and allowing the user to access the application's business services and the transaction's system-level logic. The front-end of the application consists of the user interface and graphic controls for the client by using PHP, Bootstrap, CSS, and JQuery. In addition, the section will provide examples and snippets of the code to demonstrate the development of the 3-tier application. Furthermore, the last section consist of answering survey questions over my performance and the outcome I achieved during the course.

5.1 Functionalities and User Group of the GUI Application

In the development of the front-end application for Kiwi's Coffee Shop database, the accessibility of different users was thoroughly considered. The application was customized to interact with both the customer and employee. Therefore, this required for the development and creation of different graphical user interfaces dedicated for the two types of users with a different set of functionalities.

5.1.1 Itemized Descriptions

The following provides the itemized description of graphical user-interface applications for two types of users in Kiwi's Coffee Shop.

Customer Users:

The **customer** users view the main webpage and are required to toggle the navigation bar to the categories they wish to order the products from. The customer is able to add products, cancel products, and upon checkout are required to sign up with their email address to complete their order and must provide the payment method type to fully checkout.

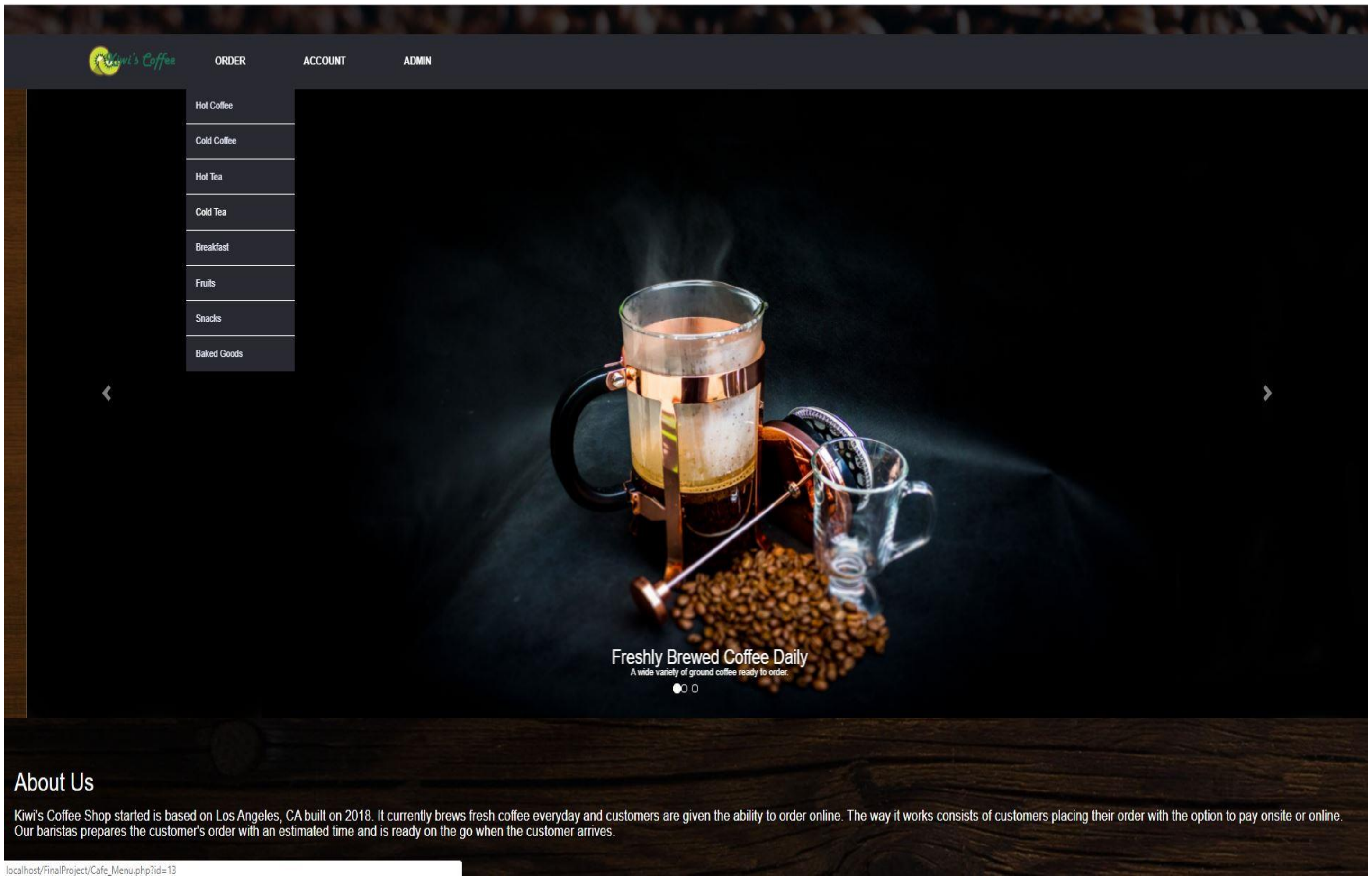
- View categories
- View products
- View product information
- View and delete products from shopping cart
- Add customer information (such as street, state, city, zip code, and phone number)
- Add payment method
- Create Order
- View and delete order (generates the ‘Cancelled’ status)

Employee Users:
The **employee** users are required to add, update, and delete properties on the administrator content of the application.

- View dashboard with reports of revenue, new customers, orders, status, and suppliers
- Manage orders and updating the status
- Add, update, or delete suppliers
- Add, update, or delete categories
- Add, update, or delete products
- Add, update, or delete employees
- Manage employees (e.g. promoting employee to new role)
- Creating an order for supplier to supply products

5.1.2 Screenshots of the Application

The following section presents screenshots of the application and a description of its functionalities. This describes the functionalities for each command button, contents of data and so on used in the application.



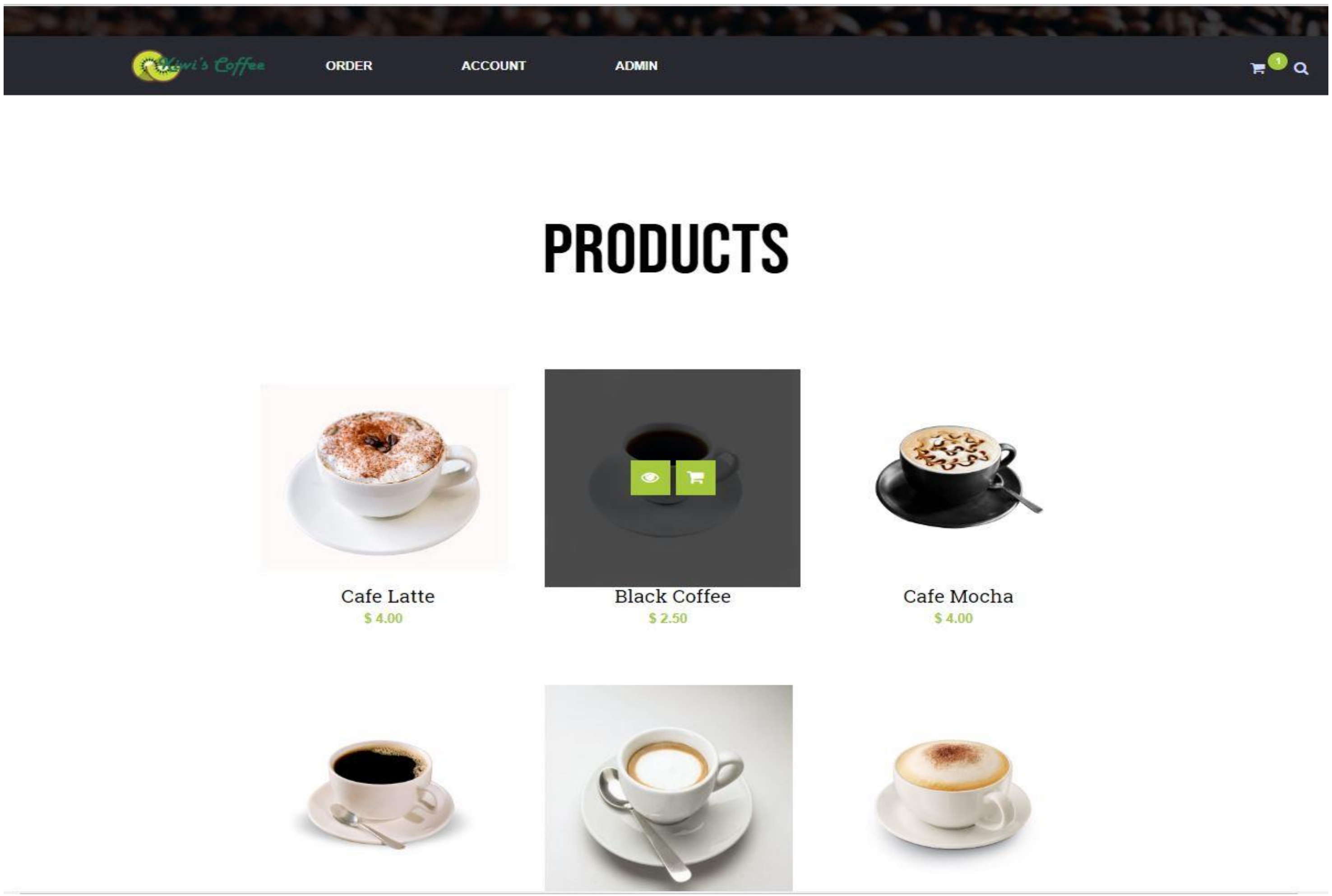
About Us

Kiwi's Coffee Shop started is based on Los Angeles, CA built on 2018. It currently brews fresh coffee everyday and customers are given the ability to order online. The way it works consists of customers placing their order with the option to pay onsite or online. Our baristas prepares the customer's order with an estimated time and is ready on the go when the customer arrives.

localhost/FinalProject/Cafe_Menu.php?id=13

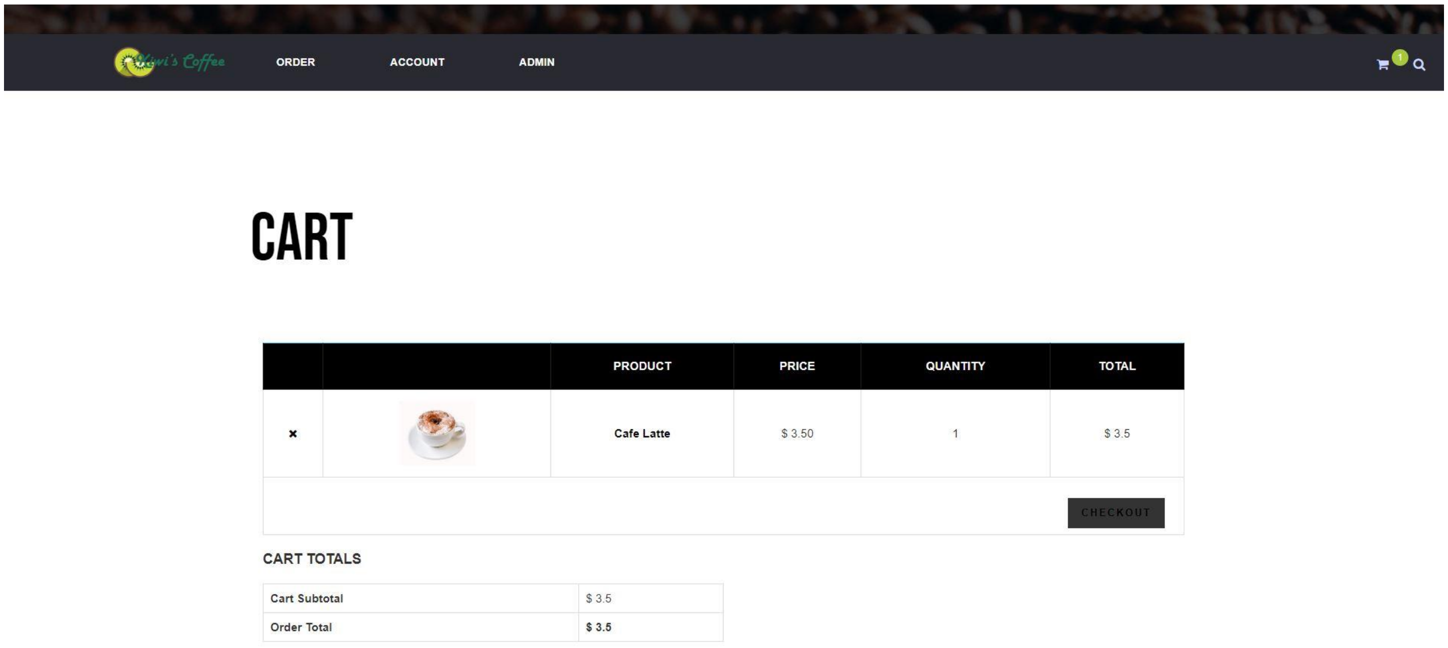
Main Page

The main page for Kiwi's Coffee Shop is the opening page where customer users and employee users are redirected. The customer users are able to point-click to 'Order' and have the option to create an account through the 'Account' tab before they order products. The 'Order' tab on the navigation bar lists a variety of categories that the customer user can choose from. Otherwise, the customer user can click on 'Order' and redirected to view all of the products without their corresponding category. Furthermore, the employee users are able to point-click to the 'Admin' and click 'Login' where they are redirected to the 'Admin' login page. In addition, the main page provides an appealing opening that users can navigate and view extra details over the enterprise.



Order Page (Products)

The products page displays all of the product contents that the customer user may be inclined to purchase. The customer is able to narrow down the selection through the navigation bar where the ‘Order’ tab is available to lists the categories. The ‘Order’ page displays all of the products and gives the customer user the ability to view the product and its details by clicking on the eye button. However, if they make their decision solely by viewing the product, they are alternatively given the option to click on the shopping cart that adds the product to their cart before completing an order.



CART

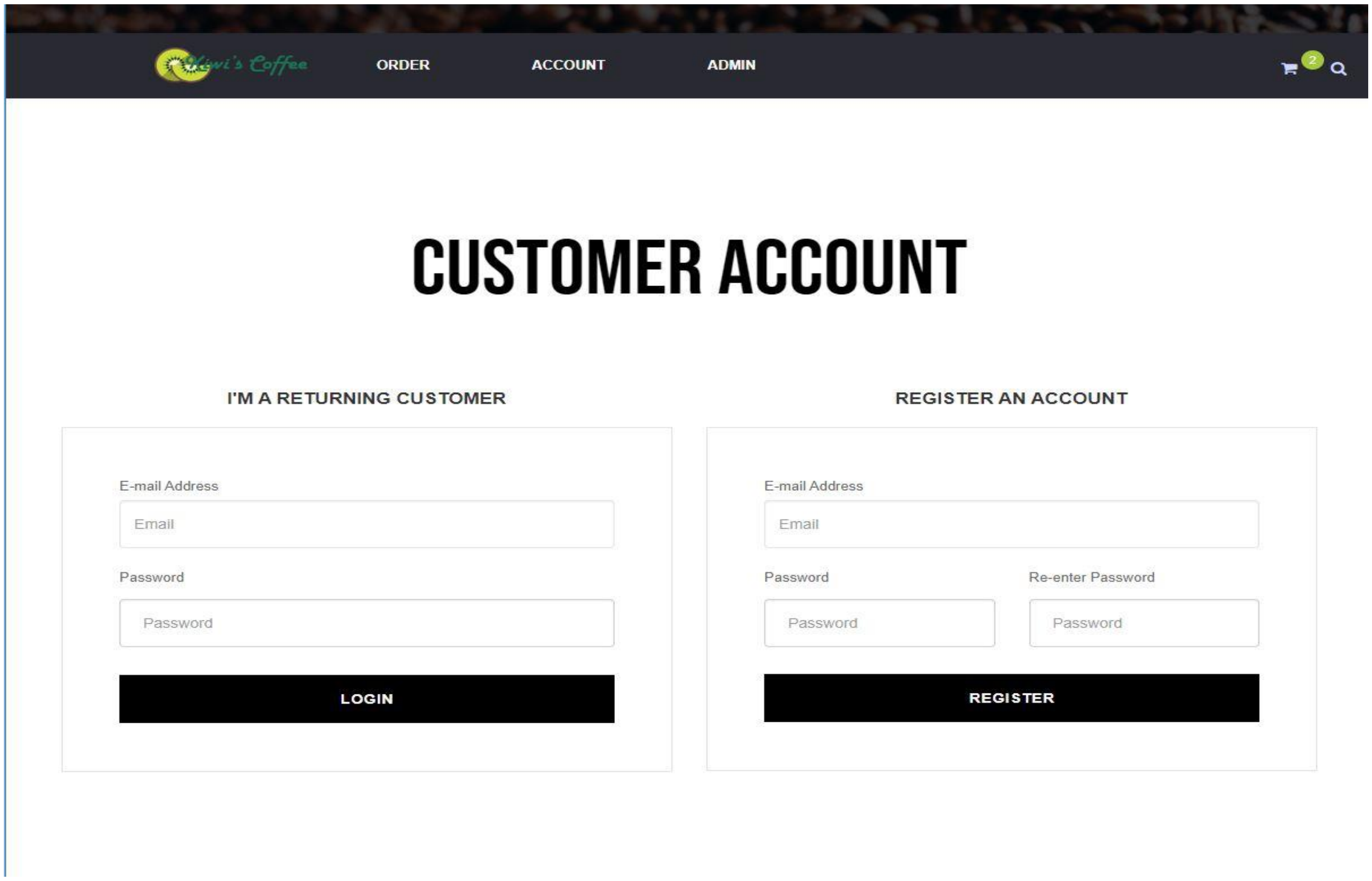
		PRODUCT	PRICE	QUANTITY	TOTAL
×		Cafe Latte	$ 3.50	1	$ 3.5

CART TOTALS

Cart Subtotal	$ 3.5
Order Total	**$ 3.5**

Cart Page

The cart page displays the product, price, quantity, and total. If the customer user is not satisfied with their current product in the cart before checking out, they are able to delete the contents of the cart through the 'x' button. Otherwise, the customer user is able to checkout from cart page by clicking on the 'Checkout' button below the product's information.



CUSTOMER ACCOUNT

I'M A RETURNING CUSTOMER

E-mail Address

Email

Password

Password

LOGIN

REGISTER AN ACCOUNT

E-mail Address

Email

Password

Password

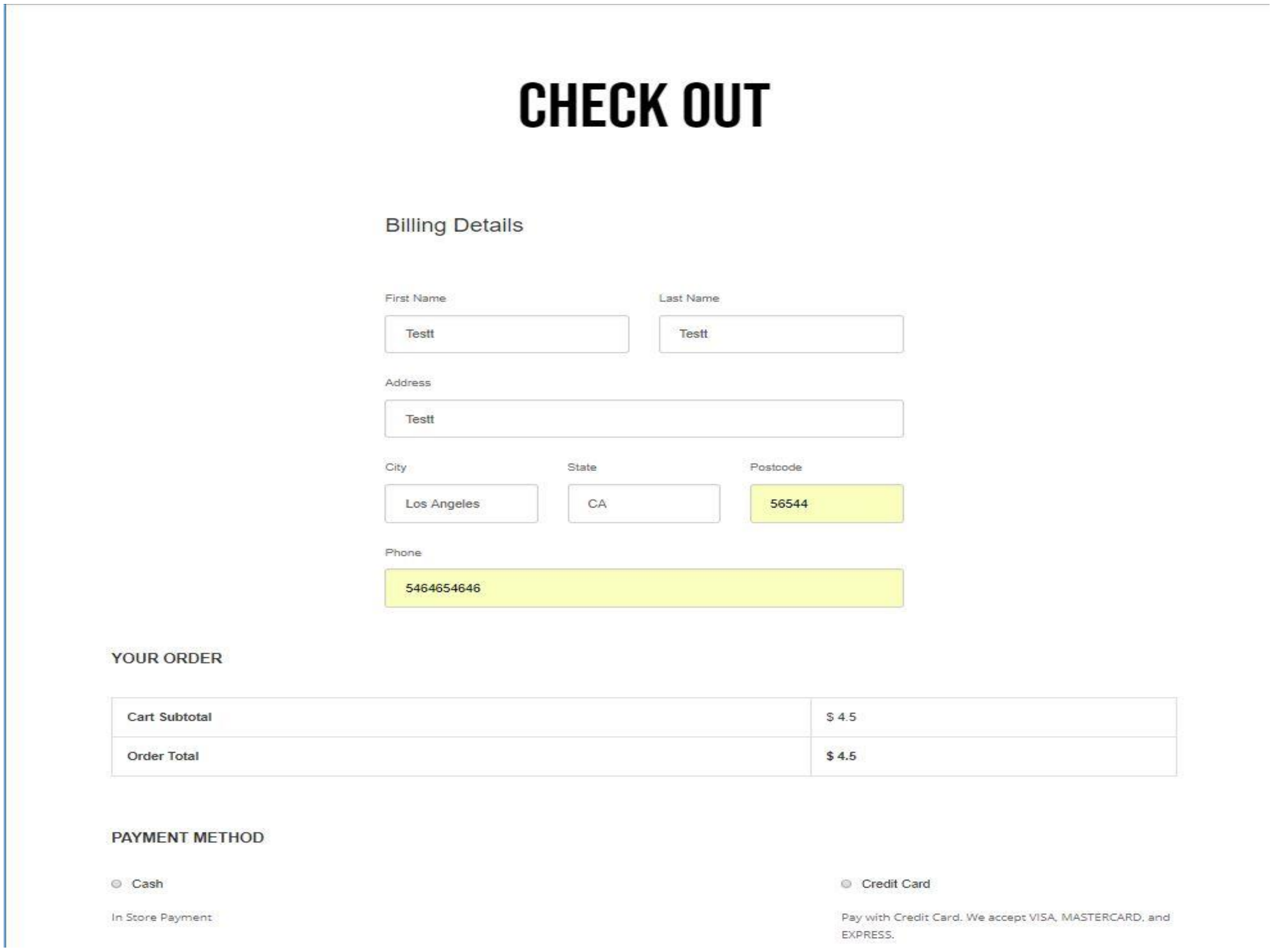
Re-enter Password

Password

REGISTER

Customer Account

The customer user is required to create an account before they checkout to keep track of their orders. If the customer user is a returning customer, the customer user is able to enter their stored information through the input boxes. Otherwise, the customer must register and enter required information such as their email address and password before gaining access to purchasing their order.



CHECK OUT

Billing Details

First Name	Testt
Last Name	Testt
Address	Testt
City	Los Angeles
State	CA
Postcode	56544
Phone	5464654646

YOUR ORDER

Cart Subtotal	$ 4.5
Order Total	$ 4.5

PAYMENT METHOD

○ Cash

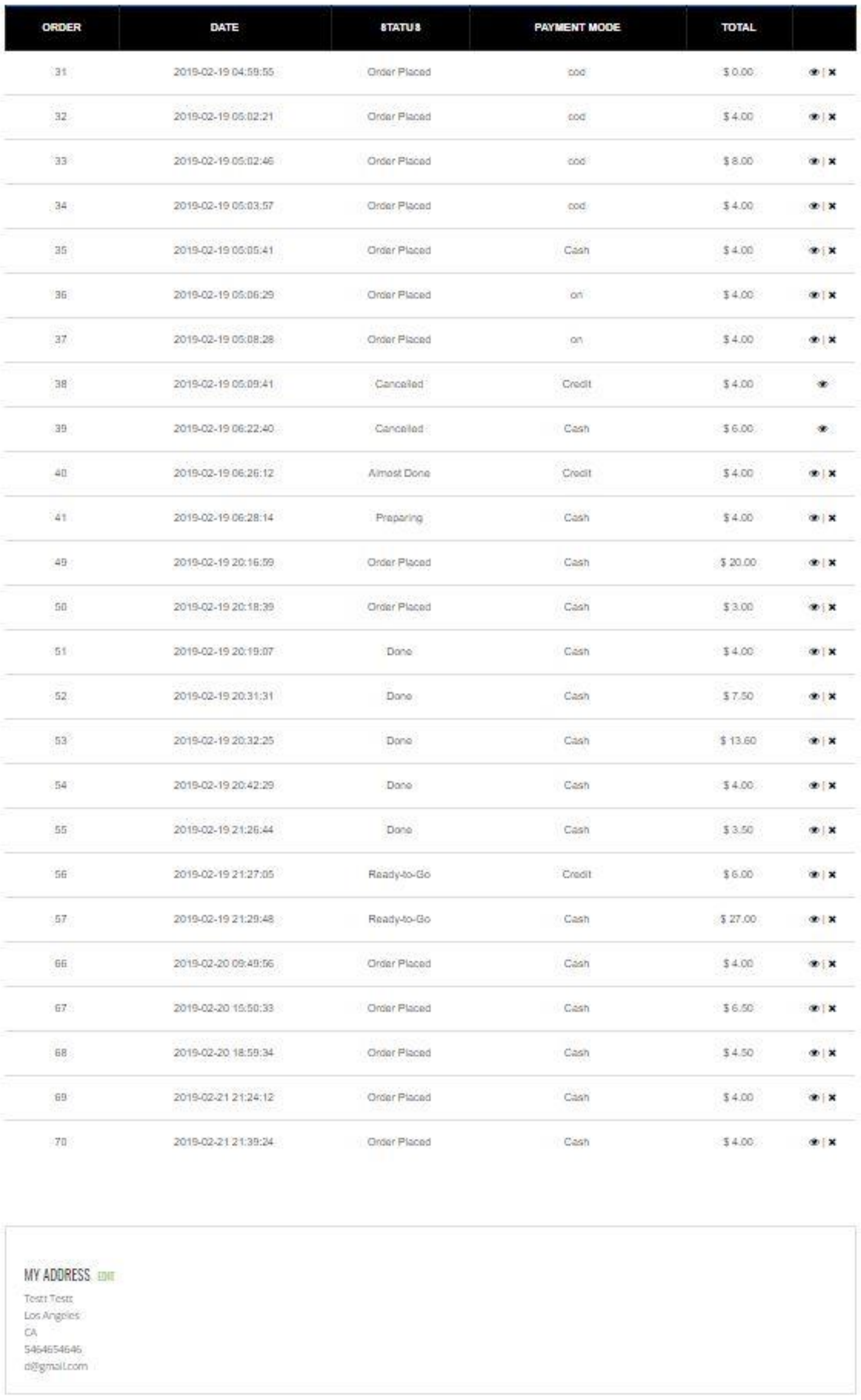
In Store Payment

○ Credit Card

Pay with Credit Card. We accept VISA, MASTERCARD, and EXPRESS.

Check-Out Page

The checkout page requires for the customer user to input their information such as their first name, last name, street address, city, state, zip code, and phone number. The page also displays the total amount the customer user owes. The customer is required to fill in the radio buttons by choosing the payment method. Below the page, the "Pay Now" button is displayed to have the customer user click and proceed in processing their order.



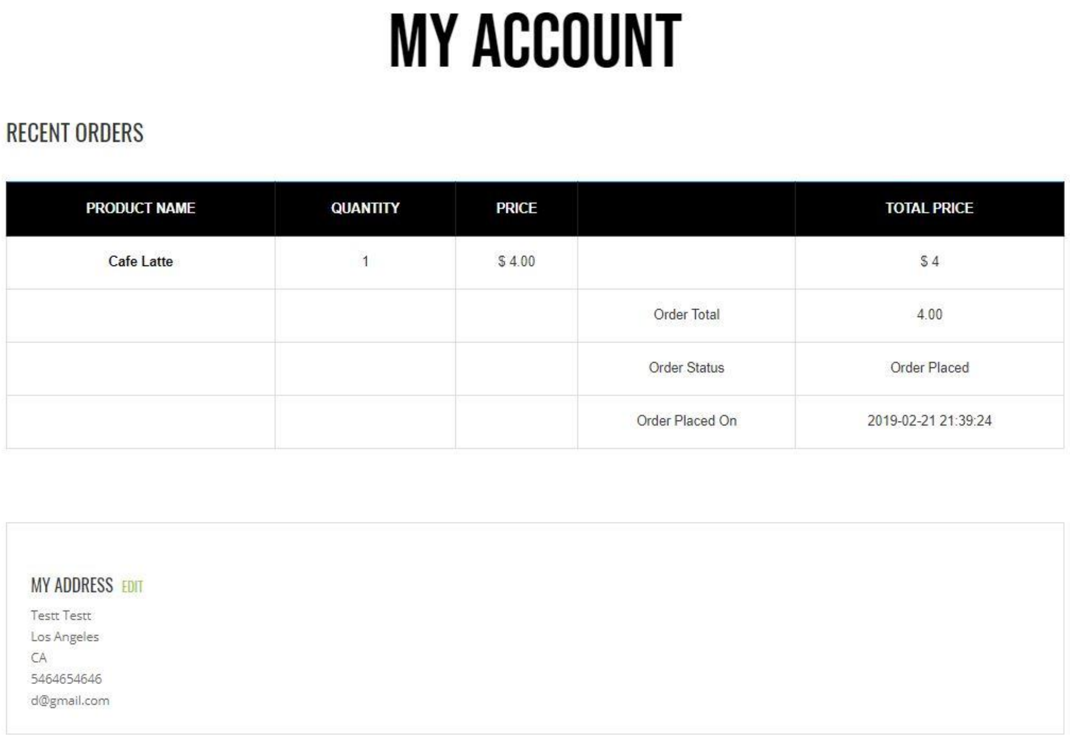
ORDER	DATE	STATUS	PAYMENT MODE	TOTAL	
31	2019-02-19 04:59:55	Order Placed	cod	$ 0.00	👁 | ✖
32	2019-02-19 05:02:21	Order Placed	cod	$ 4.00	👁 | ✖
33	2019-02-19 05:02:46	Order Placed	cod	$ 8.00	👁 | ✖
34	2019-02-19 05:03:57	Order Placed	cod	$ 4.00	👁 | ✖
35	2019-02-19 05:05:41	Order Placed	Cash	$ 4.00	👁 | ✖
36	2019-02-19 05:06:29	Order Placed	on	$ 4.00	👁 | ✖
37	2019-02-19 05:08:28	Order Placed	on	$ 4.00	👁 | ✖
38	2019-02-19 05:09:41	Cancelled	Credit	$ 4.00	👁
39	2019-02-19 06:22:40	Cancelled	Cash	$ 6.00	👁
40	2019-02-19 06:26:12	Almost Done	Credit	$ 4.00	👁 | ✖
41	2019-02-19 06:28:14	Preparing	Cash	$ 4.00	👁 | ✖
49	2019-02-19 20:16:59	Order Placed	Cash	$ 20.00	👁 | ✖
50	2019-02-19 20:18:39	Order Placed	Cash	$ 3.00	👁 | ✖
51	2019-02-19 20:19:07	Done	Cash	$ 4.00	👁 | ✖
52	2019-02-19 20:31:31	Done	Cash	$ 7.50	👁 | ✖
53	2019-02-19 20:32:25	Done	Cash	$ 13.60	👁 | ✖
54	2019-02-19 20:42:29	Done	Cash	$ 4.00	👁 | ✖
55	2019-02-19 21:26:44	Done	Cash	$ 3.50	👁 | ✖
56	2019-02-19 21:27:05	Ready-to-Go	Credit	$ 6.00	👁 | ✖
57	2019-02-19 21:29:48	Ready-to-Go	Cash	$ 27.00	👁 | ✖
66	2019-02-20 09:49:56	Order Placed	Cash	$ 4.00	👁 | ✖
67	2019-02-20 15:50:33	Order Placed	Cash	$ 6.50	👁 | ✖
68	2019-02-20 18:59:34	Order Placed	Cash	$ 4.50	👁 | ✖
69	2019-02-21 21:24:12	Order Placed	Cash	$ 4.00	👁 | ✖
70	2019-02-21 21:39:24	Order Placed	Cash	$ 4.00	👁 | ✖

MY ADDRESS EDIT

Testt Testt
Los Angeles
CA
5464654646
d@gmail.com

My Account Page - Viewing Orders

The customer user is redirected to viewing their orders through their account where they can view the status of their order, cancel their order, and click on the ‘eye’ button to view their order details. They are also given then option to change their address through the ‘edit’ button in the lower-left of the page.



MY ACCOUNT

RECENT ORDERS

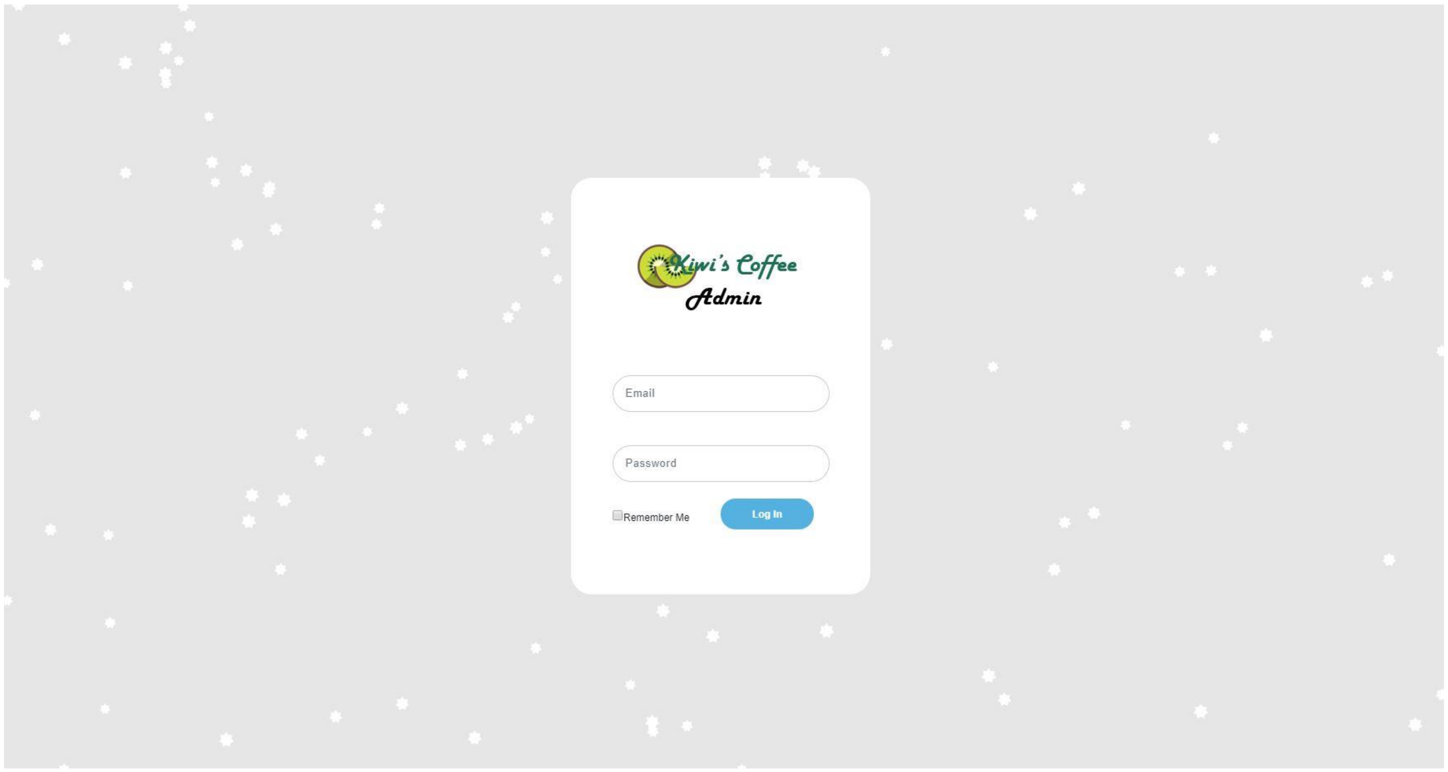
PRODUCT NAME	QUANTITY	PRICE		TOTAL PRICE
Cafe Latte	1	$ 4.00		$ 4
			Order Total	4.00
			Order Status	Order Placed
			Order Placed On	2019-02-21 21:39:24

MY ADDRESS EDIT

Testt Testt
Los Angeles
CA
5464654646
d@gmail.com

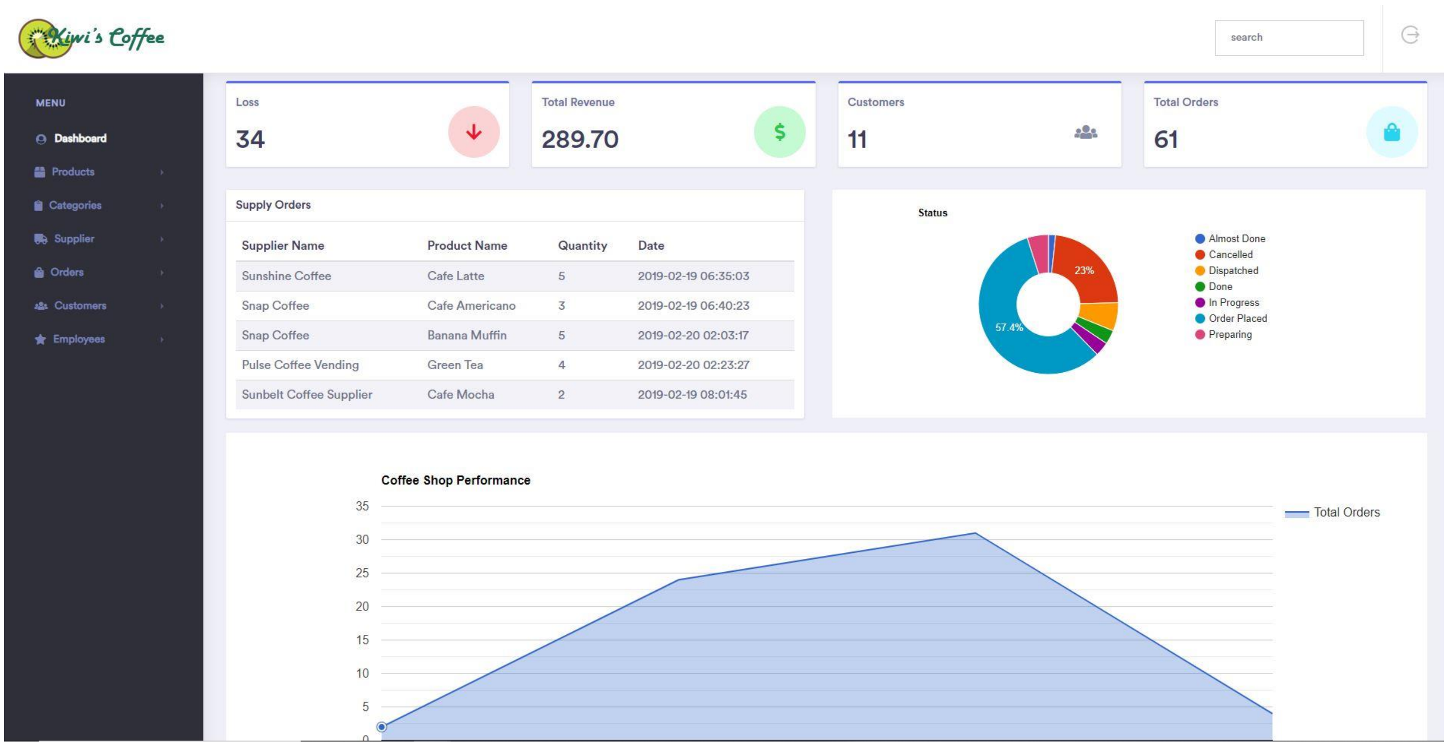
My Account-View a Specific Order

The customer is able to view a specific order within their list of orders to view the status of their order, date it was placed, total amount, the price of each item during the time, the quantity, and the name of the product. Like before, the customer user is given the option to click on the ‘edit’ button to edit their address contents, as well as name.



Admin Login Page

The employee user is redirected to the 'Admin Login Page' after they clicked 'login' in the navigation bar of the main page. The employee user is required to enter the correct information such as their email address and password where they are required to click submit to access the Admin dashboard page. If the employee user does not have the correct credentials, they will face with an error message indicating they are entering the incorrect data and therefore must try again.



Loss 34

Total Revenue 289.70

Customers 11

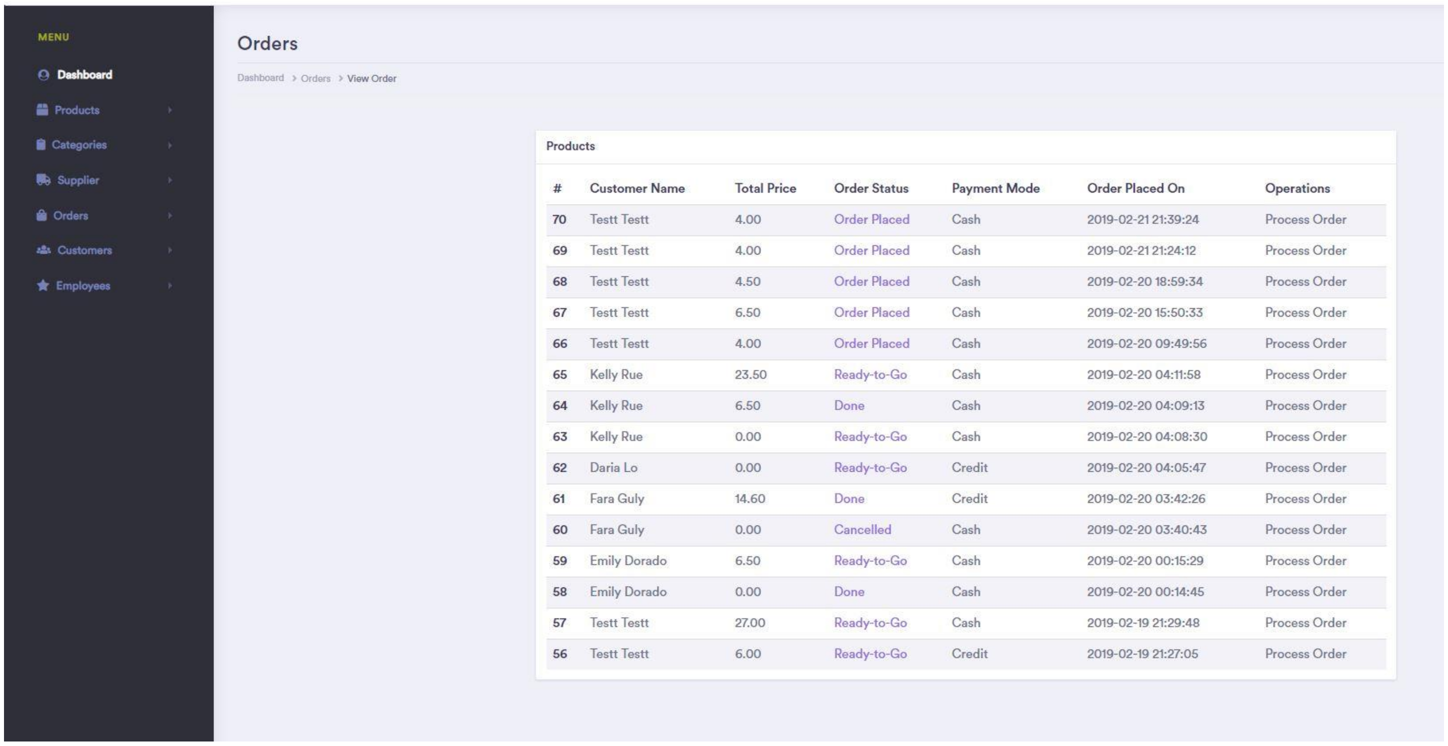
Total Orders 61

Supply Orders

Supplier Name	Product Name	Quantity	Date
Sunshine Coffee	Cafe Latte	5	2019-02-19 06:35:03
Snap Coffee	Cafe Americano	3	2019-02-19 06:40:23
Snap Coffee	Banana Muffin	5	2019-02-20 02:03:17
Pulse Coffee Vending	Green Tea	4	2019-02-20 02:23:27
Sunbelt Coffee Supplier	Cafe Mocha	2	2019-02-19 08:01:45

Admin Page - Dashboard

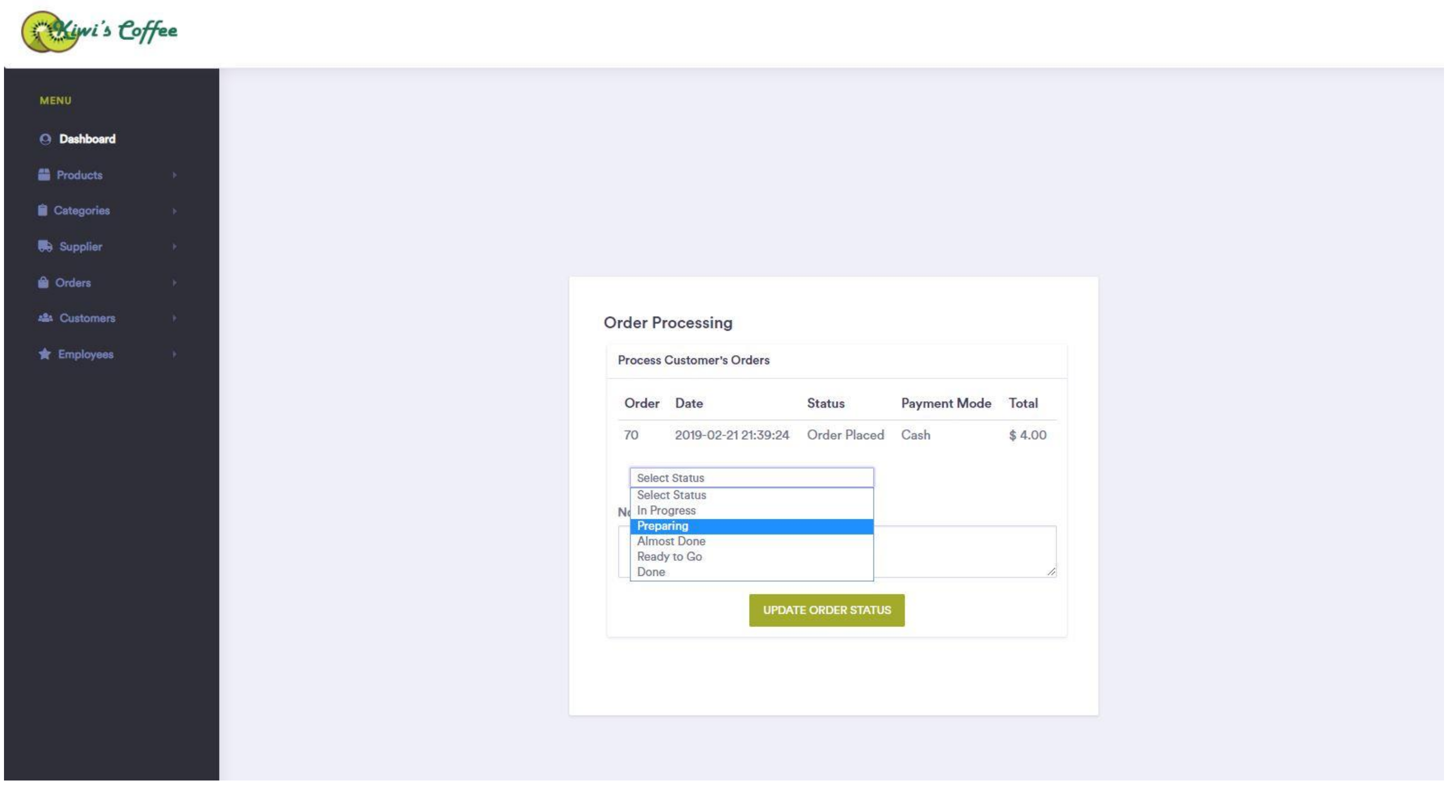
The employee user is redirected to the 'Admin Page' after they had access through the 'login' page with valid credentials. The employee user views information such as the total revenue produced in the coffee shop, the number of customers, number of total orders, the loss revenue when customers cancel their orders, the products orders from suppliers, and the coffee shops performance on a daily basis. Through the 'Admin Page', the employee user is able to navigate the side bar and choose to add, update, or delete items belonging to a specific entity. For instance, the employee user may want to delete a product. The employee user point-clicks on the sidebar 'Products' and is given a list of functions to perform. The employee user's goal is to delete a product, and therefore chooses to view the 'Products' where the employee user is given an option to either delete or edit the product. The user clicks on the 'trash' button, and deletes the product.



#	Customer Name	Total Price	Order Status	Payment Mode	Order Placed On	Operations
70	Testt Testt	4.00	Order Placed	Cash	2019-02-21 21:39:24	Process Order
69	Testt Testt	4.00	Order Placed	Cash	2019-02-21 21:24:12	Process Order
68	Testt Testt	4.50	Order Placed	Cash	2019-02-20 18:59:34	Process Order
67	Testt Testt	6.50	Order Placed	Cash	2019-02-20 15:50:33	Process Order
66	Testt Testt	4.00	Order Placed	Cash	2019-02-20 09:49:56	Process Order
65	Kelly Rue	23.50	Ready-to-Go	Cash	2019-02-20 04:11:58	Process Order
64	Kelly Rue	6.50	Done	Cash	2019-02-20 04:09:13	Process Order
63	Kelly Rue	0.00	Ready-to-Go	Cash	2019-02-20 04:08:30	Process Order
62	Daria Lo	0.00	Ready-to-Go	Credit	2019-02-20 04:05:47	Process Order
61	Fara Guly	14.60	Done	Credit	2019-02-20 03:42:26	Process Order
60	Fara Guly	0.00	Cancelled	Cash	2019-02-20 03:40:43	Process Order
59	Emily Dorado	6.50	Ready-to-Go	Cash	2019-02-20 00:15:29	Process Order
58	Emily Dorado	0.00	Done	Cash	2019-02-20 00:14:45	Process Order
57	Testt Testt	27.00	Ready-to-Go	Cash	2019-02-19 21:29:48	Process Order
56	Testt Testt	6.00	Ready-to-Go	Credit	2019-02-19 21:27:05	Process Order

Admin Page - View All Orders

The employee user views all orders placed by the customer user where they can view the customer's name, total price, order status, payment mode, and the time the order was placed. The employee user is required to process the order by pressing 'Process Order' if the customer's order is pending. Otherwise, if the customer cancelled their order, the employee user is not required to process their order.



Order Processing

Process Customer's Orders

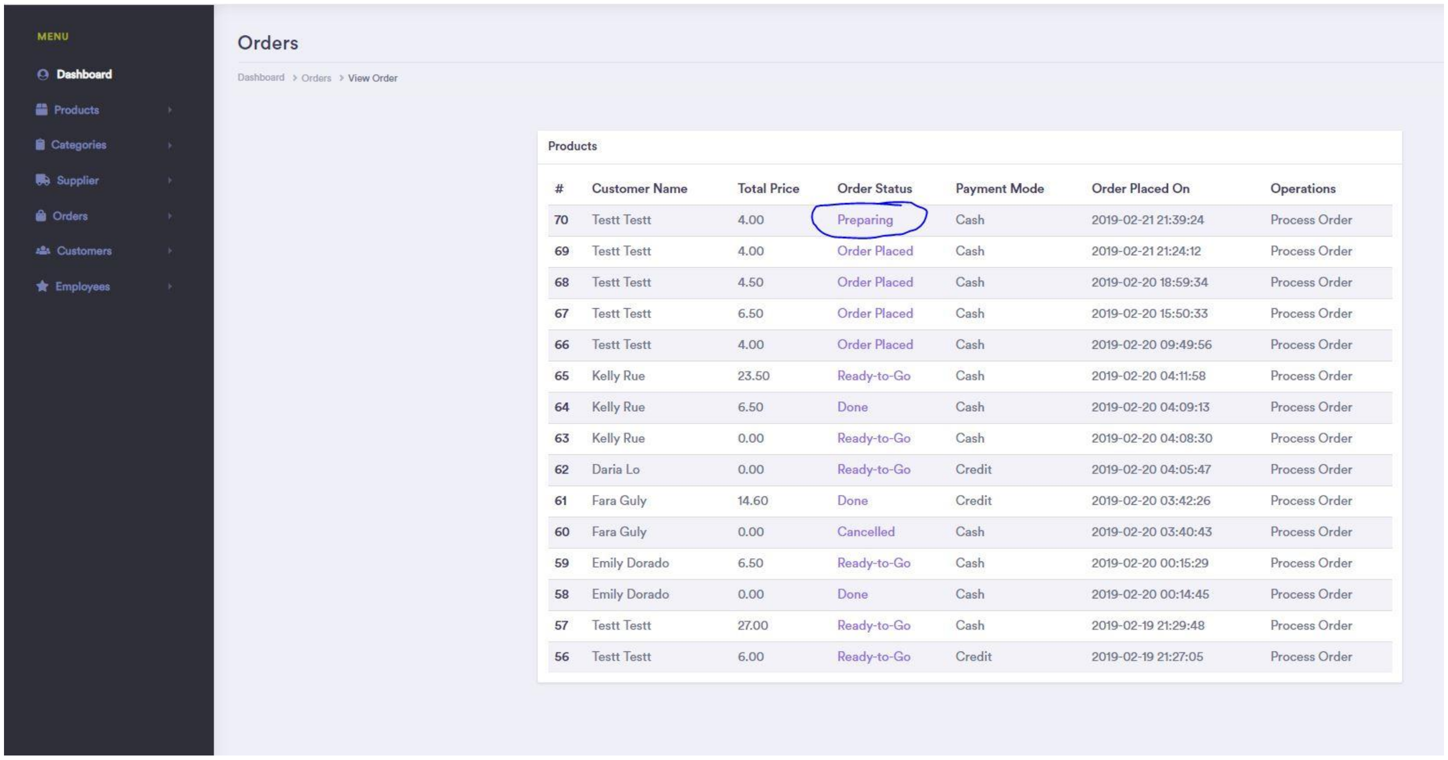
Order	Date	Status	Payment Mode	Total
70	2019-02-21 21:39:24	Order Placed	Cash	$ 4.00

Select Status
- Select Status
- In Progress
- Preparing
- Almost Done
- Ready to Go
- Done

UPDATE ORDER STATUS

Admin Page- Process Customer's Orders

The employee user processes the orders by selecting the status type. The status type lists out all possible status to have the employee user notify the customer user the progress of their products from start to finish.



Orders

Dashboard > Orders > View Order

Products

#	Customer Name	Total Price	Order Status	Payment Mode	Order Placed On	Operations
70	Testt Testt	4.00	Preparing	Cash	2019-02-21 21:39:24	Process Order
69	Testt Testt	4.00	Order Placed	Cash	2019-02-21 21:24:12	Process Order
68	Testt Testt	4.50	Order Placed	Cash	2019-02-20 18:59:34	Process Order
67	Testt Testt	6.50	Order Placed	Cash	2019-02-20 15:50:33	Process Order
66	Testt Testt	4.00	Order Placed	Cash	2019-02-20 09:49:56	Process Order
65	Kelly Rue	23.50	Ready-to-Go	Cash	2019-02-20 04:11:58	Process Order
64	Kelly Rue	6.50	Done	Cash	2019-02-20 04:09:13	Process Order
63	Kelly Rue	0.00	Ready-to-Go	Cash	2019-02-20 04:08:30	Process Order
62	Daria Lo	0.00	Ready-to-Go	Credit	2019-02-20 04:05:47	Process Order
61	Fara Guly	14.60	Done	Credit	2019-02-20 03:42:26	Process Order
60	Fara Guly	0.00	Cancelled	Cash	2019-02-20 03:40:43	Process Order
59	Emily Dorado	6.50	Ready-to-Go	Cash	2019-02-20 00:15:29	Process Order
58	Emily Dorado	0.00	Done	Cash	2019-02-20 00:14:45	Process Order
57	Testt Testt	27.00	Ready-to-Go	Cash	2019-02-19 21:29:48	Process Order
56	Testt Testt	6.00	Ready-to-Go	Credit	2019-02-19 21:27:05	Process Order

Admin Page-Order Status Updated

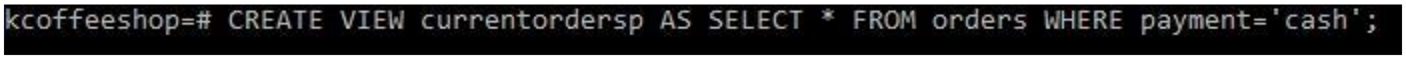
After the employee user updates the order status, the employee can redirect to the 'Orders' page to view the current status. By updating the status, the customer can immediately view the status of their order. The dashboard is also updated to indicate the status of each order.

5.1.3 Tables, Views, Stored, Subprograms and Triggers

The following utilizes functionalities provided in PostgreSQL comprising of tables, views, stored, subprograms, and triggers. In this section, the demonstration of views and procedures are described and displayed.

View: Current Order Payment Mode

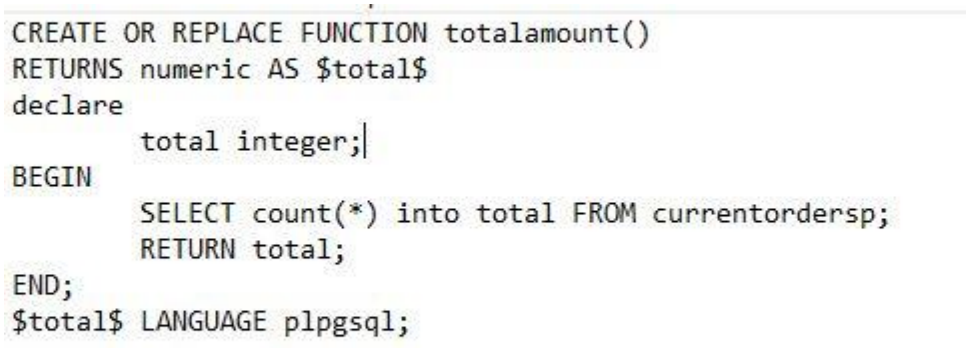
The purpose of the current order payment mode view is to display the current cash payments placed by certain customers. The table is useful to separate the table from the card payment purchases and simplifies the data by filtering out who used the card payment mode, and who used the cash payment mode.



```
kcoffeeshop=# CREATE VIEW currentordersp AS SELECT * FROM orders WHERE payment='cash';
```

Stored Procedure: View Total Amount of Records

The purpose of the stored procedure is to view the total amount of records from the view created previously for the current order payment mode. This is a sample that illustrates a function that returns the number of records in the 'currentorderp' table.



```
CREATE OR REPLACE FUNCTION totalamount()
RETURNS numeric AS $total$
declare
        total integer;
BEGIN
        SELECT count(*) into total FROM currentordersp;
        RETURN total;
END;
$total$ LANGUAGE plpgsql;
```

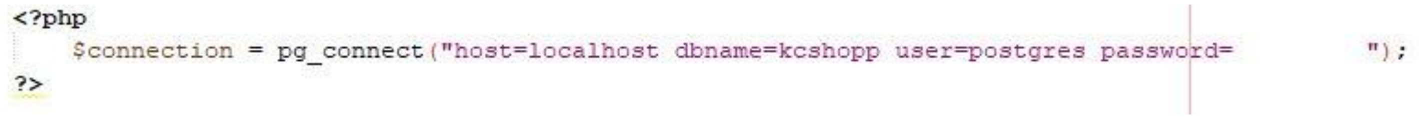
5.2 Programming Sections

The following section will review the 3-tier application comprising of the back-end, middle-tier, and front-end for Kiwi's Coffee Shop database application. The programming sections will briefly describe the three layers involved in the application with code samples and screenshots. Furthermore, the three-tier application architecture comprises of the client, business, and data layer aside from a two-tier application that contains only the client application and database server. The advantage of working with a three-tier web application involves high performance, scalability, flexibility of deployment, as well as improved data integrity and security. The three-tier architecture is applied to

Kiwi's Coffee Shop application composed of the data server using PostgreSQL, the business layer that contains dynamic content based on PHP, and the client layer based on Bootstrap, HTML, CSS, and JavaScript.

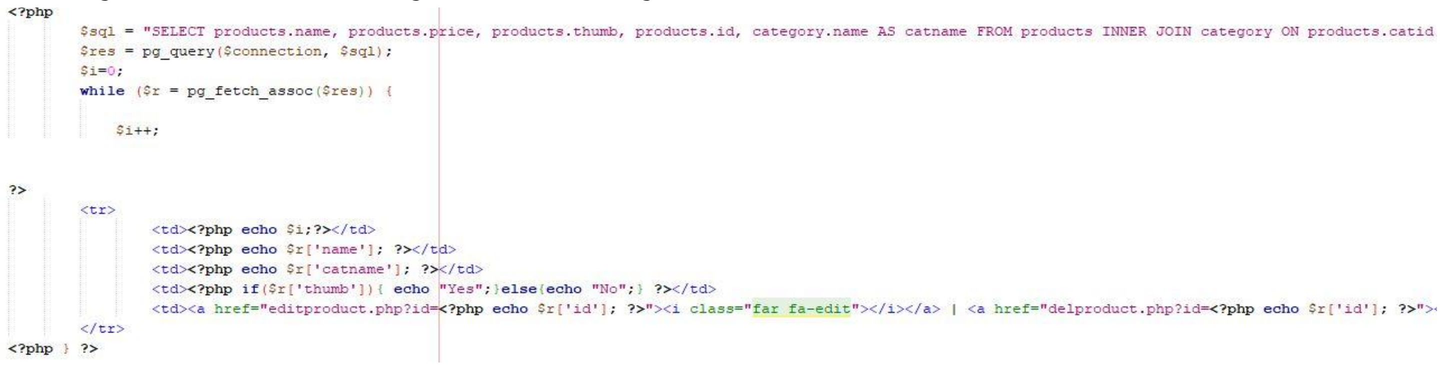
5.2.1 Server-Side Programming

The server-side programming deals with the information stored and retrieved from the database. The information is then passed back to the middle-tier for processing, and eventually goes back to the user in the client-server. The server-side programming consists of the SQL server and stored procedures to retrieve data from the SQL server. In the case of Kiwi's Coffee Shop, we use PostgreSQL database to store data into the database server. However, the SQL server must be connected where a connection string must be integrated in the PHP scripting language.



```
<?php
    $connection = pg_connect("host=localhost dbname=kcshopp user=postgres password=          ");
?>
```

The connection string allows execution of queries using pg_query() and returning the associative array that corresponds to the fetched records using pg_fetch_assoc(). The following is a sample of utilizing the connection string and set of PostgreSQL functions to execute and return data.



```
<?php
        $sql = "SELECT products.name, products.price, products.thumb, products.id, category.name AS catname FROM products INNER JOIN category ON products.catid
        $res = pg_query($connection, $sql);
        $i=0;
        while ($r = pg_fetch_assoc($res)) {

            $i++;


?>
        <tr>
                <td><?php echo $i;?></td>
                <td><?php echo $r['name']; ?></td>
                <td><?php echo $r['catname']; ?></td>
                <td><?php if($r['thumb']){ echo "Yes";}else{echo "No";} ?></td>
                <td><a href="editproduct.php?id=<?php echo $r['id']; ?>"><i class="far fa-edit"></i></a> | <a href="delproduct.php?id=<?php echo $r['id']; ?>">
        </tr>
<?php } ?>
```

5.2.2 Middle-Tier Programming

The majority of the application logic is in the middle-tier programming. This is because the client-side programming presents data to and collects data from the user, which leads for the server-side programming tier to store and retrieve data from the database. The middle-tiers' purpose is to bring together the other tiers drive the application's core capabilities. In other words, the middle-tier programming coordinates the application, processes commands, makes logical decisions and evaluations, and performs calculations.

The middle-tier programming language used in Kiwi's Coffee Shop application comprised of the scripting language PHP (Hypertext Preprocessor), a web server using XAMPP (open-source cross platform stack package), a scripting language engine for PHP (included in XAMPP). The use of a web server processes the HTTP requests and formulates responses, where in the case of the web database application, these request are made so programs can interact with the database management system. Using PHP scripting language as the middle-tier programming language offered flexibility by embedding scripts in Bootstrap/HTML that allowed for easy integration with the client-tier.

5.2.3 Client-Side Programming

The client-server side programming is the program that runs on the browser and deals with the user interface and any other processing that happens on the client machine such as reading and writing cookies. The client-side interacts with temporary storage, makes interactive web pages, interacts with local storage, sends request for data to the server, sends request to the server, and works as an interface between the server and user.

The client-side programming languages used in Kiwi's Coffee Shop is Bootstrap containing HTML and CSS-based design templates with JavaScript extensions and hypertext preprocessor(PHP). The open-source cross-platform web server solution stack package called Xampp is utilized to deploy and instantiate web pages that acts as a localhost when running it on the integrated development environment of NetBeans. In addition, Kiwi's Coffee Shop is mainly written with the combination of PHP and HTML to simplify the program.

5.3 Survey Questions

The following consists the survey questions provided by Dr. Wang to evaluate ourselves over our overall performance and outcome.

3b) An ability to analyze a problem, and identify and define the computing requirements and specifications appropriate to its solution	7
3e) An ability to design, implement, and evaluate a computer-based system, process, component, or program to meet desired needs. An ability to understand the analysis, design, and implementation of a computerized solution to a real-life problem.	6
3f) An ability to communicate effectively of audiences. An ability to write a technical document such as a software specification white paper or a user manual.	7
3j) An ability to apply mathematical foundations, algorithms principles, and computer science theory in the modeling and design of computer-based systems in a way that demonstrates comprehension of the tradeoffs involved in design choices.	7