Soda Vending Machine Refill Database

CMPS 3420 - Spring 2019

Group: 01

Gracelove Simons

Jianntyng Lu

Edwin Gonzalez - Computer Science

20th May, 2019

Table of Contents

1. Data collection and Conceptual Database Design

1.1 General Methods

The following chapter and its sections give an overview of the process used to design an Entity Relationship (ER) database model for a vending machine refill company. First, an overview of the company is given followed by a brief description on the ideas that helped shaped the design. After that, the ER model entities, relationships, constraints, and other important descriptions are explained. Although this ER model is used for one specific company it is general enough to serve as an all purpose model of any vending supply company.

1.1.1 Introduction to Organization

The imaginary vending machine inventory supply company we are designing this database for is called “Reloading Service Inc.” This company provides a full beverage delivery and inventory management service package to their clients. The office staff are in charge of creating orders that supply the company warehouse and client vending machines. Dispatchers organize orders into delivery routes and assign truck drivers to a route. Drivers deliver orders to designated vending machines, restock them, and record all necessary information, such as number of inventory upon arrival.

A vending machine is a great multi-purpose container. It provides temperature control like a refrigerator, storage space just like a small store, secure payment handling, and vandalism prevention systems.

1.1.2 Description of Fact-Finding Techniques

California State University Bakersfield (CSUB) is an ideal site for first-hand data collections. With over 8000 enrolled students on campus, vending machines are popular. There are 14 vending machines at CSUB alone. Delivery drivers constantly come to campus. We will ask drivers their company contact information and contact the company as potential vending machine owners to ask contract details.

A rough business model may also be studied online. Numerous business franchises already provide existing models strongly similar to our business model, with minor modifications.

The purpose of this information gathering techniques is to draw the schema of the business structure. This phase focuses on the design of an ER database schema to effectively represent a real world business model. Thus, for all tuples, we will use software generated data inputs in phase 3 when we type raw data (tuples) into each table (Relation in Relation Model).

1.1.3 Scope of the Conceptual Database

For simplicity reasons and to finish within our academic time constraint, the details of the corporate department are ignored. This design focuses on the barebone needs of a “Reloading Service.” A “mini-world” is represented by Drivers, Dispatchers, and all the entities they impact. This database will let the Reload Service Inc track inventory sales and orders, which locations are in need of supplies, and how effective delivery routes are. The ultimate goal is to provide clients with accurate reports of how their machines are managed while creating an efficient and manageable system for the delivery company at the same time .

Three key mini-world ideas:

1. A dispatcher places orders and dispatches drivers on routes.

2. A truck driver picks up an order (or orders) put in by a dispatcher, and carries out the delivery from a company warehouse to multiple locations with multiple vending machines on site.

3. A client report is able to be generated showing sales and other key financial attributes.

1.1.4 Entity and Relationship Sets Description

<u>Entity Set is as follows:</u>

Employee: <u>SSN</u>, Name, Address, Phone, Position, Salary, Start Date
An Employee works for Reloading Service Inc as has basic identifying information.

Dispatcher: Badge Number
A Dispatcher is in charge of placing orders, creating routes, and data keeping on behalf of the company.

Driver: License Number, License Exp
Drivers are responsible for delivering orders to vending machines.

Gas Receipt: Receipt ID, Address, Price, Date
A receipt used to keep track of gas expenditures.

Transportation: Plate Number, Make, Model, VIN, Registration Date
A truck driven a driver. One truck is to be assigned to one driver.

Supplier: Supplier_ID, FNAME, LNAME, Address, Phone
The 3rd party which supplies soda items in bulk quantity.

Order: Order ID, Order Type
An order is put into the system by a dispatcher. Vending orders are orders delivered by the truck driver to distribute inventory from a warehouse to its nearby locations, which has multiple vending machines.

Warehouse orders are orders delivered by a supplier to the company warehouse.

Warehouse: Warehouse ID, Address, Manager Name, Capacity
Represents the information of a physical warehouse on site.

Item Type: Item Type ID, Item Type Name, MSRP
Represents an Item Type that might be refilled from a 3rd party to our warehouse with its contracted reloading price in bulk.

Vending Machine: Machine_ID, Build, Items_per_slot, Capacity
Represents a single machine in a remote location owned by one of our clients.

Location: Location_ID, Address, Num_machines
Represents a single site which may be occupied by a VM.

Client: Customer_ID, FName, LName, Email, Phone, Company, Machines
A client is a private party who owns more than one VM on more than one location.
A client will pay for each soda we put into their machine plus 10% to 30%.

Relationship set is as follows:

A Dispatcher **Places Order** to Order; *Cardinality:* M..N; *Participation:* Partial, Total (Respectively)

A Dispatcher **Orders From** a Supplier; *Cardinality:* N..M; *Participation:* Partial, Total

A Supplier **Supplies** an Order with items in bulk quantity; *Cardinality:* 1..N; *Participation:* Partial, Total

A Driver gets an Order and **Delivers Order** to a Location; *Cardinality:* N..N; *Participation:* Total, Partial

A Location **Has** Vending Machines; *Cardinality:* 1..N; *Participation:* Total, Total

A Vending Machine contains **Vending Items** of Item Type; *Cardinality:* 1..N; *Participation:* Partial, Total

A Warehouse **Stocks** Item Types; *Cardinality:* 1..N; *Participation:* Partial, Total

An Order contains an **Order Item** of Item Type; *Cardinality:* 1..N; *Participation:* Partial, Total

A Warehouse receives a **Warehouse Order** of warehouse Order type;

Cardinality: 1..N; *Participation:* Partial, Total

A Warehouse creates a **Vending Order** of vending Order type;

Cardinality: 1..N; *Participation:* Partial, Total

A Driver **Receives** a vending Order type; *Cardinality:* 1..N; *Participation:* Partial, Partial

A Driver **Drives** Transportation; *Cardinality:* 1..1; *Participation:* Total, Partial

A Driver **Fuels** a truck and creates Gas Receipts; *Cardinality:* 1..N; *Participation:* Partial, Total

1.1.5 User Groups, Data Views and Operations

A user may interact with only a subset of ER diagram. That is to say, users need only be concerned with Entities and Relationship immediately associate to their pertinent interest.

Within the 1st boundary, comes the limit of tuple.

Certain Entities may only be able to see other Relationship/Entities to better model the mini-world.

Dispatcher

A dispatcher will have access to all supplier and placed order information. They will also have knowledge of the drivers who work for the company, the location of the company warehouse and the placement of client vending machines. A dispatcher will have the ability to pull data that shows inventory levels in both warehouses and vending machines as well as client contact information.

Driver

A driver will only view a subset of order (vending order) which are assigned to him/her by a dispatcher. They will have routes that contain the location of vending machines. A Driver is unaware of the business a dispatcher conducts behind the scenes and is only concerned with fulfilling a route created by a dispatcher. Drivers are also responsible for gathering gas receipts and submitting them to a dispatcher for logging. A dispatcher has access to all receipts while a driver is only aware of his.

Client:

A client does not need to be concerned with the details of the business. Sales and inventory reports will be generated for him by the delivery company.

1.2 Conceptual Database Design

In order to form a system to hold data for a company, one must gather all information and understand how they coexist and interlink with each other. In the previous section, we entail how we gathered that information.

Thus, this section will be about the information gathered and show how they are important and necessary to the system we are creating. The method of diagramming that we choose to display the data structure is the Entity-Relationship (ER) model. This method helps us represent the relationships between different aspects (or entities) of a company and how they affect each other.

Entity types, such as Employee, are created to unify entity sets where we define and name attributes, such as Name. We form relationship types that connect to show the relations between entity types and how they are related. Attributes are attached to relationship types as well for further understanding and relation displaying.

1.2.1 Entity Set Description

An entity is a real world object that exists independently. It could have a physical or conceptual existence. Our main entities are Employee, Client, Vending Machine, and Warehouse. They are the core parts of owning and running a vending machine refilling company. The following information will entail the entity types, their attributes, domain constraints, and keys that distinguish each entity from the next.

Entity Name: Employee

Description: An Employee is either a dispatcher or driver that is hired by our company owner to fulfill those posltions. Due to the scope of our database being the sole objective of refilling machines owned by others, the positions listed are driver and dispatcher, which are disjoint. Due to our company being small and local, the rate at which employees will not be high and the different types will not change drastically enough to offset the database.

Candidate Keys: Employee_ID, SSN
Primary Key: Employee_ID
Strong / Weak Entity: Strong
Fields to be indexed: Employee_ID, SSN, Name, Address, Phone, Position, Salary, Start Date

Attributes:

Attribute Name:	Employee_ID	SSN	Name
Description	Number assigned to each employee by the manager for clock in/out tracking	Distributed by the Social Security Department to uniquely identify persons.	Name of employee (First, Middle Initial, Last)

Domain/Type	Integer	Integer	String, Char, String
Value-Range	0 - MaxID	000000000 - 999999999	Any, A-Z, Any
Default Value	MaxID + 1	None	None
Null Allowed or not	Not	Not	Not
Unique	Yes	Yes	No
Single/Multivalued	Single	Single	Single
Simple/Composite	Simple	Simple	Composite

Attribute Name:	Address	Phone	Position
Description	The location of where the employee lives. (Street, City, State, Zip)	A contact number for the employee. (Mobile, House, or both)	The job the employee is assigned.
Domain/Type	String, String, String, Integer	Integer	String
Value-Range	Any, Any, Any, 00000-99999	0000000000-9999999999	Any
Default Value	None	None	None
Null Allowed or not	Not	Allowed	Not
Unique	No	Yes	No
Single/Multivalued	Multivalued	Multivalued	Single
Simple/Composite	Composite	Composite	Simple

Attribute Name:	Salary	Start Date
Description	How much the employee has been contracted to make.	The day the employee starts. (MM, DD, YYYY)

Domain/Type	Integer	Integer, Integer, Integer
Value-Range	0 - 9999999	01-12, 01-31, 0000-9999
Default Value	None	None
Null Allowed or not	Allowed	Not
Unique	No	No
Single/Multivalued	Single	Single
Simple/Composite	Simple	Simple

Entity Name: Driver

Description: Drivers are in charge of delivering orders and refilling vending machines on behalf of the company. Drivers are required to have an A-class driver's license and can operate any assigned vehicle in the company. A driver must also fuel their vehicle and record these transactions. An order is picked up at a designated pickup location (a warehouse).
A driver is a disjoint subclass with the entity **Employee** as a superclass. This entity frequency will be low as driver information should not change often and new entities will only be created when new drivers are hired.

Candidate Keys: License Number

Primary Key: License Number

Strong / Weak Entity: Strong
Fields to be indexed: License Number, License Exp

Attributes:

Name	License Number	License Exp

Description	A California issued Class A driver's license is needed to operate vehicle. A unique 8-digit number makes a license number and uniquely identifies a driver.	Expiration date of the driver's license.
Domain / Type	Integer	Date
Value / Range	Any 8-digit number combination: 00000000 - 99999999	Any
Default Value	None	None
Null Value Allowed	No	No
Unique	Yes	No
Single or Multi-value	Single	Single
Simple or Composite	Simple	Simple

Entity Name: Transportation

Description: Transportation is the vehicle that a driver uses to carry out deliveries. A driver is assigned one truck to drive for insurance purposes. A truck has a unique plate number, a vehicle identification number (VIN), registration information, and descriptors to help identify a vehicle. A model will have a year and can have a subclass specifier.
Example of make / model: 2016 Honda Accord or 2015 Honda Accord SE
Truck information should hardly change so there will be a low frequency of table operations.

Candidate Keys: Plate Number, VIN
Primary Key: License Number
Strong / Weak Entity: Strong
Fields to be indexed: Plate Number, Make, Model, VIN, Registration Date

Attributes:

Name	Plate Number	Make	Model
Description	Every vehicle must have a unique plate number. In California, a plate number is 7 digits.	The vehicle type. Will have a year and make.	Identifies a subclass of Make. Gives more detail on the type of vehicle.
Domain /	Integer	Integer, String	String

Type			
Value / Range	Any 7-digit number combination: 0000000 - 9999999	Integer: Any valid year. String: Any	Any
Default Value	None	None	None
Null Value Allowed	No	No	No
Unique	Yes	No	No
Single or Multi-value	Single	Single	Single
Simple or Composite	Simple	Composite	Simple

Name	VIN	Registration Date
Description	A unique number that identifies a vehicle. Created at time of manufacture.	Last renewal date as well as next expiration date.
Domain / Type	Integer	Date, Date
Value / Range	Any 17-digit number combination: (9)^17 choices where each choice is greater than or equal to zero.	Any, Any
Default Value	None	None
Null Value Allowed	No	No
Unique	Yes	No
Single or Multi-value	Single	Single

Simple or Composite	Simple	Composite

Entity Name: Gas Receipt

Description: A driver must fuel their vehicle. A fuel transaction produces a receipt that provides a location, charge, and date. Receipts can be used to check how much gasoline is being spent on specific routes or to check that vehicles are being fueled frequently. A moderate amount of these entities should be created per month as gasoline is a key component to keeping vehicles running.

Candidate Keys: Receipt ID, Address
Primary Key: Receipt ID
Strong / Weak Entity: Strong
Fields to be indexed: Receipt ID, Address, Price, Date

Attributes:

Name	Receipt ID	Address	Price	Date
Description	Identifies a Receipt	Street Address, City, State, Zip of gas station location.	Price paid for refueling	Date of gas purchase
Domain / Type	Integer	String, String, String, Integer	Float	Date
Value /	0 - MaxID	Any, Any, Any,	0 - Any positive	Any

Range		00000-99999		
Default Value	MaxID + 1	None	None	None
Null Value Allowed	No	No	No	No
Unique	Yes	No	No	No
Single or Multi-value	Single	Single	Single	Single
Simple or Composite	Simple	Composite	Simple	Simple

Entity Name: Order

Description: An order is a crucial entity that will be created extremely frequently. Two order types exist (each containing one or more item types). One type is a Warehouse order: this order contains supplies shipped from a supplier to a warehouse. These orders are large bulk orders that are meant to keep the company warehouse stocked. The second order type is a Vending order: this order contains the supplies that are to be delivered to a vending machine for restocking. These orders are created at the company warehouse. An order is placed by a dispatcher. The delivery to a warehouse is done by an outside supplier while the delivery to a vending machine is done by a company driver.

Candidate Keys: Order ID, Order Type
Primary Key: Order ID
Strong / Weak Entity: Strong
Fields to be indexed: Order ID, Order Type

Attributes:

Name	Order ID	Order Type
Description	Identifies an order	IDs if order ships to a warehouse or to a vending machine
Domain / Type	Integer	String

Value / Range	0 - MaxID	Any
Default Value	MaxID + 1	None
Null Value Allowed	No	No
Unique	Yes	No
Single or Multi-value	Single	Single
Simple or Composite	Simple	Simple

Entity Name: Warehouse

Description: A warehouse is a building that stores orders as inventory. Vending machine orders are created using warehouse inventory. A storage capacity is decided by the company and the number of total item types stored should not exceed the capacity. There can be an expected high number of table operations as this entity is directly linked to the creation of each order entity.

Candidate Keys: Warehouse ID
Primary Key: Warehouse ID
Strong / Weak Entity: Strong
Fields to be indexed: Warehouse ID, Address, Manager Name, Capacity

Attributes:

Name	Warehouse ID	Address	Manager Name	Capacity
Description	Identifies a unique warehouse building	Street Address, City, State, Zip	An Employee who manages the warehouse	The maximum amount of items that can be stored in a warehouse.
Domain / Type	Integer	String, String, Sring, Integer	String	Integer
Value /	0 - MaxID	Any, Any, Any,	Any	0 - Any positive

Range		00000-99999		
Default Value	MaxID + 1	None	None	0
Null Value Allowed	No	No	No	No
Unique	Yes	No	No	No
Single or Multi-value	Single	Single	Single	Single
Simple or Composite	Simple	Composite	Simple	Simple

Entity Name: Item Type

Description: Item Type represents what can be contained in an order, what a vending machine can hold, or what a warehouse will store. An item type represents a type of item, such as a Pepsi bottle. This company deals specifically with beverages: all items will be a type of beverage. However, an item type can represent any food or beverage if a vending machine owner wishes to expand their inventory. An item type has a name and a recommended retail price. This entity will be created extremely frequently as every order must contain an item type.

Candidate Keys: Item Type ID, Item Type Name
Primary Key: Item ID
Strong / Weak Entity: Strong
Fields to be indexed: Item Type ID, Item Type Name, MSRP

Attributes:

Name	Item Type ID	Item Type Name	MSRP
Description	Identifies a unique item	Name of an item.	Manufacturer Suggested Retail Price
Domain / Type	Integer	String	Float
Value / Range	0 - MaxID	Any	0 - Any positive
Default Value	MaxID + 1	None	None

Null Value Allowed	No	No	No
Unique	Yes	No	No
Single or Multi-value	Single	Single	Single
Simple or Composite	Simple	Simple	Simple

Entity Name: Location

Description: The Location entity refers to a site where vending machines are located. One location entity represents a general area, such as a school, and the number of machines currently at the location. Table operations should not be very frequent: usually locations and machine placements don't move very often for consistency.

Candidate Key: Location_ID, Address
Primary Key: Location_ID
Strong / Weak Entity: Strong
Fields to be indexed: Location_ID, Address, Num_machines

Attributes:

Attribute Name:	Location_ID	Address
Description	Each location that a machine is located at will have a number assigned for easier identification.	The street, city, state and zip of a delivery location.
Domain/Type	Integer	String, String, String, Integer
Value-Range	0 - MaxID	Any, Any, Any, 00000-99999
Default Value	MaxID + 1	None

Null Allowed or not	Not	Not
Unique	Yes	No
Single/Multivalued	Single	Multivalued
Simple/Composite	Simple	Composite

Entity Name: Vending machine

Description: This entity is in place because the dispatcher and company owner need to know what is inside of the machine at all times so that they know what is to be refilled, switched for another and to allocate space for more of an item. It is linked to “Location” and “Item” due to it being at that spot and having specific items based on what’s popular at that location.

Candidate Key: Machine_ID, Build
Primary Key: Machine_ID
Strong / Weak Entity: Strong
Fields to be indexed: Machine_ID, Build, Items_per_slot, Capacity

Attributes:

Attribute Name:	Machine_ID	Build	Items_per_slot
Description	Assigned to the machine by the manufacturer and used to keep track of the machine.	The make, model and year made.	How many items can be in each slot.
Domain/Type	Integer	String, String, Integer	Integer
Value-Range	0 - MaxID	Any, Any, 0000-9999	0-99

Default Value	MaxID + 1	None	0
Null Allowed or not	Not	Allowed	Not
Unique	Yes	No	No
Single/Multivalued	Single	Multivalued	Single
Simple/Composite	Simple	Composite	Simple

Attribute Name:	Capacity
Description	This represents the amount of items the machine can hold.
Domain/Type	Integer
Value-Range	0-999
Default Value	0
Null Allowed or not	Not
Unique	No
Single/Multivalued	Single
Simple/Composite	Simple

Entity Name: Client

Description: The Customer is the owner of the vending machine. Our main goal is to keep their machine in perfect condition so that we both can make as much profit as possible. Each customer and the amount of machines they own that are refilled by us is to be recorded.

Candidate Key:
Primary Key: Customer_ID
Strong / Weak Entity: Strong
Fields to be indexed: Customer_ID, FName, LName, Email, Phone, Company, Machines

Attributes:

Attribute Name:	Customer_ID	Name	Email
Description	Number assigned to the company that we refill machines for	Name of the point of contact, or owner, of the machine(s). (First, Middle Initial, Last)	Email for the point of contact.
Domain/Type	Integer	String, Char, String	String
Value-Range	0 - MaxID	Any, A-Z, Any	Any
Default Value	MaxID + 1	None	None

Null Allowed or not	Not	Not	Not
Unique	Yes	No	Yes
Single/Multivalued	Single	Multivalued	Single
Simple/Composite	Simple	Composite	Composite

Attribute Name:	Phone	Company Name	Machines
Description	The number used to contact the company and or send/receive a fax to/from.	Name of the company that we refill machines for .	
Domain/Type	Integer	String	
Value-Range	0000000000-9999999999	Any	
Default Value	None	None	
Null Allowed or not	Not	Not	
Unique	Yes	Yes	
Single/Multivalued	Multivalued	Single	
Simple/Composite	Composite	Simple	

Entity Name: Dispatcher

Description: Due to the size of our company, the Dispatcher has the job of sending orders to the Driver, maintaining the inventory in warehouse, and keeping records for the machine owner. They'll have a separate identification number and keep in touch with machine owners and maintain contracts, order and keep track of the warehouse inventory, and send orders to drivers.

Candidate Key: Badge Number
Primary Key: Badge Number
Strong / Weak Entity: Strong
Fields to be indexed: Badge Number

Attributes:

Attribute Name:	Badge Number
Description	Number assigned to a dispatcher to identify length of employment with company and what orders they place and send out.
Domain/Type	Integer
Value-Range	0 - MaxID

Default Value	MaxID + 1
Null Allowed or not	Not
Unique	Yes
Single/Multivalued	Single
Simple/Composite	Simple

Entity Name: Supplier

Description: The Supplier is the company or site we order our products from. That can range from sodas and water for the machine refills to toilet paper to paper. The entity is an important aspect of the database in the sense that we are to keep track of all the things purchased related to the business for tax, legal and budgeting purposes.

Candidate Key: Supplier_ID, LNAME
Primary Key: Supplier_ID
Strong / Weak Entity: Strong
Fields to be indexed: Supplier_ID, FNAME, LNAME, Address, Phone

Attributes:

Attribute Name:	Supplier_ID	Name	Address
Description	A special number assigned to each company/person we order from.	Name of the person we place orders through. (First, Middle Initial, Last)	Where the company we are ordering from is located.
Domain/Type	Integer	String, Char, String	String, String, String, Integer

Value-Range	0 - MaxID	Any, A-Z, Any	Any, Any, Any, 00000-99999
Default Value	MaxID + 1	None	None
Null Allowed or not	Not	Not	Not
Unique	Yes	No	Yes
Single/Multivalued	Single	Multivalued	Multivalued
Simple/Composite	Simple	Composite	Composite

Attribute Name:	Phone
Description	Number used to contact the company we order from.
Domain/Type	Integer
Value-Range	0000000000-9999999999
Default Value	None
Null Allowed or not	Not
Unique	Yes
Single/Multivalued	Multivalued
Simple/Composite	Composite

1.2.2 Relationship Set Description

Relationships connect two or more entities by defining how two entities interact in the mini-world. Just like entities relationships can have attributes. They contain constraints that specify whether the existence of an entity depends on it being related to another entity and the minimum number of instances each entity can participate in.

The following section defines each relationship type in this ER model. Descriptions of a relationship include its name, cardinality and participation constraints, and any attributes it may have. These descriptors help tie together relationships with their involved entity sets.

Relationship: Drives
Description: Drivers use vehicles to transport and fulfill deliveries. The mapping is 1..1: a driver must be assigned one vehicles and one vehicle can be commandeered by one driver.
Entity Sets Involved: Driver, Transportation
Cardinality Mapping: 1..1
Descriptive Field: None
Participation Constraint: Total participation for Driver and and partial Vehicle. A driver must be assigned a vehicle to delivery orders. A vehicle can exist without being assigned a driver. For example, an extra vehicle may remain unused until a new driver is hired and assigned to it.

Relationship: Fuels
Description: Vehicles need fuel to operate. A driver is responsible for supplying their assigned trucks with gasoline. A state of the transaction must be cataloged in the entity Gas Receipt. This relationship allows the company to better monitor fuel cost and develop routes based on fuel costs. Mapping is 1..N: many receipts must link to one driver but one driver must take responsibility for all gas transactions.
Entity Sets Involved: Driver, Gas Receipt
Cardinality Mapping: 1..N
Descriptive Field: None
Participation Constraint: Total participation for Gas Receipt and partial for Driver. A driver may not have to pump gas (it can be assumed all vehicles are always fueld) or may not be tasked with collecting receipts. A receipt can only exist once a driver has decided to pump gas and record the transaction.

Relationship: Receives
Description: A driver will receive one or more vending orders that need to be delivered to designated locations. An order or group of orders can only be delivered by one driver per route. A Date and Receiving Time will be recorded through this relationship. A driver can only receive a vending order.
Entity Sets Involved: Driver, Order
Cardinality Mapping: 1..N
Descriptive Field: Date, Receiving Time
Participation Constraint: Partial participation for both. An order can exist without a driver, such as a warehouse order. A driver can exist without having to deliver an order.

Relationship: Warehouse Order
Description: This relationship links a warehouse **Order** with the entity **Warehouse**. A warehouse order is intended to be shipped and stored at the company warehouse. Each order delivered to a warehouse should be used to increment the warehouse inventory (which is done through the stocks relationship). This relationship records the time and date an order is placed.
Entity Sets Involved: Order, Warehouse
Cardinality Mapping: N..1
Descriptive Field: Time Ordered, Order Date

Participation Constraint: Total participation for Order. An order is required to be shipped to a warehouse. An order would cease to exist if it had no location to be stored in. A warehouse is partial participation as it can exist without receiving orders.

Relationship: Vending Order
Description: This relationship links a vending machine **Order** with the entity **Warehouse**. A vending machine order is created using warehouse inventory and is used by drivers to refill vending machines. When a vending order is created, the warehouse inventory must be decremented through the same relationship that it is incremented.
Entity Sets Involved: Order, Warehouse
Cardinality Mapping: N..1
Descriptive Field: Time Created, Order Date
Participation Constraint: Partial participation by warehouse. A warehouse can exist solely to receive warehouse orders and is not dependent on truck orders to exist. Total participation for order as it can only exist if a warehouse creates it.

Relationship: Order Item
Description: Every order must contain at minimum one item type. This relationship links the entity **Item Type** to **Order**. When an order is placed, a number of item types, price per item type, and expiration date are identified through this relationship.
Entity Sets Involved: Order, Item Type
Cardinality Mapping: N..N
Descriptive Field: Num Item Type, Item Type Price, Exp Date
Participation Constraint: Total participation by both. In this diagram, an item type cannot exist unless some outside entity is containing it. At the same time, an order cannot exist without at least one item being created.

Relationship: Vending Item
Description: A vending machine contains item types. The relationship describes the number of item types, what slots of a machine are filled with an item type, the price an item is to be sold for at a machine, and the expiration date. With this relationship, inventory and sales of individual machines can be tracked so more accurate orders may be placed.
Entity Sets Involved: Item Type, Vending Machine
Cardinality Mapping: N..1
Descriptive Field: Num Item Type, Slots Filled, Price, Exp Date

Participation Constraint: Total participation by Item Type. An item cannot exist unless some outside entity is containing it. A vending machine can exist without any inventory so it has partial participation.

Relationship: Stocks
Description: A warehouse must be able to hold item types as inventory. This relationship links the **Item Type** entity to a warehouse for inventory keeping. When a warehouse order is delivered, the number of items are incremented. When a vending order is created, the number of items are decremented.
Entity Sets Involved: Warehouse, Item Type
Cardinality Mapping: 1..N
Descriptive Field: Num Item Type, Exp Date
Participation Constraint: Total participation by Item. An item cannot exist unless some outside entity is containing it. Partial participation by warehouse as it can exist without having any items.

Relationship: Orders From
Description: This relationship links **Dispatcher** to **Supplier**. A **Dispatcher** will place orders from a **Supplier**. Those supplies can range from sodas and waters to paper and other office supplies needed. The mapping is N:M because the we can have N dispatchers and they all can place orders to M amount of suppliers.
Entity Sets Involved: Dispatcher, Supplier
Mapping Cardinality: N:M
Descriptive Field: None
Participation Constraint: Total participation for Supplier. The supplier wouldn't have anywhere to send an order without the dispatcher placing one. Partial participation for Dispatcher because it can exist without ordering from a Supplier.

Relationship: Places Order
Description: This relationship links **Dispatcher** to **Order**. The dispatcher places orders for refilling machines and restocking the warehouse. The mapping is N:N because N dispatchers can place N orders. The Time Placed and Num Orders descriptive fields are necessary for keeping track of how much was ordered and when so that it is easily re-traceable if need be.
Entity Sets Involved: Dispatcher, Order
Mapping Cardinality: N:M

Descriptive Field: Time Placed, Num Orders
Participation Constraint: Total participation for Order. An order wouldn't be placed if there was no Dispatcher to put it in. Partial participation for Dispatcher because they can exist without having to place orders.

Relationship: Supplies
Description: This relationship links **Supplier** to **Order** where the supplier supplies us with the items we ordered from them and they get sent to the warehouse to eventually be sent out to the machines. The mapping of this is 1:N because 1 supplier can supply N orders.
Entity Sets Involved: Supplier, Order
Mapping Cardinality: 1:N
Descriptive Field: None
Participation Constraint: Total participation for Order. An order couldn't be supplied without a supplier that we order from. Partial participation for Supplier because they can be a supplier used but not for every or the majority of orders.

Relationship: Delivers Order
Description: This relationship links an **Order** to a **Location** where a machine (or more than one) will be filled. The mapping for this is N:N because our company can be refilling the location as many times as need throughout the contract designated time. The date and time delivered are used to keep track of when and if shipping was successful.
Entity Sets Involved: Order, Location
Mapping Cardinality: N:N
Descriptive Field: Date, Time
Participation Constraint: Partial participation for Location because location can stand on its own and there would be no use in making a delivery if there isn't a location to deliver to. Total participation for Order because there wouldn't be anything to deliver without it.

Relationship: Has
Description: This relationship links **Location** to **Vending Machine**. A location can hold more than one vending machine, creating a 1:N mapping. That location can be a school, larger or small business, or apartment complex. Each location will be identifiable by their given IDs and addresses.
Entity Sets Involved: Location, Vending Machine
Mapping Cardinality: 1:N
Descriptive Field: None

Participation Constraint: Total participation for both Vending Machine and Location. The reason being is that a Location can stand alone, but there would be no point in us knowing of that location if it has no vending machine. And a Vending Machine has to have a location or else we wouldn't be able to find and refill it.

Relationship: Owner
Description: This relationship links **Client** to **Vending Machine**. The client is basically the owner of the machine(s). They are the contract holder and addressee. The mapping to this is 1:N because 1 client can own many machines.
Entity Sets Involved: Client, Vending Machine
Mapping Cardinality: 1:N
Descriptive Field: None
Participation Constraint: Total participation for both **Client** and **Vending Machine**. Without a client there would be no vending machine and if a client had no machine, there would be no need for them to be our client.

1.2.3 Related Entity Type

Specialization is defining a set of subclasses from a parent entity type. The parent entity type is defined to be a superclass and the set of subclasses inherit all attributes of the parent class. When subclasses are created, there must be some attribute that distinguishes one subclass from another in a set. Subclasses are also able to act as a superclass and spawn subclasses of their own. A **specific attribute** is an attribute that is unique to a subclass and should not exist in other subclasses of the specialized set.

In this ER schema, Driver and Dispatcher are a set of specialized subclasses of the entity type Employee.

Generalization is the opposite thought process of specialization but achieves the same goal. With generalization, you create entities first that are neither a set of subclasses or a superclass. However, after designing a schema, a set of entities might share many of the same attributes minus a few attributes that distinguish them from each other. A decision can then be made to create a superclass entity that holds these attributes that each subclass will inherit from.

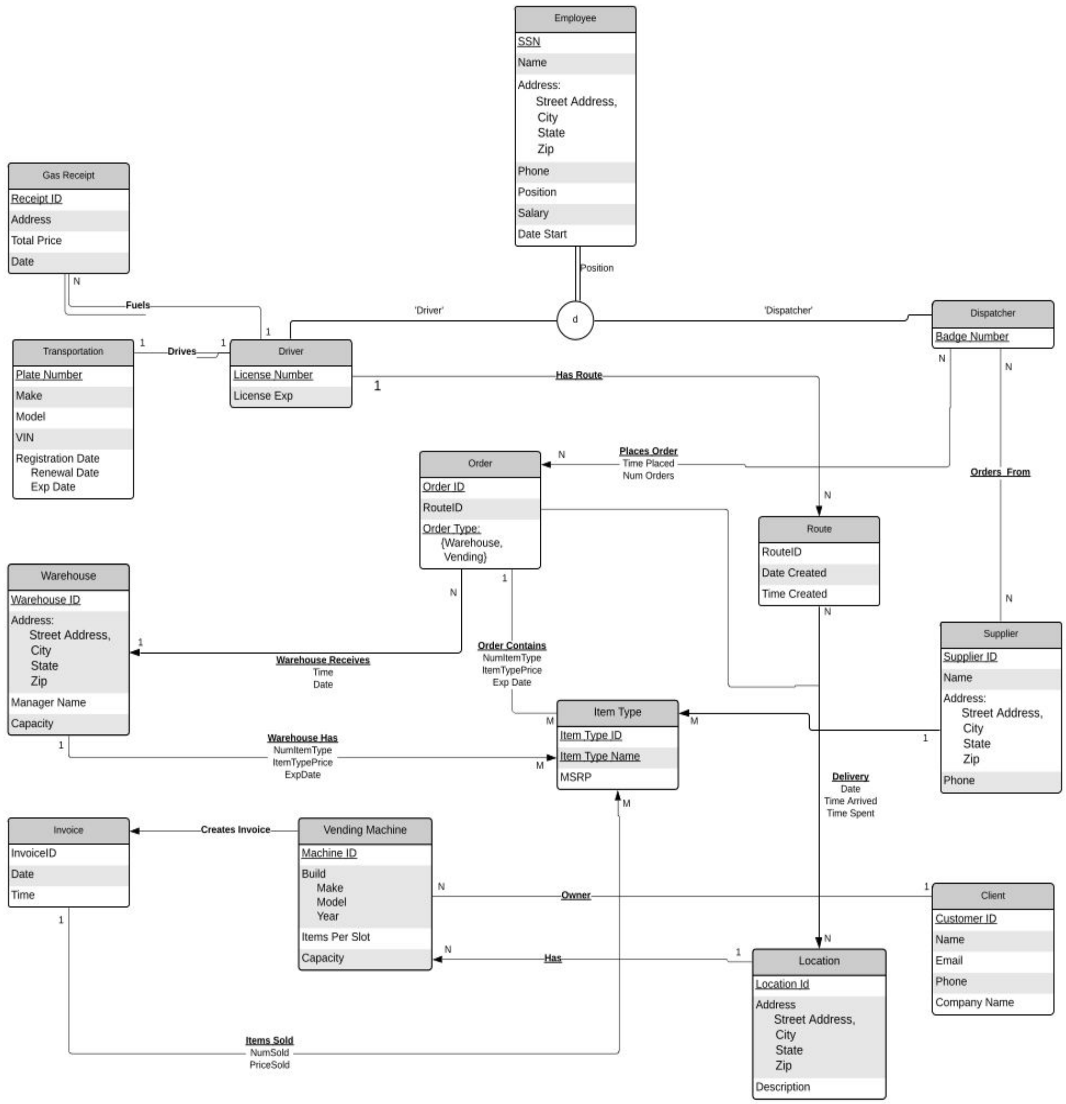
A constraint that exist in our schema is what is called a **predicate-defined (condition-defined)** constraint. These constraints are used to determine which entities will become what subclass.

The entity **Driver** is created when an employee has their position attribute set to 'Driver.' The same logic happens with the entity **Dispatcher**. Both entities also have a total participation constraint. A Driver and Dispatcher must exist and are dependent on the Employee entity existing. These two entities are disjoint subclasses, meaning each subclass entity is unique from each other and have specific attributes that separate them.

1.2.4 ER Diagram

An Entity Relationship(ER) Diagram helps model the structural design of a database. An ER diagram is a structure made up of symbols that are interconnected by lines. Entities and relationships are represented as some container symbol and a list of associated attributes are listen with each entity or relationship. Lines show which entities link to which relationships as well as what degree of participation are involved between said entities. Double lines signify total participation and single lines represent partial participation. Markers such as "1, N, M" show the cardinality mapping of entities and relationships. A U represents a union of subclasses and a D a disjoint set of subclasses. These basic concepts help form a diagram that paints a good picture of the overall concepts a database should accomplish.

Bellow is the ER diagram for a Soda Vending Machine Refill company:



Employee
SSN
Name
Address: Street Address, City State Zip
Phone
Position
Salary
Date Start
Position
d
'Driver'
'Dispatcher'
Gas Receipt
Receipt ID
Address
Total Price
Date
Fuels
Transportation
Plate Number
Make
Model
VIN
Registration Date Renewal Date Exp Date
Drives
Driver
License Number
License Exp
Has Route
Dispatcher
Badge Number
Places Order
Time Placed
Num Orders
Orders_From
Order
Order ID
RouteID
Order Type: {Warehouse, Vending}
Route
RouteID
Date Created
Time Created
Warehouse
Warehouse ID
Address: Street Address, City State Zip
Manager Name
Capacity
Warehouse Receives
Time
Date
Order Contains
NumItemType
ItemTypePrice
Exp Date
Supplier
Supplier ID
Name
Address: Street Address, City State Zip
Phone
Item Type
Item Type ID
Item Type Name
MSRP
Warehouse Has
NumItemType
ItemTypePrice
ExpDate
Delivery
Date
Time Arrived
Time Spent
Invoice
InvoiceID
Date
Time
Creates Invoice
Vending Machine
Machine ID
Build Make Model Year
Items Per Slot
Capacity
Owner
Client
Customer ID
Name
Email
Phone
Company Name
Has
Location
Location Id
Address Street Address, City State Zip
Description
Items Sold
NumSold
PriceSold

2. Conceptual Database and Logical Database

The following chapter describes the relational model schema for our database. A brief history and explanation of the relational model will be given followed by a comparison between the ER model and the relational model. After that, the process of converting from an ER model to a relational model will be laid out. Following this, all relations will be detailed and a series of sample queries on the database will be given and answered.

2.1 E-R model and Relational Model

2.1.1 Description of the E-R model and Relational Model

History of ER Model

ER diagrams were modeled after Charles Bachman self-named Bachman Diagrams. Bachman's diagrams modeled data structures and recognized a need for a diagram that modeled higher abstraction. Bachman inspired fellow computer scientist Peter Chen. Chen published his interpretation for a database schema using Entities and Relationships. Chen's concepts of Entities and Relationships took off and over time the ER model was refined to become what it is today.

What is the ER Model

The Entity Relationship Model is a way to diagram a database schema at a high level of abstraction. There are two major components to an ER model, an Entity and a Relationship. An Entity represents some real word object or idea, such as a store, an employee, or an order of various items. A Relationship links two or more Entities together. Entities contain attributes which are descriptors of the Entity. An example: an Entity named Employee would have attributes Name, SSN, Gender. A Relationship can also have attributes. Lines connect symbols to form a diagram. It is also important to note cardinality between relationship, such as one to many or many to many.

Major Features of ER Model / Purpose

Entities represent real world objects or Ideas. Relationships link Entities together. Primary keys (detailed below) are used to distinguish entities from each other. This model is abstract and presents a clear, logical connection between data that is easy to read and understand.

History of the Relational Model

E.F. Codd, a computer scientist and employee at IBM conceived the relational database model. Codd recognized that other existing models had too much coupling between data and the physical storage of data. His relational model helped address this issue by decoupling the two mediums. Codd's model also eliminated duplicated data which lead to a very optimized schema.

Paired with SQL, a powerful query language, the relational model has become the most popular model for database design.

What is the Relational Model

The relational model takes real world objects or ideas and creates relations to represent them. A relation has a name and attributes (columns). A combination attributes, A_1, A_2...A_n form a tuple (row). A relation is made up of records, or tuples. An example of an employee relation tuple would look like → Employee(Name:String, Age:Integer, DOB: Date). Because relations consist of a set of records, they can be represented as tables where each row is a unique record. Similar to the ER Model, the Relational Model connects relations to form a database schema.

Major Features of the Relational Model / Purposes

Relation: A set of records that represent a real world object or idea.
Table: Contains a set of records that are linked to other sets of records.
Record: A tuple of data, contains all the information on one specific real world object or idea.
Field: Data that describes a record, such as a String or Integer.
Attribute: A description of a field, such as Gender or Age.
Primary Key: An attribute in a relation that can be used to uniquely identify a tuple in that relation. A Social Security Number is an example of a primary key. If more than two attributes make a PK, it is known as a composite key.
Foreign Key: A primary key that is placed in a different record and used to refer to the original record that contains the FK as a PK. The relation that holds the FK will also have its own PK.
Candidate Key: A key that helps narrow down a set of records. For example, an employee CK can be SSN and Department. However, a candidate key must be able to have all attributes that are not a PK removed and still be identifiable from other records.

A relational database stores data in tables rather than listing them as giant files of loosely related data. Having relations allows a computer to cross references tables and jump between data much faster than doing a sequential search and comparing all data. The features mentioned above are used to reference other tables and narrow down searches to specified conditions. Most importantly, a relational model produces a type of functional mapping. That is to say, a query is an expression that produces a relation. A set of inputs can be treated as sample point X, where X = {x_1, x_2 . . . x_n} and each x_n represents a condition or combination of conditions. An output can then be treated as sample point Y. The function $f(X) = Y$ can be used to visualize relational mapping at a very abstract level. Because this mapping is simple, powerful languages have been created to carry out queries extremely quickly. The combination of these languages and the versatile features of the relational model make this a great model to build a database schema.

2.1.2 Comparison of the Two Different Models

Advantages and Disadvantages of Relational versus ER Model

The relational model is the most widely used model for database implementation. There are several advantages that the relational model has that makes it a great tool for database design.

The following points present some of these advantages:

1. A relational model can be supported by powerful query languages, such as the widely used SQL language. These languages allow for complex databases to be created but still be manageable. The ER model has no language to support database implementation.
2. The relational model can represent multi-valued and complex attributes that the ER Model is NOT able to properly represent.
3. A relational model is known as a representational model, which shows how data is to be structured and organized. This has a lower level of abstraction than the ER Model which is conceptual. This reason is key because it allows languages like SQL to integrate with the relational model.
4. The relational model has only one 'data container' to keep track of, a relation(table) as opposed to the ER Models entities and relationships.

Listed below are a few of the disadvantages:

1. The ER Model is easier to understand as it presents a highly abstract view of the database schema. A user needs less knowledge about database design to understand the ER Model; the relational model packs data into many tables that can be complicated to diagram in one 'big picture.'
2. Relationships are explicitly shown in the ER Model. With the relational model it can be very difficult for users to jump from table to table to see how all the data is stored.

Differences and Similarities

Both databases are used to model a database schema. The two models have some 'container' for data organization (relations, entities). Each model has a way to show how data is related and through what other data. Despite having different ways of organizing data and displaying data, the goal of the two models is the same:

Model a database so that a DMBS can easily implement a collection of records.

The key differences between the two are the level of abstraction. A relational model is less abstract that the ER Model and is closer to the actual implementation and organization of record. An ER model is more abstract and can be read more like a diagram with flow of data rather than a storage model.

2.2 From Conceptual Database to Logical Database

The following section will describe in detail the methods and techniques used to convert a conceptual database to a logical database. For this database, the ER model from Chapter 1 will be converted to a relational database. First, a description will be given on how entity types are converted to relations. After that, the methods used to convert relationships to relations will be given, such as mapping. Lastly, the relational database constraints will be explained as well as their impact on the design process.

2.2.1 Converting Entity Types to Relations

Strong Entity

A strong entity is an entity that can exist without dependence on other entities. Converting this entity type is simple, create a relation that contains the same attributes as the entity including any keys.

Weak Entity

A weak entity can only exist if its parent entity also exist. A weak entity to relation conversion is very similar to a strong entity. The primary key of the strong relation should be the foreign key of the weak relation. For example, a Employee can have Dependents. Dependent would contain an Employee SSN and a Dependent SSN. The first key serves as a primary key and the second as a secondary (weak) key.

Simple vs Composite Attributes

A simple attribute is one that is made up of exactly one attribute. For example, the attribute Gender can be either M or F. A composite attribute is an attribute that is made up of several attributes. As an example, attribute Name is made up of Fname, MName, LName. To create a relation with a composite attribute, simply take the composite attributes and turn them into simple attributes.

Entity with composite value to relation

ER Model → NAME(SSN, NAME(FName,MName,LName), Gender)
Relational → nAME(SSN, FName, MName, LName, Gender)

Single vs Multi-valued attributes

A single valued attribute is an attribute that represents a real world idea or concept. Color is a good example of a single-valued attribute. A color can be any combination of colors (Composite) but the term color only means one thing, a color. A multi-valued attribute is an attribute that can be representative of many different real world objects or ideas. A phone number can be multi-valued if it can represent a **Home Phone, Mobile Phone,** or **Work Phone**. Each of these attributes would come from the same relation PHONE, which would eliminate the need to create a relation for each specific

phone type.
Multi-valued attributes must be turned into their own relations.

ER Model → Staff(ssn, name, phone)
Relational model → Staff(ssn, name)   1 to M
PHONE(ssn, phoneNum, phoneType)

2.2.2 Converting Relationship Types to Relations

One to One (1:1) conversion methods

- Foreign key approach is used to map a relation T's primary key to S as a foreign key, or vice versa. All simple attributes are included if the entity type has total participation.
- Merge the two entity types and relation into one single relation. It's possible to do this when there is total participation from both sides, and the two tables will have the same amount of attributes.
- Cross-reference is to set up a third relation, R, so that we can cross-reference the first two relations, S and T, where R is a "lookup table" and represents a relationship instance that relates one tuple from S with one tuple from T. R will contain the primary keys of S and T as foreign keys to S and T.

One to Many (1:M)

- Foreign key method can be used similar to the one-to-one method. The primary key of the 1-side side should be placed as a foreign key in the N-side because each entity instance on the N-side is related to at most one entity instance on the 1-side of the relationship type.
- The cross-reference method would be used if few tuples in *S* participate in the relationship to avoid excessive NULL values in the foreign key.

Many to Many (M:N)

- Foreign key method will be used to because many to many cannot be represented by a single foreign attribute in one relation, thus creating a new relationship relation.
- Cross-reference method is used because unlike the one to many relationships, many to many relationship have to have a separate relation that holds the primary keys of the first two relations and makes them foreign keys so that there is correspondence. A lookup table is necessary to find all the many instances that relation T can have of S.

IS-A and HAS-A

- These are specialized entities where entities are linked to a superclass entity and dependent entity into subclasses. For example, EMPLOYEE would be the superclass and DRIVER and DISPATCHER would be subclasses. The subclasses will take the attributes of the superclass and inherit them.
- Conversion Methods:

- Create relations for each of the subclasses and superclasses, have foreign keys in the subclasses that connect them to their superclasses, and keep their simple attributes. This works for any specialization.
    - Create a relation for each subclass. This only works for specializations whose subclasses are total participation or if it has a disjointedness constraint.
    - Create a single relation with a type attribute to indicate which subclass each tuple belongs to. This only works for specializations whose subclasses are disjoint.
    - Create a single relation with a Boolean type attribute that indicates whether a tuple belongs to a subclass. This is used only when the specialization have subclasses that overlap.

Recursive

- Recursive relationships are when an entity has a relationship with itself. So to convert it to a relation, we must create a foreign key to reference its own primary key. Take the EMPLOYEE entity for example. When converted over to a relation, we place the key attributes of DRIVER and DISPATCHER into it as foreign keys. When the information for those two needs to be retrieved, it'll be done so through itself. Thus, making it a recursive relation.

N-ary

- For the foreign key model where n > 2, create new relations S to represent R, include foreign keys in S that are the primary keys of R and the simple attributes of the relationship type as attributes of S where the primary key of S is a combination of the foreign keys. Repeat for N amount of relations.

Union type

- A subclass of the union of two or more superclasses that can have different keys because they can be of different entity types
- A surrogate key is used when a category's superclass have different keys; a new key attribute
    - Keys cannot be used to identify all entities in the category because they are keys of different defining classes

2.2.3 Database Constraints

Once an ER Model has been converted to a relational model it is important to consider the constraints that need to be placed on the database to ensure valid states always exist. To fully represent a complete relation schema, these rules are needed in order to set appropriate limitations on top of the collection of data.

Constraints are used to ensure users input data which satisfy all type constraint. When constraints are violated, DBM Systems should give the appropriate information.
Data must be able to be inserted, deleted, or updated and any data that violates the constraints described below would destroy mini-world concepts.

Examples of data manipulation that would fire constraint violations:

- Wage input by a company accountant is lower than minimum wage requirements.
- Two students of the same last and first name are assigned the same student ID Number.
- A teaching record of a new faculty being inserted when this faculty instance does not exist in the database system.

Entity Constraint:

Entity constraint, aka "Entity Integrity constraint" ensures that each tuple of the same relation is uniquely identified, such that it can be retrieved separately if needed. This concept forms the foundation of database design and implementation.

This is carried out by the implementation of primary keys. The value set for a primary key is unique from one another; each record in a relation is therefore uniquely identified.

Primary Key Constraint:

A primary key is used to identify individual records in a relation. In order to satisfy this requirement each tuple in a relation must be both non-null and unique.

Unique Key Constraint:

Columns other than the combination of primary keys that must be unique value while "Null" is allowed unless future specified.

Sometimes we will allow duplicates a single column, but applying unique constraint on the combination of columns. That is, any 2 record in the combination of such columns must at least have one unique column to identify each other.

Referential Constraints:

Referential constraints are checked when data in a record is modified or deleted. A referential constraint checks that modifying a certain record or set of records in relation T won't result in the altering of a record in different relations. The DMBS must check T's foreign keys and see which relations they reference and ensure that the data in other relations maintain a valid state.

Before a primary key is to be deleted, all foreign keys in other relations that reference this primary key must be deleted, and this applies to all the cascading levels. This is must be done to ensure that a key refers to an existing record. If foreign keys are deleted, the referenced primary keys must either be deleted or modified so they do not have null references.

The leaf node relation key must be deleted first and delete keys higher in the tree hierarchy.

Domain Constraints

Domain constraints are limitations placed on the attributes of a record. In order to ensure data integrity and ensuring valid states of the schema, domain constrains must be used to give attributes name significance. For example, the attribute name can either be a String or an Int. The number of allowed dependents might be an int that can never be greater than 10. These are left up to the designer to implement but are crucial in guaranteeing queries produce accurate expressions.

Business Constraints

Business constraints are defined by the limitations a business needs to place on the database schema that cannot be explicitly stated through other constraints. For instance, the minimum wage could be set to a legally required amount to ensure the company pays all workers the legal minimum amount.

Null Value Constraints

The null value constraint determines whether an attribute can have a null value by default or not. A primary key is an example of an attribute that must never have a null value.

2.3. Convert Your E-R/Conceptual Database into a Relational/Logical Database

The following section details each relation. Constraints, Keys, Attributes, Domains, and a description are provided for each as well.

A relational model cannot match an ER model exactly because they represent data differently. The conversion techniques described in the previous sections have been used to turn entities and relationships into relations.

2.3.1 Relation Schema for the Database

Employee

Constraints

Primary Key: SSN must be unique

Referential: Employee_ID must refer to an individual and real employee.

Business: None

Candidate Keys: Employee_ID, SSN, Name, Address, Phone, Position, Salary, Start Date

Description: This strong entity is turned into a relation with all original attributes minus address and phone.

Attribute	Domain	Description
Employee_ID	Int: Any x > 0 and unique	Number assigned to each employee by the manager for clock in/out tracking
SSN	Int: 000000000 - 999999999; unique	Distributed by the Social Security Department to uniquely identify persons.
FName	String: Any	First Name
MName	String: Any	Middle Name
LName	String: Any	Last Name
Position	String and not unique	The job the employee is assigned.
Phone	Int: 0000000000 -	The number to contact the

	9999999999	employee at.
Salary	Int: Any x > 0 and not unique	How much the employee has been contracted to make.
Start Date	Int(01-12), Int(01-31), Int(0000-9999)	The day the employee starts. (MM, DD, YYYY)
Badge Number	Int: Any x > 0 and unique	Number assigned to a dispatcher to identify length of employment with company and what orders they place and send out.
License Number	String: given by state dmv and unique	A California issued Class A driver's license is needed to operate vehicle. A unique 8-digit number makes a license number and uniquely identifies a driver.
License Exp	Int (01-12), int (01-31), int (0000-9999); not unique	Expiration date of the driver's license.
StreetName	String and not unique	The number and name of the street the employee lives at.
City	String and not unique	The city the employee resides in.
State	String and not unique	The state the employee lives in
ZIP	Int: 00000-99999	The zip code the employee

Vending Machine
Constraints
Primary Key**:** Machine_ID
Referential: Build_ID, Client_ID, Location_ID
Business: none

Candidate Keys: none

Description: The strong entity Vending_Machine is turned into a relation will all its attributes. This relation uses a Foreign Key approach to referencing Build, Client, Location.

Attribute	Domain	Description
Machine_ID	Integer: Any unique 10 digit combination	Identify a physical machine
Client_ID	Integer: Any unique 10 digit combination	Identify a single client, owner of this machine FK, referenced to Client
Location_ID	Integer: Any unique 10 digit combination	Identifies a location of this machine FK, referenced to Location
Build	String: Any	Identifies Machine Name (Brand)
Capacity	Int: X > 0	Max Capacity of Machine

Client

Constraints

Primary Key: Customer_ID must be unique

Referential: Customer_ID must refer to one real client.

Business: None

Candidate Keys: Customer_ID, Name, Company, Machines

Description: This strong entity is turned into a relation with the original attributes minus phone and address. It will use the foreign key approach to link to a machine and order.

Attribute	Domain	Description
ClientID	Int: Any x > 0 and unique	Unique number formed to identify each customer.
Email	String and unique	Email for the point of contact
FName	String and not unique	First name of the customer
LName	String and not unique	Last name of the customer
Phone	Int: 0000000000 - 9999999999	The number to contact the customer
Company Name	String and not unique	Name of the company that we refill machines for .
StreetName	String and not unique	The number and name of the street the client is located at.
City	String and not unique	The city name the client is located in
State	String and not unique	The state the client is located in
Zip	Int: 00000-99999; not unique	The zip code the client resides in

<u>Location</u>

Constraints

<u>Primary Key</u>**:** Order_ID

<u>Referential</u>: Wharehouse_ID, Supplier_ID, Lisence_Number

<u>Business</u>: The location where each vending machine seats.

Candidate Keys: non

Description: The strong entity Location is turned into a relation will all its attributes. This relation uses a Foreign Key approach to referencing Address

Attribute	Domain	Description
<u>Location_ID</u>	Integer: Any unique 10 digit combination	Identifies a location of this machine
StreetName	String and not unique	The number and name of the street the client is located at.
City	String and not unique	The city name the client is located in
State	String and not unique	The state the client is located in
Zip	Int: 00000-99999; not unique	The zip code the client resides in

Route

Constraints

Primary Key: RouteID

Referential: LicenseNumber must refer to an existing employee

Business: None

Candidate Keys: RouteID, LicenseNumber

Description: A route is created which links to many locations. A driver is assigned a delivery route.

Attribute	Domain	Description
RouteID	Integer: Any X > 0	Route identification
LicenseNumber	Integer: Any X > 0	Which driver is assigned the route
DateCreated	Date	When created
TimeCreated	Time	Time created

Delivery

Constraints

Primary Key**:** RouteID, LocationID, OrderID (compound)

Referential: RouteID, LocationID, OrderID must all refer to existing records

Business: None

Candidate Keys: RouteID, LocationID, OrderID

Description: A lookup table. Is used to link the locations that a route consist of and the orders that go to the locations.

Attribute	Domain	Description
RouteID	Integer: Any X > 0	Route identification
LocationID	Integer: Any X > 0	Which location belongs to this route
OrderID	Integer: Any X > 0	Which order goes to which location
Date	Date	Date of delivery
TimeArrived	Time	Time arrived at a location
TimeSpent	Time	Time spent at a location

Order

Constraints

Primary Key: Order_ID

Referential: Warehouse_ID, Supplier_ID, License_Number

Business: Orders are put in by Company Dispatcher in headquarter office

Candidate Keys: none

Description: The strong entity Order is turned into a relation will all its attributes. This relation uses a Foreign Key approach to referencing warehouse, supplier, and Driver.

Attribute	Domain	Description
OrderID	Integer: Any unique 10 digit combination	Identify a single order which was put in by dispatcher
OrderType	[Wharehouse_Order, Vending_Order] 2 values only	A boolean , 0 : for orders that reloading the warehouse from supplier 1: for order s that reloading the remote vending machines from warehouse
Warehuose_ID	Integer: 0 < x < 1000	Identify a sngle warehouse, FK, referenced to Warehouse
Supplier_ID	Integer: 0<x < 1000	Identify a single supplier, FK, referenced to Supplier
License_number	Integer: X > 0 and unique	Identifies which Driver drives this vehicle. FK, referenced to Driver

OrderConstaints

Constraints

Primary Key: Order_ID, ItemTypeID

Referential: Order_ID, ItemType_ID

Business: None

Candidate Keys: Order_ID

Description: A relationship relation, aka a lookup table is invented to bridge M to N relationship between Entity Order and Entity ItemType

Attribute	Domain	Description
OrderID	Integer: Any unique 10 digit combination	Identify a single order which was put in by dispatcher
ItemTypeID	Integer: Any unique 10 digit combination	Identify which single type of item was ordered
NumItemType	1~999	Number of item types. How many packs of soda ? (35 cans, 12 oz)
ItemTypePrice	Float 0.01 ~ 999.99	The contracted purchase price we purchased from Major Suppliers per (35 cans, 12oz) pack
ExpDate	Date: (1950/1/1 ~ 2049/12/31)	The last legal date this drink may be sold to customer

ItemType
Constraints
Primary Key**:** Item_Type_ID
Referential: non
Business: Non
Candidate Keys: Non

Description: The strong entity Item_Type is turned into a relation will all its attributes.

Attribute	Domain	Description
ItemTypeID	Integer: Any unique 10 digit combination	Identify a single type of item which may be stocked and later put to vending machines
SuppierID	Int: X > 0	Which supplier the Item came from
Name	String : 2 ~ 16 charachter	The name of this Item
MSRP	Integer: 0 < x < 1000	Market suggested retail price

<u>Warehouse</u>

Constraints

<u>Primary Key</u>**:** Warehouse_ID must be unique

<u>Referential</u>: Manager_SSN must refer to a real employee

<u>Business</u>: Capacity is set by warehouse manager

Candidate Keys: Warehouse_ID

Description: The strong entity Warehouse is turned into a relation minus it's address attribute. A FK mapping is between warehouse to employee, order. Employee PK is a FK in warehouse and warehouse PK is used as a FK in order.

Attribute	Domain	Description
<u>WarehouseID</u>	Integer: X > 0 and unique	Uniquely identifies a warehouse
<u>SSN</u>	Integer: Any unique 9 digit combination >= 0	Links to Employee who manages warehouse
Capacity	Integer: X >= 0	Max Item Types allowed in a warehouse
StreetName	String: Any	Name of street
City	String: Any	City location
State	String: Any	State location
Zip	Integer: Any unique 5 digit X >= 0	Zip code

WarehouseHas

Constraints

Primary Key: WarehouseID, ItemTypeID (Compound)

Referential: WarehouseID, ItemTypeID must refer to existing records

Business: None

Candidate Keys: WarehouseID, ItemTypeID

Description: A lookup table that is used to determine warehouse inventory. Links to a warehouse and to the item type that is sold.

Attribute	Domain	Description
WarehouseID	Integer: Any X > 0	Warehouse identification
ItemTypeID	Integer: Any X > 0	Which Items are being referred to
NumItemType	Integer: Any X > 0	Num of items sold
ItemTypePrice	Integer: Any X > 0	Price of item
ExpDate	Date	When item expires

WarehouseReceives

Constraints

Primary Key**:** OrderID, WarehouseID (Compound)

Referential: Both primary keys must refer to existing records

Business: None

Candidate Keys: OrderID, WarehouseID

Description: Track which orders are received by a warehouse. This information is useful when assessing warehouse performance and accounting for shipments.

Attribute	Domain	Description
OrderID	Integer: Any X > 0	Which order is received
WarehouseID	Integer: Any X > 0	Which warehouse receives
TimeRec	Time	Time order received
DateRec	Date	When order received

Supplier

Constraints

Primary Key: Supplier_ID must be unique

Referential: Supplier_ID must refer to one real supplier.

Business: None

Candidate Keys: Supplier_ID, Name

Description: This strong entity is turned into a relation with one original attribute.

Attribute	Domain	Description
SupplierID	Int: Any x > 0 and unique	Unique supplier identifier
Name	String and not unique	The name of the supplier.
Phone	Int: 0000000000 - 9999999999	The number to contact the supplier at
StreetName	String and not unique	The street number and name the supplier is located at.
City	String and not unique	The city the supplier is located in.
State	String and not unique	The state the supplier is located in.
ZIP	Int: 00000-99999	The zip code the supplier is located in.

<u>Gas Receipt</u>
Constraints
<u>Primary Key</u>**:** Driver
<u>Weak Key</u>: Receipt_ID
<u>Referential</u>: PK Driver must point to an existing employee tuple.
<u>Business</u>: Receipts are to be collected by a driver and turned into corporate.

Candidate Keys: Driver

Description: The weak entity Gas Receipt is converted into a relation will all original attributes minus an address. A FK key mapping is used to link this relation to Driver.

Attribute	Domain	Description
<u>LicenseNumber</u>	Integer: Any unique 9 digit number	Which Employee has interacted with this receipt
<u>Recept_ID</u>	Integer: Any positive	Uniquely ID receipt
StreetName	String: Any	Name of street
City	String: Any	City of warehouse locat
State	String: Any	State location
Zip	Integer: Any unique 5 digit X >= 0	Zip code
Total_price	Float: X > 0	Total gas purchase price
Date	Date	Date of purchase

Invoice
Constraints
Primary Key**:** InvoiceID
Referential: MachineID must point to a real Machine record
Business: Will be generated electronically by machine.

Candidate Keys: InvoiceID

Description: An invoice is a report generated by a machine upon the request of a driver or employee. The generated report includes items sold and helps determine profit.

Attribute	Domain	Description
InvoiceID	Integer: Any X > 0	ID of Invoice
MachineID	Integer: Any X > 0	Machine that created invoice
Date	Date	Date of creation
Time	Time	Time of creation

ItemsSold
Constraints
Primary Key**:** ItemTypeID
Referential: ItemTypeID and MachineID must match existing records
Business: None

Candidate Keys: ItemTypeID, InvoiceID

Description: A look up table to track how many items are sold per invoice

Attribute	Domain	Description
ItemTypeID	Integer: Any X > 0	Which Item is Sold
InvoiceID	Integer: Any X > 0	From which invoice
Num	Integer: Any X > 0	Num of items sold
PriceSold	Integer: Any X > 0	Price sold at

PlacesOrder
Constraints
Primary Key**:** OrderID, BadgeNumber
Referential: OrderID and BadgeNumber must refer to existing records
Business: Only Employee who is a dispatcher can place an order

Candidate Keys: OrderID, BadgeNumber

Description: A look up table to track orders placed by a dispatcher.

Attribute	Domain	Description
OrderID	Integer: Any X > 0	Identifies unique order
BadgeNumber	Integer: Any X > 0	Dispatcher who placed order
TimePlaced	Time	Time order placed
DatePlaced	Date	Date order placed

Vehicle
Constraints
Primary Key**:** PlateNumber, LicenseNumber
Referential: PlateNumber and LicenseNumber must match existing records
Business: Vehicles can only be driven by driver Employees

Candidate Keys: PlateNumber, LicenseNumber

Description: This strong entity is converted to a relation and is assigned the foreign key that is the license number of an Employee.

Attribute	Domain	Description
PlateNumber	Integer: Any X > 0	License Plate Number of Vehicle
LicenseNumber	Integer: Any X > 0	Employee who drives the vehicle
Make	String: Any	Which vehicle (brand)
Model	String: Any	Type of vehicle
VIN	Int: Any unique 17 combo	Unique manufacturer number

2.3.2. Sample Data of Relation

Employee(SSN, FName, LName, Phone, Position, Salary, StartDate, StreetName, City, State, Zip, License Number, License Exp, Badge Number)

HasRoute(LicenseNumber, RouteID)

Vending Machine(MachineID, ClientID, LocationId, Build, Capacity)

Client(ClientID, FName, LName, Email, Phone, StreetName, City, State, Zip, Company Name)

Location(LocationID, StreetName, City, State, Zip, Description)

Route(RouteID, LicenseNumber, DateCreated, TimeCreated)

Delivery(RouteID, LocationID, OrderID, Date, TimeArrived, TimeSpent)

Order(OrderID, SupplierID, OrderType)

OrderContains(ItemTypeID, OrderID, NumItemType, ItemTypePrice, ExpDate)

ItemType(ItemTypeID, SupplierID, ItemTypeName, MSRP)

Warehouse(WarehouseID, ManagerSSN, Capacity, StreetName, City, State, Zip)

WarehouseHas(WarehouseID, ItemTypeID, NumItemType, ItemTypePrice, ExpDate) **//wait**

WarehouseReceives(OrderID, WarehouseID, TimeRec, DateRec)

OrdersFrom(SSN, SupplierID, DateOrdered, TimeOrdered)

PlacesOrder(OrderID, BadgeNumber, TimePlaced, DatePlaced) //here

Supplier(SupplierID, Name, Phone, StreetName, City, State, Zip)

GasReceipt(LicenseNumber, ReceiptID, StreetName, City, State, Zip, Total Price, Date)

Invoice(InvoiceID, MachineID, Date, Time)

ItemsSold(ItemTypeID, InvoiceID, NumSold, PriceSold)

Vehicle(PlateNumber, LicenseNumber, Make, Model, VIN)

2.4. Sample Queries to our Database

Detailed below are 10 sample queries based on the newly designed relational database. Each query is answered in three academic query languages:

Relational Algebra, Tuple Relational Calculus, Domain Relational Calculus

2.4.1 Design Of Queries

1. List the SSN and Name of all Employees who are drivers.
2. List SSN, Name, and Start Date of all Driver's who have started in the past year.
3. List all Clients who own at least two machines.
4. List Clients who own only one machine.
5. List all Machines that stock at least two ItemTypes.
6. List all Drivers who have delivered to every location
7. List all Dispatchers who have ordered from all suppliers.
8. List all Driver who delivered to CSUB between Jan 2018 and Jan 2019.
9. Invoice that contains the least amount of items sold.
10. Invoice that contains the second least amount of items sold.

2.4.2 Relational Algebra Expressions for Queries of 4.1

Relational algebra is a procedural query language that outputs a relational expression. A procedural query language is one that relies on the state of a expression and the steps that are taken to attain the desired state. A select statement, denoted by the symbol σ, is used to select records with specified attributes. A project statement, denoted by π, is used to select attributes of a relation (columns). If there is a cross-product of two or more relations it is important to select attributes that can be used to combine the many relations. Because select and project statements act on relational expressions, it is possible to combine many select and project statements to single out a specific relational expression.

1. List the SSN and Name of all Employees who are drivers.

   $\pi_{SSN,Name}(\sigma_{position\ =\ Driver}(Employee))$

2. List SSN, Name, and Start Date of all Driver's who have started in the past year.

   $Drivers \leftarrow (\sigma_{position\ =\ Driver\ \wedge\ StartDate\ >=\ (Today\ -\ 1\ year)}(Employee))$

   $\pi_{SSN,\ Name,\ StartDate}\ (Drivers)$

3. List all Clients who own at least two machines.

   ClientTwoM ← $\sigma_{c.ClientID\ =\ m1.ClientID\ \wedge\ c.ClientID\ =\ m2.ClientID\ \wedge\ m1.MachineID\ !=\ m2.MachineID}$(Client X Vending Machine X Vending Machine)

   π_*(Client * ClientTwoM)

4. List Clients who own only one machine.

   ClientTwoM ← $\sigma_{c.ClientID\ =\ m1.ClientID\ \wedge\ c.ClientID\ =\ m2.ClientID\ \wedge\ m1.MachineID\ !=\ m2.MachineID}$(Client X Vending Machine X Vending Machine)

   π_*(Client * (Client - ClientTwoM))

5. List all Machines that stock at least two ItemTypes.

   VMTwo ← $\sigma_{vm.MachineID\ =inv.MachineID\ \wedge\ itm1.InvoiceID\ =\ inv.InvoiceID\ \wedge\ itm2.InvoiceID\ =\ inv.InvoiceID\ \wedge\ itm1.ItemTypeID\ !=\ itm2.ItemTypeID\ \wedge\ Date\ >=\ Present)}$ (Vending Machine X Invoice X ItemsSold/Needed X ItemsSold/Needed)

   $\pi_{(*)}$(VMTwo)

6. List all Drivers who have delivered to every location

   Driver ← $\sigma_{Position\ =\ Driver}$(Employee * Route * Delivery)

   $\pi_{(SSN,\ Name,\ LicenseNumber)}$(Driver) % $\pi_{(LicenseNumber)}$(Location)

7. List all Dispatchers who have ordered from all suppliers.

   Dispatchers ← $\sigma_{Position\ =\ Dispatcher}$(Employee * OrdersFrom)

   $\pi_{(Name,\ BadgeNumber,\ SupplierID)}$(Dispatchers) % $\pi_{(SupplierID)}$(Supplier)

8. List all Driver who delivered to CSUB between Jan 2018 and Jan 2019. (TODO)

   TEMP ← (Employee*Route*Delivery)

   $\pi_{SSN,Name,LicenseNumber}$($\sigma_{t.LocationID\ =\ l.LocationID\ \wedge\ l.Address\ =\ CSUB}$(TEMP X Location)

9. Invoice that contains the least amount of items sold.

   Invoice * (ItemsSold - $\pi_{(i1.*)}$($\sigma_{i1.NumSold\ >\ i2.NumSold}$ (ItemsSold X ItemsSold)))

10. Invoice that contains the second least amount of items sold.

    Invoice * (ItemsSold - $\pi_{(r1.*)}$($\sigma_{i1.NumSold\ >\ i2.NumSold\ \wedge\ i2.NumSold\ >\ i3.NumSold}$(ItemsSold X ItemsSold X ItemsSold)))

2.4.3 Tuple Relational Calculus Expressions for Queries

Tuple relational calculus is a non-procedural query language that focuses on what to do rather than how to do it. We define a tuple variable, specify the relation that the tuple is in and the condition. By doing this, we give a description of the query on how to get to the result. Tuple calculus has both bounded and free variables that specify whether the condition will remain the same or change throughout time.

1. List the SSN and Name of all Employees who are drivers.
   {e.SSN, e.Name | Employee(e) ^ e.position = Driver}
2. List SSN, Name, and Start Date of all Driver's who have started in the past year.
   {e.SSN, e.Name, e.StartDate | Employee(e) ^ e.position = Driver ^ e.StartDate >= (Today - 1 year)}

3. List all Clients who own at least two machines.
   {c | Client(c) ^ (∃m1)(∃m2) (VendingMachine(m1) ^ VendingMachine(m2) ^ c.ClientID = m1.MachineID ^ c.ClientID = m2.MachineID ^ m1.MachineID != m2.MachineID)}
4. List Clients who own only one machine.
   {c | Client(c) ^ (∃m1)(∃m2) (VendingMachine(m1) ^ VendingMachine(m2) ^ c.ClientID = m1.MachineID ^ c.ClientID = m2.MachineID ^ m1.MachineID != m2.MachineID)}
5. List all Machines that stock at least two ItemTypes.
   {m | VendingMachine(m) ^ (∃i)(∃itm1)(∃itm2) (Invoice(i) ^ ItemsSold(itm1) ^ ItemsSold(itm2) ^ m.MachineID =i.MachineID ^ itm1.InvoiceID = i.InvoiceID ^ itm2.InvoiceID = i.InvoiceID ^ itm1.ItemTypeID != itm2.ItemTypeID ^ Date >= Present)}
6. List all Drivers who have delivered to every location
   {m | VendingMachine(m) ^ (∃i)(∃itm1)(∃itm2) (Invoice(i) ^ ItemsSold(itm1) ^ ItemsSold(itm2) ^ m.MachineID =i.MachineID ^ itm1.InvoiceID = i.InvoiceID ^ itm2.InvoiceID = i.InvoiceID ^ itm1.ItemTypeID != itm2.ItemTypeID ^ Date >= Present)}
7. List all Dispatchers who have ordered from all suppliers.
   {e | Employee(e) ^ e.Position = Dispatcher ^ (∀s)(Supplier(s) → (∃o)(OrdersFrom(o) ^ o.BadgeNumber = e.BadgeNumber ^ o.SupplierID = s.supplierID))}

8. List all Driver who delivered to CSUB between Jan 2018 and Jan 2019.
   {e | Employee(e) ^ e.Position = Driver ^ (∃r)(∃d)(∃l)(Route(r) ^ Delivery(d) ^ Location(l) ^ cCSUB)}
9. Invoice that contains the least amount of items sold.
   {i | Invoice(i) ^ (∃s)(ItemsSold(s) ^ s.InvoiceID = i.InvoiceID ^
   ¬(∃s2)(ItemsSold(s2)
   ^ s2.NumSold < s.NumSold))}
10. Invoice that contains the second least amount of items sold.
   {i | Invoice(i) ^ (∃s)(ItemsSold(s) ^ s.InvoiceID = i.InvoiceID ^
   (∃s2)(ItemsSold(s2)
   ^ s2.NumSold < s.NumSold ^ ¬(∃s3)(ItemsSold(s3) ^ s3.NumSold < s.NumSold ^ s3.NumSold != s2.NumSold)))

2.4.4 Domain Relational Calculus Expressions for Queries

Domain relational calculus is a procedural query language and is very similar to tuple relational calculus. Filtering is done based on the domain of the attribute and not on the tuples value. A relational expression is returned if the relation has an attribute that matches the domain of the free variables. What this means is rather than specifying that a specific relation with specific attributes exist, domain calculus specifies that there exists one or many relations that match one or more of the free variables. Similar to tuple calculus, existential or universal quantifiers can be used.

1. List the SSN and Name of all Employees who are drivers.
   {<s,f,l,p> | Employee(s,f,l,p) ^ p = Driver}
2. List SSN, Name, and Start Date of all Driver's who have started in the past year.
   {<ssn,fName,lName,Position,StartDate> |
   Employee(ssn,fName,lName,Position,DateStart) ^ StartDate >= (Today - 1 year)}
3. List all Clients who own at least two machines.
   {<ClientID,FName,LName> | Client(ClientID,FName,LName) ^
   (∃m1)(VendingMachine(m1,ClientID) ^ VendingMachine(!=m1,ClientID))}
4. List Clients who own only one machine.
   {<ClientID,FName,LName> | Client(ClientID,FName,LName)
   (∃m1)(∃m2)(VendingMachine(m1,ClientID) ^¬(Machine(m1!=m2,ClientID))}
5. List all Machines that stock at least two ItemTypes.
   {<ClientID,FName,LName> | Client(ClientID,FName,LName)
   (∃m1)(∃m2)(VendingMachine(m1,ClientID) ^¬(Machine(m1!=m2,ClientID))}
6. List all Drivers who have delivered to every location
   {<s,f,l,d> | Employee(s,f,l,d) ^ Route(r,d) ^ (∀l)(Delivery(r,l)}

7. List all Dispatchers who have ordered from all suppliers. l
   {<s,f,l,p> | Employee(s,f,l,p) ^ Suppler(sd) ^ (∀sd2)(OrdersFrom(s,sd=sd2))}
8. List all Driver who delivered to CSUB between Jan 2018 and Jan 2019.
   {<s,f,l,p> | Employee(s,f,l,p) ^ (∃r)(Route(r,p) ^ (∃l)(Delivery(r,l) ^ (∃a)(Location(l,a) ^ a = csub)))}
9. Invoice that contains the least amount of items sold.
   {<i,m,> | Invoice(i,m) ^ (∃n)(ItemsSold(i,n) ^ ¬(∃n2)(ItemsSold(n2) ^ n2 < n))}
10. Invoice that contains the second least amount of items sold.
   {<i,m,> | Invoice(i,m) ^ (∃n)(ItemsSold(i,n) ^ (∃n3)(ItemsSold(n3) ^ n3 < n ^¬(∃n2)(ItemsSold(n2) ^ n2 < n ^ n3!=n2))}

3 Implementation of Relational Database

3.1 Relation Normalization

3.1.1 Normalizations

Normalization is a methodical mathematical approach, which has been proven to minimize data redundancy in a relational database system. Mathematically, the normalization process is divided into 1st degree, 2nd degree, 3rd degree, Boyce-Codd, and 4th degree. Boyce-Codd is a higher normalized form than 3rd degree , but need not satisfy 4th degree normalization. In most industry relations, Boyce-Codd normalized form is considered sufficiently met the data normalized requirements. Fourth Degree form is optional in most line of business.

From the Entity-Relationship Model down to Relational Model, the originally goal was to use only one entity to represent one object in mini world. This object may be concrete item or abstract ideas. Then, use relationships to represent additional information flow in between 2 or 3 Entities. Nonetheless, this process does very little to avoid data redundancy. Redundancy occupies unnecessary storage as well as causes maintenance issue in a database system.

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

First Normal Forms:

Relation is in at least 1st normal form if it only has **singled valued attributes**. For a cell, which represent an attribute of single tuple in this relation, must always be atomic.

The 2nd, 3rd, BCNF, and 4th Normalization all deals among the columns within the same relation.

At all these levels, a collection of columns of a relation is redundant if all the records in these columns can be derived from another collection of columns in this relation. In other words, they are "functionally dependent" on other columns.

Second Normal Forms:

Second Normal Forms exist if and only if "No partial dependency". Partial Dependency, happens only when we have composite primary key. For relations with single column primary key, it automatically satisfy 2nd normal forms.

For relations with composite primary key, If a set of "non-prime " attributes are dependent on only a portion of the composite primary key, this is called "Partial Dependency."

There are some observation of rules here:

- If the relation is a lookup table, it has no “non-prime” attribute, there is no partial dependency => 2^{nd} Normalization satisfied

  Ex :

  OrdersFrom (SSN , SupplierID)

Third Normal Forms:

Third Normal Forms exist if and only if “No transitive dependency”.

A->B , B->C , if both are true, then A->C .

Attribute C can be determined by both attribute B and attribute A. By nature, a transitive dependency requires 3 or more columns (this means 2 column relation satisfy 3^{rd} normalization by default).

Ex :

Client (ClientID, FName, LName, Email, Phone , StreetName, City, State, Zip, Company Name)

ClientId -> Phone, Phone -> Zip, Zip-> State

After 3^{rd} Normalization:

Client (ClientID, FName, LName, Email, Phone , Company Name, AddressID)

Address(AddressID, StreetName, City, State, Zip)

Client has new foreign key , AddressID , referencing Address (AddressID)

Boyce-Codd Normal Forms:

Boyce-Codd Normal Forms exist if and only if

“for all Functional dependency in Relation R,

A -> B , A is a valid candidate key , ”.

The left hand side of all functional dependency of a Relation must be a candidate key. Boyce-Codd Normal Form is guaranteed to have no redundancy caused by. Functional dependency.

3.3.3 Anomalies

For relations yet to normalized, data comes data anomaly. Insertion anomaly occurs when certain attributes cannot be inserted into the database without the presence of other attribute in the same relation. Deletion anomaly occurs when attributes cannot be deleted from the database without the deletion of other attribute in the same relation. An update anomaly = insertion anomaly + deletion anomaly; redundancy caused data anomaly. BCNF is the minimum requirement to thoroughly eliminate all data modification anomalies.

3.2 PostgreSQL

3.2.1 Purpose

PostgreSQL is a object-relational database management system that has user-defined types, table inheritance, foreign key referential integrity, and allows the user to add custom functions developed using different languages. It's main purpose is serve as the database backend where queries are ran to insert, cross, or select data.

3.2.2 Schema Objects for PostgreSQL Database

Table

Postgres uses tables as a unit of data storage. Relations are turned into tables and their attributes are used as columns. The data inserted into those columns have their own data types, which are determined by what is to be stored in that column, and they create rows. The data can be inserted, updated, deleted or queried on demand.

Syntax:

```
CREATE TABLE tablename(
	column1 datatype,
	column2 datatype,
	column3 datatype,
		.....
);
```

Views

View is a virtual table used to simplify complex queries and to apply security for a set of records. When forming, we create a query and assign it a name. This makes it useful for wrapping a commonly used complex query. Views are read only because the system doesn't allow it to do anything else.

Syntax:

```
CREATE VIEW name AS
        SELECT *
        FROM ______
        WHERE ________ ;
```

Trigger

A trigger is a specification that the database must automatically execute a specific function whenever a specific type of operation is performed. They can be attached to both views and tables. They can execute before or after an INSERT, UPDATE, or DELETE operation on tables. And they can be set to execute in place of an INSERT, UPDATE, or DELETE operation. The function it's triggering must be defined before the trigger can be created, and it must be declared as a function taking no arguments and returning type.

Syntax:

```
CREATE [  OR REPLACE  ] TRIGGER trigger_name
{  BEFORE  |  AFTER  |  INSTEAD OF  }
{  INSERT  [ OR ]  | UPDATE [ OR ]  |  DELETE  }
[ OF col_name ]
ON table_name
[ REFERENCING OLD AS o NEW AS n ]
[ FOR EACH ROW ]
WHEN (condition)
BEGIN
        --- statements
END;
```

Indexes

The index approach is similar to those in books where the back of the book has a collection of terms and where they appear in the book. Once an its created, the programmer doesn't have to intervene as much because the system will update the index when the table is modified and use the index in queries when it sees it best fit over a sequential search.

Syntax:

```
CREATE INDEX test_b_index ON test (b);
```

Stored Procedures

Stored procedures are user-defined functions. These procedures take in parameters and perform some logic on them. They allow for database functionality to be extended as it allows data to be manipulated beyond the standard SQL statements. Stored procedures are used to create triggers or aggregate functions. These functions are also pre-compiled in PostgresSQL and as a result reduces the amount of round trips an application has to make to a database server. However, it is important to note that stored procedures are highly specialized and should be handled by experience database engineers to avoid unmaintainable procedures.

<u>Syntax</u>

```
CREATE FUNCTION function_name(parameters)
	RETURNS type AS
	BEGIN
		--logic
	END;
```

3.3 Relational Schema Data

3.3.1 Employee

```
vendingmachine=# \d employee
                              Table "public.employee"
   Column   |  Type   | Collation | Nullable |                   Default
------------+---------+-----------+----------+-----------------------------------------------
 employeeid | integer |           | not null | nextval('employee_employeeid_seq'::regclass)
 fname      | text    |           | not null |
 lname      | text    |           | not null |
 phone      | text    |           |          |
 position   | text    |           | not null |
 salary     | money   |           |          |
 sdate      | date    |           | not null |
 streetname | text    |           | not null |
 city       | text    |           | not null |
 state      | text    |           |          |
 zip        | integer |           |          |
 lnum       | integer |           |          |
 bnum       | integer |           |          |
 email      | text    |           |          |
 ssn        | text    |           | not null |
Indexes:
    "employee_pkey" PRIMARY KEY, btree (employeeid)
    "employee_ssn_key" UNIQUE CONSTRAINT, btree (ssn)
```

employeeid	fname	lname	phone	position	salary	sdate	streetname	city	state	zip	lnum	bnum	email	ssn
1	Nadiya	Sleightholme	858-121-5457	Dispatcher	$87,018.92	2012-12-13	1842 Ridge Oak Hill	San Diego	CA	92105	12000	1	nsleightholme0@bbc.co.uk	444-87-3220
3	Pascal	Radbourne	415-583-1366	Driver	$68,600.87	2008-07-20	46 Sommers Point	San Francisco	CA	94159	34000	3	pradbourne2@tiny.cc	384-48-7106
4	Brittani	Tuxsell	714-645-1073	Dispatcher	$18,727.07	2007-05-06	80 Green Court	Orange	CA	92862	45000	4	btuxsell3@alibaba.com	588-41-0157
5	Alexis	Treece	661-620-2794	Driver	$18,186.96	2014-02-20	0490 Nancy Alley	Bakersfield	CA	93399	56000	5	atreece4@baidu.com	241-07-2426
6	Jobey	Shemmans	909-729-9475	Driver	$30,795.37	2004-05-24	1996 Erie Court	San Bernardino	CA	92424	67000	6	jshemmans5@woothemes.com	114-33-9719
7	Maurits	Chrismas	619-507-8595	Dispatcher	$17,554.17	2009-04-28	052 Dahle Plaza	San Diego	CA	92191	78000	7	mchrismas6@kickstarter.com	231-89-1445
8	Huntlee	Huntress	805-109-9870	Driver	$76,534.54	2017-02-13	71373 Pine View Alley	Santa Barbara	CA	93106	89000	8	hhuntress7@blogs.com	223-32-5721
9	Etienne	Waddington	562-188-2684	Manager	$10,423.81	2001-04-30	59 Barby Point	Long Beach	CA	90847	98000	9	ewaddington8@salon.com	296-06-0266
10	Cheston	Thompkins	626-145-7789	Dispatcher	$79,168.62	2008-03-11	982 Esker Road	Anaheim	CA	92805	10000	10	cthompkins9@1688.com	774-58-1056
2	Giulia	McSorley	916-134-2215	Driver	$77,363.26	2019-01-01	532 Washington Parkway	Sacramento	CA	94297	23000	2	gmcsorley1@go.com	221-52-7822

(10 rows)

3.3.2 VendingMachine

```
vendingmachine=# \d vendingmachine
                              Table "public.vendingmachine"
    Column    |  Type   | Collation | Nullable |                        Default
--------------+---------+-----------+----------+-------------------------------------------------------
 machineid    | integer |           | not null | nextval('vendingmachine_machineid_seq'::regclass)
 clientid     | integer |           | not null | nextval('vendingmachine_clientid_seq'::regclass)
 locationid   | integer |           | not null | nextval('vendingmachine_locationid_seq'::regclass)
 build        | text    |           |          |
 itemsperslot | integer |           | not null |
 capacity     | integer |           | not null |
Indexes:
    "vendingmachine_pkey" PRIMARY KEY, btree (machineid)
```

machineid	clientid	locationid	build	itemsperslot	capacity
1	1	10		10	610
2	7	10		15	467
3	10	8		10	200
4	8	8		20	631
5	5	8		15	786
6	8	1		10	639
7	2	5		15	678
8	10	7		20	205
9	8	9		15	544
10	10	8		15	540

(10 rows)

3.3.3 Client

```
                              Table "public.client"
   Column    |  Type   | Collation | Nullable |                  Default
-------------+---------+-----------+----------+-------------------------------------------
 clientid    | integer |           | not null | nextval('client_clientid_seq'::regclass)
 fname       | text    |           | not null |
 lname       | text    |           | not null |
 email       | text    |           |          |
 phone       | text    |           | not null |
 streetname  | text    |           | not null |
 city        | text    |           | not null |
 state       | text    |           |          |
 zip         | text    |           | not null |
 compname    | text    |           |          |
Indexes:
    "client_pkey" PRIMARY KEY, btree (clientid)
```

```
vendingmachine=# select * from client;
 clientid |   fname    |   lname   |           email            |    phone     |        streetname         |     city      | state |  zip  | compname
----------+------------+-----------+----------------------------+--------------+---------------------------+---------------+-------+-------+----------
        1 | Garald     | De Cruz   | gdecruz0@bloomberg.com     | 805-346-7783 | 65 Elka Circle            | San Mateo     | CA    | 94405 | Mydo
        2 | Freddi     | Floris    | ffloris1@nytimes.com       | 951-240-5919 | 9948 Sundown Drive        | Riverside     | CA    | 92519 | Gigabox
        3 | Clarance   | Grubbe    | cgrubbe2@yelp.com          | 310-654-8928 | 4 American Ash Circle     | Los Angeles   | CA    | 90071 | Vitz
        4 | Gabriello  | Cantrell  | gcantrell3@xrea.com        | 415-395-6080 | 9 Havey Street            | San Francisco | CA    | 94137 | Quatz
        5 | Tomi       | Sirrell   | tsirrell4@examiner.com     | 707-160-8061 | 38289 Loftsgordon Place   | Petaluma      | CA    | 94975 | Mymm
        6 | Maisey     | Hadenton  | mhadenton5@friendfeed.com  | 714-515-3857 | 31 Arrowood Trail         | Anaheim       | CA    | 92825 | Kare
        7 | Aimee      | Pittel    | apittel6@furl.net          | 559-307-4377 | 29749 New Castle Parkway  | Fullerton     | CA    | 92640 | Centimia
        8 | Kirby      | Clurow    | kclurow7@gnu.org           | 559-737-5033 | 5 Rusk Drive              | Fresno        | CA    | 93773 | Eayo
        9 | Micky      | Cantrell  | mcantrell8@squidoo.com     | 916-395-6954 | 19 Moose Lane             | Sacramento    | CA    | 95852 | Dynabox
       10 | Moira      | Beange    | mbeange9@smugmug.com       | 626-372-9260 | 155 Dixon Junction        | Pasadena      | CA    | 91199 | Tekfly
(10 rows)
```

3.3.4 Location

```
                              Table "public.location"
   Column    |  Type   | Collation | Nullable |                   Default
-------------+---------+-----------+----------+-----------------------------------------------
 locationid  | integer |           | not null | nextval('location_locationid_seq'::regclass)
 streetname  | text    |           | not null |
 city        | text    |           | not null |
 state       | text    |           | not null |
 zip         | integer |           | not null |
 description | text    |           |          |
Indexes:
    "location_pkey" PRIMARY KEY, btree (locationid)
```

```
vendingmachine=# select * from location;
 locationid |       streetname       |    city     | state |  zip  | description
------------+------------------------+-------------+-------+-------+-------------
          2 | 18459 Butternut Pass   | Pensacola   | FL    | 32575 |
          3 | 1 Garrison Street      | Sparks      | NV    | 89436 |
          4 | 51872 Hoffman Center   | Denver      | CO    | 80255 |
          5 | 9 Myrtle Junction      | Bellevue    | WA    | 98008 |
          6 | 948 Cardinal Place     | Greensboro  | NC    | 27455 |
          7 | 620 Washington Drive   | Las Vegas   | NV    | 89140 |
          8 | 5950 John Wall Point   | Baltimore   | MD    | 21265 |
          9 | 18096 Elmside Way      | Pittsburgh  | PA    | 15205 |
         10 | 21380 Hintze Lane      | Maple Plain | MN    | 55572 |
          1 | 89425 Kipling Street   | Baton Rouge | LA    | 70883 | csub
(10 rows)
```

3.3.5 Route

```
vendingmachine=# \d route
                                  Table "public.route"
    Column     |          Type          | Collation | Nullable |                Default
---------------+------------------------+-----------+----------+----------------------------------------
 routeid       | integer                |           | not null | nextval('route_routeid_seq'::regclass)
 licensenumber | integer                |           | not null |
 datecreated   | date                   |           | not null |
 timecreated   | time without time zone |           | not null |
Indexes:
    "route_pkey" PRIMARY KEY, btree (routeid)
```

```
vendingmachine=# select * from route;
 routeid | licensenumber | datecreated | timecreated
---------+---------------+-------------+-------------
       1 |         12000 | 2018-05-24  | 02:12:00
       2 |         56000 | 2017-07-26  | 02:15:00
       3 |         23000 | 2018-08-22  | 18:18:00
       4 |         78000 | 2018-06-20  | 17:56:00
       5 |         89000 | 2018-02-05  | 12:13:00
       6 |         23000 | 2018-10-30  | 09:53:00
       7 |         12000 | 2018-07-16  | 14:10:00
       8 |         10000 | 2018-10-19  | 11:41:00
       9 |         12000 | 2018-12-22  | 17:42:00
      10 |         78000 | 2018-10-08  | 10:53:00
      11 |         67000 | 2018-02-16  | 03:26:00
      12 |         45000 | 2017-05-23  | 14:41:00
      13 |         23000 | 2019-03-30  | 08:12:00
      14 |         23000 | 2017-09-10  | 09:00:00
      15 |         56000 | 2017-10-29  | 16:23:00
      16 |         23000 | 2018-12-28  | 04:56:00
      17 |         12000 | 2017-09-04  | 23:33:00
      18 |         23000 | 2018-05-18  | 21:31:00
      19 |         78000 | 2018-05-09  | 00:51:00
      20 |         67000 | 2018-05-17  | 08:22:00
      21 |         98000 | 2017-08-14  | 10:27:00
      22 |         98000 | 2017-07-20  | 04:44:00
      23 |         10000 | 2017-09-17  | 15:30:00
      24 |         78000 | 2018-11-27  | 03:01:00
      25 |         78000 | 2019-01-27  | 04:43:00
      26 |         10000 | 2017-12-11  | 19:54:00
      27 |         23000 | 2018-07-25  | 15:32:00
      28 |         89000 | 2017-12-23  | 22:42:00
      29 |         12000 | 2018-04-20  | 09:17:00
      30 |         56000 | 2018-04-15  | 22:44:00
      31 |         56000 | 2018-03-26  | 11:24:00
```

3.3.6 Delivery

```
[vendingmachine=# \d delivery
                                   Table "public.delivery"
   Column    |          Type          | Collation | Nullable |                   Default
-------------+------------------------+-----------+----------+--------------------------------------------
 routeid     | integer                |           | not null | nextval('delivery_routeid_seq'::regclass)
 locationid  | integer                |           | not null | nextval('delivery_locationid_seq'::regclass)
 orderid     | integer                |           | not null | nextval('delivery_orderid_seq'::regclass)
 odate       | date                   |           | not null |
 timearrived | time without time zone |           | not null |
 timespent   | integer                |           | not null |
```

```
[vendingmachine=# select * from delivery
[vendingmachine-# ;
 routeid | locationid | orderid |   odate    | timearrived | timespent
---------+------------+---------+------------+-------------+-----------
      39 |          6 |       1 | 2018-05-06 | 04:03:00    |         1
      10 |          1 |       2 | 2017-07-07 | 12:35:00    |         5
      25 |          8 |       3 | 2018-10-23 | 06:17:00    |         4
      45 |          2 |       4 | 2017-08-12 | 00:34:00    |         3
      18 |          6 |       5 | 2019-03-07 | 06:14:00    |         2
      28 |          8 |       6 | 2017-07-27 | 21:49:00    |         4
      39 |          9 |       7 | 2018-02-03 | 14:31:00    |         1
      18 |          5 |       8 | 2017-12-06 | 13:20:00    |         3
      41 |          7 |       9 | 2019-02-17 | 10:00:00    |         1
      34 |          2 |      10 | 2018-11-24 | 18:27:00    |         1
      27 |          3 |      11 | 2018-02-19 | 16:34:00    |         1
      25 |          7 |      12 | 2017-07-19 | 17:37:00    |         5
       3 |          5 |      13 | 2017-07-19 | 14:27:00    |         4
      49 |          8 |      14 | 2018-07-07 | 11:26:00    |         1
      38 |          3 |      15 | 2017-08-17 | 20:25:00    |         5
       8 |          3 |      16 | 2017-12-06 | 16:09:00    |         1
      26 |          2 |      17 | 2017-04-29 | 01:46:00    |         4
      40 |          9 |      18 | 2017-10-17 | 19:51:00    |         1
      31 |          3 |      19 | 2017-08-15 | 22:13:00    |         1
      19 |          3 |      20 | 2017-07-18 | 08:20:00    |         2
      38 |          9 |      21 | 2017-07-06 | 06:49:00    |         2
       7 |          5 |      22 | 2018-04-06 | 08:15:00    |         2
      36 |          2 |      23 | 2018-10-15 | 09:13:00    |         4
      23 |          8 |      24 | 2018-05-22 | 22:46:00    |         3
      36 |          1 |      25 | 2018-08-28 | 20:03:00    |         5
      41 |          1 |      26 | 2018-07-27 | 22:26:00    |         5
      41 |          4 |      27 | 2019-02-09 | 09:44:00    |         4
       6 |          6 |      28 | 2018-02-19 | 06:56:00    |         4
       9 |         10 |      29 | 2017-04-30 | 12:11:00    |         3
      44 |          6 |      30 | 2017-06-09 | 18:33:00    |         4
      13 |          6 |      31 | 2018-07-27 | 02:07:00    |         1
       7 |          6 |      32 | 2018-12-10 | 06:44:00    |         1
      37 |          9 |      33 | 2017-05-18 | 02:39:00    |         5
       7 |          5 |      34 | 2017-04-22 | 21:57:00    |         5
      48 |          3 |      35 | 2017-12-31 | 03:21:00    |         4
      10 |         10 |      36 | 2018-08-02 | 03:24:00    |         4
      24 |          2 |      37 | 2017-10-05 | 12:53:00    |         3
      25 |          9 |      38 | 2019-03-05 | 03:18:00    |         2
      16 |          9 |      39 | 2017-12-03 | 09:43:00    |         1
      29 |          3 |      40 | 2018-02-18 | 18:25:00    |         4
      19 |         10 |      41 | 2019-03-12 | 06:08:00    |         1
      23 |          6 |      42 | 2017-07-30 | 17:56:00    |         2
      31 |          9 |      43 | 2017-11-07 | 01:09:00    |         1
      48 |          6 |      44 | 2018-05-07 | 21:44:00    |         5
      42 |          8 |      45 | 2017-08-03 | 08:51:00    |         3
```

3.3.7 Orders

```
vendingmachine=# \d orders
                               Table "public.orders"
   Column   |  Type   | Collation | Nullable |                  Default
------------+---------+-----------+----------+--------------------------------------------
 orderid    | integer |           | not null | nextval('orders_orderid_seq'::regclass)
 supplierid | integer |           | not null | nextval('orders_supplierid_seq'::regclass)
 ordertype  | text    |           | not null |
Indexes:
    "orders_pkey" PRIMARY KEY, btree (orderid)
```

```
vendingmachine=# select * from orders
vendingmachine-# ;
 orderid | supplierid |   ordertype
---------+------------+----------------
       1 |          5 | WarehouseOrder
       2 |         10 | VendingOrder
       3 |          2 | WarehouseOrder
       4 |          4 | WarehouseOrder
       5 |          8 | WarehouseOrder
       6 |          4 | WarehouseOrder
       7 |          3 | WarehouseOrder
       8 |          6 | VendingOrder
       9 |          7 | VendingOrder
      10 |          8 | WarehouseOrder
      11 |          2 | VendingOrder
      12 |          1 | VendingOrder
      13 |          7 | WarehouseOrder
      14 |          3 | VendingOrder
      15 |          7 | VendingOrder
      16 |          6 | VendingOrder
      17 |          5 | VendingOrder
      18 |          2 | WarehouseOrder
      19 |         10 | VendingOrder
      20 |         10 | WarehouseOrder
      21 |          8 | VendingOrder
      22 |          9 | VendingOrder
      23 |          8 | WarehouseOrder
      24 |          6 | VendingOrder
      25 |          3 | WarehouseOrder
      26 |          7 | WarehouseOrder
      27 |          8 | VendingOrder
      28 |          3 | VendingOrder
      29 |          9 | VendingOrder
      30 |          5 | VendingOrder
      31 |          7 | VendingOrder
      32 |          4 | VendingOrder
      33 |          5 | WarehouseOrder
      34 |          7 | VendingOrder
      35 |          6 | VendingOrder
      36 |          5 | VendingOrder
```

3.3.8 OrderContains

```
[vendingmachine=# \d ordercontains
                              Table "public.ordercontains"
    Column     |  Type   | Collation | Nullable |                      Default
---------------+---------+-----------+----------+----------------------------------------------------
 itemtypeid    | integer |           | not null | nextval('ordercontains_itemtypeid_seq'::regclass)
 orderid       | integer |           | not null | nextval('ordercontains_orderid_seq'::regclass)
 numitemtype   | integer |           | not null | nextval('ordercontains_numitemtype_seq'::regclass)
 itemtypeprice | money   |           | not null |
 expdate       | date    |           | not null |
```

```
[vendingmachine=# select * from ordercontains
[vendingmachine-# ;
 itemtypeid | orderid | numitemtype | itemtypeprice |  expdate
------------+---------+-------------+---------------+------------
          8 |       1 |         101 |         $3.90 | 2020-03-20
          4 |       2 |         175 |         $2.75 | 2019-04-05
         10 |       3 |          38 |         $1.85 | 2019-07-14
          7 |       4 |         156 |         $3.66 | 2020-01-07
          3 |       6 |          28 |         $3.74 | 2019-04-08
          2 |       7 |          59 |         $1.09 | 2019-11-05
          7 |       8 |         184 |         $1.15 | 2019-09-15
          4 |       9 |          51 |         $1.19 | 2019-08-03
          5 |      10 |         116 |         $1.15 | 2020-01-13
          2 |      11 |         110 |         $1.48 | 2020-02-23
          1 |      12 |          97 |         $2.08 | 2020-01-07
          5 |      13 |          47 |         $2.27 | 2019-04-22
          2 |      14 |         190 |         $1.06 | 2019-06-03
         10 |      15 |          10 |         $1.60 | 2019-04-20
          8 |      16 |          39 |         $2.99 | 2020-03-22
          9 |      17 |          67 |         $2.78 | 2019-09-22
          8 |      18 |         110 |         $2.17 | 2019-12-18
          9 |      19 |          33 |         $3.83 | 2019-12-04
          1 |      20 |         198 |         $2.97 | 2019-10-12
          1 |      21 |          15 |         $1.18 | 2019-10-16
          3 |      22 |          49 |         $1.33 | 2019-12-18
          6 |      23 |          67 |         $1.33 | 2019-08-31
          4 |      24 |          12 |         $3.06 | 2020-02-14
          1 |      25 |         114 |         $2.63 | 2019-07-06
          4 |      26 |         133 |         $3.31 | 2019-07-23
          5 |      27 |          27 |         $3.62 | 2019-04-20
          2 |      28 |          99 |         $1.31 | 2019-06-02
          4 |      29 |          49 |         $3.27 | 2020-03-29
          4 |      30 |         172 |         $3.86 | 2019-08-22
          8 |      31 |          22 |         $2.30 | 2020-01-08
         10 |      32 |          90 |         $3.76 | 2019-10-12
          3 |      33 |          96 |         $3.76 | 2019-10-24
          1 |      34 |          35 |         $1.02 | 2019-11-13
          9 |      35 |         154 |         $2.33 | 2019-08-01
          1 |      36 |          63 |         $2.55 | 2019-08-28
          9 |      37 |         185 |         $1.41 | 2019-04-14
          2 |      38 |         166 |         $3.96 | 2020-01-20
```

3.3.9 ItemType

Column	Type	Collation	Nullable
itemtypeid	integer		not null
invoiceid	integer		not null
numsold	integer		
pricesold	money		

itemtypeid	supplierid	itemtypename	msrp
1	1	Pepsi	$3.20
2	2	Sprite	$1.35
3	3	Squirt	$2.90
4	4	Coke	$2.56
5	5	Diet Pepsi	$3.29
6	6	DIet Coke	$1.03
7	7	Fanta	$1.04
8	8	Powerade	$1.78
9	9	Minute Maid	$3.94
10	10	Crush	$2.41

(10 rows)

3.3.10 Warehouse

```
                                Table "public.wareh
   Column    |  Type   | Collation | Nullable |
-------------+---------+-----------+----------+-------
 warehouseid | integer |           | not null | nextva
 ssn         | text    |           |          |
 capacity    | integer |           |          |
 streetname  | text    |           |          |
 city        | text    |           |          |
 state       | text    |           |          |
 zip         | integer |           |          |
Indexes:
    "warehouse_pkey" PRIMARY KEY, btree (warehouseid)
```

```
student=> select * from warehouse;
 warehouseid |     ssn     | capacity |  streetname   |    city     | state |  zip
-------------+-------------+----------+---------------+-------------+-------+-------
           1 | 123-45-6789 |    10000 | Doe Crossing  | Bakersfield | CA    | 11111
           2 | 987-65-4321 |    50000 | Dayton        | Bakersfield | CA    | 11112
           3 | 987-65-4321 |    50000 | Hanover       | Bakersfield | CA    | 11113
           4 | 987-65-4321 |    50000 | Oxford        | Bakersfield | CA    | 11114
           5 | 987-65-4321 |    50000 | Glacier Hill  | Bakersfield | CA    | 11115
           6 | 123-45-6789 |    50000 | Redwing       | Bakersfield | CA    | 11116
           7 | 123-45-6789 |    50000 | Onsgard       | Bakersfield | CA    | 11117
           8 | 123-45-6789 |    25000 | Pawling       | Bakersfield | CA    | 11118
           9 | 123-45-6789 |    10000 | Columbus      | Bakersfield | CA    | 11119
          10 | 987-65-4321 |    20000 | Towne         | Bakersfield | CA    | 11120
(10 rows)
```

3.3.11 WarehouseHas

Column	Type	Collation	Nullable
warehouseid	integer		not null
itemtypeid	integer		not null
numitemtype	integer		
itemtypeprice	money		

warehouseid	itemtypeid	numitemtype	itemtypeprice
8	9	198	$1.12
5	2	77	$1.59
10	8	58	$2.23
8	3	132	$1.65
6	5	156	$3.35
2	3	100	$2.72
9	5	78	$3.60
4	5	159	$2.23
10	6	119	$2.66
2	6	98	$2.83
6	10	62	$2.68
1	7	155	$2.84
10	5	8	$2.76
7	4	126	$1.05
2	8	200	$3.29
5	10	95	$2.80
7	2	21	$1.96
5	2	49	$2.83
1	9	25	$1.63
6	9	161	$1.18
1	4	141	$1.00
9	4	71	$3.42
10	10	22	$3.41
5	8	101	$2.32
9	1	111	$3.10
4	3	138	$3.20
3	4	71	$1.49
5	9	91	$2.54
7	3	68	$2.62
10	4	68	$1.89
6	6	41	$1.41
3	10	69	$1.25
6	3	115	$3.90
2	1	73	$3.92
9	3	118	$1.99
4	4	176	$3.42
3	9	62	$2.18
5	5	178	$3.02
8	9	137	$3.36

3.3.12 WarehouseReceives

Column	Type	Collation	Nullable
orderid	integer		not null
warehouseid	integer		not null
timerec	time without time zone		
daterec	date		

orderid	warehouseid	timerec	daterec
2	10	06:51:00	2018-01-10
5	1	08:22:00	2017-06-09
10	10	21:38:00	2017-05-28
1	2	07:19:00	2017-09-27
9	8	13:57:00	2018-02-23
9	10	18:56:00	2018-09-01
7	8	14:52:00	2018-01-17
6	7	12:10:00	2019-02-11
2	7	12:57:00	2019-02-17
6	1	14:17:00	2017-06-01
3	10	16:41:00	2018-02-24
2	2	02:06:00	2018-07-14
3	9	11:09:00	2018-11-24
8	7	16:43:00	2018-05-08
6	5	23:23:00	2018-04-03
6	2	13:56:00	2018-01-14
2	3	00:23:00	2018-06-26
10	5	13:43:00	2017-11-09
7	7	12:12:00	2018-09-14
9	10	15:06:00	2019-03-13
3	8	16:38:00	2018-05-01
6	4	14:25:00	2018-02-23
10	6	04:35:00	2017-08-02
10	6	22:35:00	2017-09-07
10	1	22:50:00	2018-10-02
5	4	19:52:00	2017-12-14
5	9	21:04:00	2017-05-14
10	8	05:07:00	2017-05-26
10	1	01:23:00	2017-07-30
7	1	10:36:00	2018-07-29
6	2	22:33:00	2017-05-08
3	1	13:44:00	2019-02-16
3	7	07:53:00	2018-05-22
4	7	00:14:00	2018-12-19
3	4	09:06:00	2018-09-06
6	5	06:20:00	2018-08-12
1	3	14:00:00	2018-12-07
4	9	23:18:00	2018-12-02
1	4	19:47:00	2019-02-11
9	6	05:41:00	2017-09-20
8	3	09:00:00	2017-10-06
2	3	07:39:00	2019-02-25
8	6	04:07:00	2018-09-06
5	4	16:26:00	2018-03-26
5	7	11:52:00	2018-03-13
3	2	21:09:00	2018-10-30
7	8	22:23:00	2018-09-01
10	5	03:07:00	2017-10-13
1	8	12:46:00	2018-09-10
9	8	03:28:00	2017-08-20

(50 rows)

3.3.13 OrdersFrom

Column	Type	Collation	Nullable
badgenumber	integer		not null
supplierid	integer		not null
dateordered	date		
timeordered	time without time zone		

badgenumber	supplierid	dateordered	timeordered
1	8	2018-10-18	02:45:00
2	8	2018-06-18	05:59:00
8	7	2018-03-14	05:18:00
5	6	2017-09-22	01:34:00
5	9	2017-08-14	17:30:00
6	4	2018-05-17	01:14:00
6	2	2019-02-20	06:57:00
9	10	2018-01-05	05:59:00
6	4	2018-09-19	17:17:00
10	1	2018-07-27	23:17:00
2	7	2017-12-23	03:33:00
4	5	2018-07-09	16:55:00
7	2	2018-10-23	00:26:00
1	1	2018-11-14	11:06:00
8	10	2018-04-04	02:39:00
7	3	2018-01-01	17:58:00
6	2	2018-12-09	05:47:00
3	1	2018-06-17	18:42:00
5	6	2017-11-17	03:57:00
5	3	2018-07-26	07:39:00
5	8	2017-09-21	08:48:00
2	5	2018-11-30	02:18:00
5	6	2018-05-13	03:24:00
8	1	2018-11-06	17:35:00
5	1	2018-07-30	07:36:00
5	3	2018-12-15	14:45:00

3.3.14 PlacesOrder

Table "public.placesor

Column	Type	Collation	Nullable
orderid	integer		not null
badgenumber	integer		not null
timeplaced	time without time zone		
dateplaced	date		

orderid	badgenumber	timeplaced	dateplaced
2	6	05:20:00	2017-11-18
3	8	10:22:00	2017-08-12
3	2	01:03:00	2018-04-02
7	2	14:02:00	2018-02-18
6	10	07:56:00	2018-11-28
6	1	08:29:00	2018-08-07
2	3	23:49:00	2017-10-26
5	8	01:17:00	2017-10-24
6	2	11:38:00	2017-10-24
1	10	17:10:00	2018-10-13
7	8	05:59:00	2018-10-19
6	5	09:52:00	2017-07-28
8	1	03:40:00	2018-02-12
2	8	08:16:00	2017-04-15
9	2	06:09:00	2018-12-09
5	9	07:50:00	2017-12-15
1	8	15:38:00	2018-02-18
2	1	17:25:00	2018-05-20
2	8	03:53:00	2018-04-13
1	6	07:53:00	2019-01-20
2	10	09:08:00	2018-08-11
6	6	08:09:00	2017-07-19
3	2	05:51:00	2018-04-15
9	10	20:17:00	2018-03-23
2	10	04:44:00	2018-05-31
9	10	00:53:00	2017-10-06
4	7	03:49:00	2018-06-14
10	8	02:18:00	2018-11-25
8	8	08:47:00	2017-08-21

3.3.15 Supplier

```
   Column    |  Type   | Collation | Nullable |
-------------+---------+-----------+----------+-----
 supplierid  | integer |           | not null | next
 name        | text    |           |          |
 phone       | text    |           |          |
 streetname  | text    |           |          |
 city        | text    |           |          |
 state       | text    |           |          |
 zip         | integer |           |          |
Indexes:
    "supplier_pkey" PRIMARY KEY, btree (supplierid)
```

```
 supplierid |    name    |    phone     | streetname |      city       | state |  zip
------------+------------+--------------+------------+-----------------+-------+-------
          1 | Lajo       | 213-882-2592 | Mosinee    | Los Angeles     | CA    | 11111
          2 | Cogidoo    | 209-341-5236 | Daystar    | Stockton        | CA    | 11112
          3 | Quaxo      | 818-647-8909 | Nevada     | North Hollywood | CA    | 11113
          4 | Photospace | 559-779-8191 | Fulton     | Visalia         | CA    | 11114
          5 | Youspan    | 818-544-7711 | Walton     | North Hollywood | CA    | 11115
          6 | Oyope      | 310-529-6251 | Parkside   | Long Beach      | CA    | 11116
          7 | Skidoo     | 818-175-5739 | Packers    | Glendale        | CA    | 11117
          8 | Skimia     | 858-574-6504 | Dexter     | San Diego       | CA    | 11118
          9 | Kwinu      | 213-538-2484 | Chive      | Los Angeles     | CA    | 11119
         10 | Centizu    | 559-270-0717 | Maryland   | Visalia         | CA    | 11120
(10 rows)
```

3.3.16 GasReceipt

Column	Type	Collation	Nullable	
receiptid	integer		not null	nextv
licensenumber	integer			
streetname	text			
city	text			
state	text			
zip	text			
totalprice	money			
datecreated	date			

Indexes:
"gasreceipt_pkey" PRIMARY KEY, btree (receiptid)

receiptid	licensenumber	streetname	city	state	zip	totalprice	datecreated
1	78000	Hayes	Bakersfield	CA	11111	$47.26	2017-11-29
2	56000	Waubesa	Bakersfield	CA	11116	$129.21	2018-11-05
3	67000	Gerald	Bakersfield	CA	11121	$33.30	2018-10-27
4	67000	Glendale	Bakersfield	CA	11126	$50.31	2018-11-30
5	34000	Macpherson	Bakersfield	CA	11131	$44.57	2017-10-02
6	34000	Lindbergh	Bakersfield	CA	11136	$90.22	2018-03-23
7	78000	Acker	Bakersfield	CA	11141	$93.01	2017-08-29
8	34000	Mesta	Bakersfield	CA	11146	$54.81	2019-03-06
9	45000	Charing Cross	Bakersfield	CA	11111	$100.16	2017-10-26
10	34000	Dunning	Bakersfield	CA	11116	$129.55	2017-08-24
11	98000	Bashford	Bakersfield	CA	11121	$34.03	2018-04-23
12	12000	Fuller	Bakersfield	CA	11126	$98.30	2017-11-02
13	89000	Mayer	Bakersfield	CA	11131	$43.84	2019-02-13
14	45000	Maple Wood	Bakersfield	CA	11136	$113.26	2019-03-07
15	56000	Moose	Bakersfield	CA	11141	$105.00	2018-02-09
16	23000	Dahle	Bakersfield	CA	11146	$92.30	2017-03-30
17	67000	Crescent Oaks	Bakersfield	CA	11111	$91.59	2019-01-16
18	89000	Grayhawk	Bakersfield	CA	11116	$89.63	2019-01-11
19	12000	Oriole	Bakersfield	CA	11121	$118.32	2018-02-02
20	78000	Nevada	Bakersfield	CA	11126	$40.33	2018-01-04
21	78000	Holmberg	Bakersfield	CA	11131	$98.22	2018-01-26
22	78000	Pierstorff	Bakersfield	CA	11136	$67.42	2017-08-16
23	89000	6th	Bakersfield	CA	11141	$44.52	2019-03-01
24	34000	Portage	Bakersfield	CA	11146	$75.23	2018-10-17
25	78000	Spohn	Bakersfield	CA	11111	$26.41	2018-04-29
26	23000	Buell	Bakersfield	CA	11116	$120.01	2017-12-13
27	12000	Annamark	Bakersfield	CA	11121	$98.83	2019-02-03
28	56000	Jenna	Bakersfield	CA	11126	$65.04	2017-08-08
29	78000	Goodland	Bakersfield	CA	11131	$75.03	2017-05-26
30	34000	Oak	Bakersfield	CA	11136	$29.40	2018-10-29
31	12000	Dovetail	Bakersfield	CA	11141	$99.86	2019-02-24
32	89000	Amoth	Bakersfield	CA	11146	$96.12	2018-06-15
33	45000	Warbler	Bakersfield	CA	11111	$135.90	2018-12-13
34	89000	Quincy	Bakersfield	CA	11116	$24.61	2017-09-11
35	10000	Blackbird	Bakersfield	CA	11121	$24.80	2018-04-25
36	12000	Merry	Bakersfield	CA	11126	$77.21	2017-04-28
37	78000	Parkside	Bakersfield	CA	11131	$108.05	2018-02-01
38	34000	Lake View	Bakersfield	CA	11136	$80.20	2018-01-28
39	12000	Hazelcrest	Bakersfield	CA	11141	$96.73	2018-01-18
40	12000	Northland	Bakersfield	CA	11146	$60.86	2017-12-02
41	67000	Tomscot	Bakersfield	CA	11111	$104.01	2017-10-17
42	56000	Ramsey	Bakersfield	CA	11116	$143.97	2017-11-07
43	89000	Shasta	Bakersfield	CA	11121	$78.02	2019-01-16
44	34000	Ilene	Bakersfield	CA	11126	$41.45	2017-09-06
45	56000	Schmedeman	Bakersfield	CA	11131	$77.10	2018-07-10
46	67000	Crescent Oaks	Bakersfield	CA	11136	$110.79	2017-07-23
47	98000	Goodland	Bakersfield	CA	11141	$56.89	2017-08-19
48	67000	Onsgard	Bakersfield	CA	11146	$137.30	2018-11-08
49	45000	Luster	Bakersfield	CA	11111	$114.06	2017-08-27
50	12000	Lien	Bakersfield	CA	11116	$106.88	2017-06-11

3.3.17 Invoice

```
   Column    |          Type          | Collation | Nullable |
-------------+------------------------+-----------+----------+-
 invoiceid   | integer                |           | not null |
 machineid   | integer                |           | not null |
 datecreated | date                   |           |          |
 timecreated | time without time zone |           |          |
Indexes:
    "invoice_pkey" PRIMARY KEY, btree (invoiceid)
```

```
 invoiceid | machineid | datecreated | timecreated
-----------+-----------+-------------+-------------
         1 |         9 | 2018-12-31  | 20:03:00
         2 |         2 | 2018-01-12  | 05:11:00
         3 |         6 | 2018-07-21  | 15:41:00
         4 |         8 | 2017-09-19  | 18:57:00
         5 |         9 | 2018-05-15  | 01:00:00
         6 |         6 | 2019-03-05  | 23:22:00
         7 |         5 | 2017-09-19  | 17:52:00
         8 |         4 | 2018-02-07  | 23:38:00
         9 |         1 | 2017-10-17  | 12:03:00
        10 |         5 | 2018-11-09  | 08:32:00
        11 |         6 | 2017-08-26  | 08:47:00
        12 |         8 | 2018-01-21  | 11:08:00
        13 |        10 | 2018-03-13  | 20:17:00
        14 |         7 | 2018-05-18  | 05:24:00
        15 |         1 | 2017-05-23  | 22:46:00
        16 |         9 | 2017-11-29  | 18:30:00
        17 |         7 | 2018-12-12  | 01:37:00
        18 |         1 | 2018-02-05  | 14:19:00
        19 |         9 | 2017-12-25  | 07:03:00
        20 |         3 | 2018-05-13  | 22:41:00
        21 |         5 | 2018-01-28  | 19:50:00
        22 |         4 | 2018-08-29  | 07:06:00
        23 |         5 | 2017-04-19  | 20:15:00
        24 |         2 | 2017-12-20  | 05:01:00
        25 |         1 | 2019-01-26  | 10:08:00
        26 |        10 | 2017-06-12  | 02:55:00
        27 |         2 | 2018-12-03  | 08:44:00
        28 |         3 | 2018-02-12  | 12:59:00
        29 |        10 | 2017-07-14  | 20:30:00
        30 |         9 | 2018-04-27  | 22:34:00
        31 |         8 | 2017-10-12  | 09:18:00
        32 |         8 | 2018-06-07  | 08:48:00
        33 |         3 | 2019-03-19  | 11:25:00
        34 |         7 | 2017-05-24  | 13:33:00
```

3.3.18 ItemsSold

Column	Type	Collation	Nullable	
itemtypeid	integer		not null	r
invoiceid	integer		not null	r
numsold	integer			
pricesold	money			

itemtypeid	invoiceid	numsold	pricesold
2	8	5	$4.84
6	6	11	$0.62
10	6	5	$3.22
3	6	19	$2.00
7	10	9	$3.82
1	10	20	$4.19
9	2	13	$5.93
8	2	1	$3.56
6	4	14	$4.08
10	5	14	$4.60
7	2	18	$0.38
4	6	8	$4.92
9	2	15	$1.37
4	6	18	$0.20
4	8	5	$3.59
9	10	16	$4.71
3	1	3	$0.95
8	3	10	$1.54
2	5	8	$3.31
6	10	6	$5.49
8	7	18	$3.81
5	6	11	$4.05
4	2	6	$1.79
8	3	8	$3.27
2	7	16	$0.24
8	5	19	$3.95
4	8	19	$4.31
9	8	4	$5.51
1	10	6	$5.41
6	1	3	$1.29
3	3	2	$5.66
5	7	4	$5.54
8	3	8	$4.54
4	4	10	$0.39
8	10	1	$3.66
3	8	16	$2.30
7	4	10	$0.87
8	2	17	$1.44
3	9	17	$3.79
2	9	17	$4.59
7	7	16	$3.76
1	8	9	$1.17
8	9	11	$3.21
5	8	18	$5.79
2	2	8	$1.98
6	4	16	$4.70
8	10	10	$5.34
6	5	18	$5.13
1	1	8	$5.16
10	1	5	$0.61

3.3.19 Vehicle

```
                               Table "public.vehic
    Column      |  Type   | Collation | Nullable |
----------------+---------+-----------+----------+----
 vehicleid      | integer |           | not null | next
 platenumber    | text    |           |          |
 licensenumber  | integer |           |          |
 make           | text    |           |          |
 model          | text    |           |          |
 vin            | text    |           |          |
Indexes:
    "vehicle_pkey" PRIMARY KEY, btree (vehicleid)
```

```
student=> select * from vehicle;
 vehicleid | platenumber | licensenumber |   make   |     model      |        vin
-----------+-------------+---------------+----------+----------------+-------------------
         1 | 7BWV747     |         12000 | Mazda    | 929            | 1ZVBP8JS8B5624408
         2 | 6WJA118     |         23000 | GMC      | Savana 1500    | 3VWKP7AJXDM563299
         3 | 7AKC746     |         34000 | Nissan   | Pathfinder     | 3D73Y4EL7AG309256
         4 | 6WJA118     |         45000 | Chrysler | Voyager        | WBANB33515C895056
         5 | 7AKC746     |         56000 | Dodge    | Viper          | JN1CV6AP0BM984150
         6 | 7BRW365     |         67000 | Jeep     | Wrangler       | 3FAHP0DC2BR629741
         7 | 7BWV747     |         78000 | Hyundai  | Elantra        | YV4902BZ4B1936423
         8 | 6WJA118     |         89000 | Volvo    | S60            | SCFFDAAE3CG903450
         9 | 7AFY475     |         98000 | Dodge    | Ram Van 3500   | 1D4PT7GX0BW942084
        10 | 7AYF890     |         10000 | GMC      | Suburban 1500  | WUAGL78E86A467199
(10 rows)
```

3.4 SQL Queries

The following queries are the translations from relational algebra and relational calculus of those in phase 2.

1. **List the SSN and Name of all Employees who are drivers.**
   SELECT ssn, fname, lname FROM employee
   WHERE position = 'Driver';

   Output:

```
     ssn      |  fname   |   lname
-------------+---------+----------
 384-48-7106 | Pascal  | Radbourne
 241-07-2426 | Alexis  | Treece
 114-33-9719 | Jobey   | Shemmans
 223-32-5721 | Huntlee | Huntress
 221-52-7822 | Giulia  | McSorley
(5 rows)
```

2. **List SSN, Name, and Start Date of all Driver's who have started in the past year.**
   SELECT ssn, fname, lname, sdate FROM employee
   WHERE position = 'Driver'
   AND sdate >= now() - '1 year'::interval;

   Output:

```
vendingmachine=# select * from employee where position = 'Driver'
vendingmachine-# and sdate >= now() - '1 year'::interval;
 employeeid | fname  |  lname   |    phone     | position |   salary   |   sdate    |       streetname       |    city    | state |  zip  | lnum  | bnum |       email        |     ssn
------------+--------+----------+--------------+----------+------------+------------+------------------------+------------+-------+-------+-------+------+--------------------+-------------
          2 | Giulia | McSorley | 916-134-2215 | Driver   | $77,363.26 | 2019-01-01 | 532 Washington Parkway | Sacramento | CA    | 94297 | 23000 |    2 | gmcsorley1@go.com | 221-52-7822
(1 row)
```

3. **List all Clients who own at least two machines.**
   SELECT c1.clientid,c1.fname,c1.lname FROM client c1, vendingmachine m1,vendingmachine m2
   WHERE c1.clientid = m1.clientid AND c1.clientid = m2.clientid AND m1.machineid <> m2.machineid
   GROUP BY c1.clientid;

   Output:

```
vendingmachine=# select c1.fname from client c1, vendingmachine m1, vendingmachine m2 where    c1.clientid = m1.clientid and
vendingmachine-# c1.clientid = m2.clientid and m1.machineid <> m2.machineid group by c1.clientid;
 fname
-------
 Moira
 Kirby
(2 rows)
```

4. **List Clients who own only one machine.**

SELECT c2.* from client c2 EXCEPT SELECT c1.* FROM client c1, vendingmachine m1, vendingmachine m2
WHERE c1.clientid = m1.clientid AND c1.clientid = m2.clientid AND m1.machineid <> m2.machineid
GROUP BY c1.clientid;

Output:

```
vendingmachine=# select c2.* from client c2 except select c1.* from client c1, vendingmachine m1, vendingmachine m2 where c1.clientid = m1.clientid and
vendingmachine-# c1.clientid = m2.clientid and m1.machineid <> m2.machineid group by c1.clientid;
 clientid |   fname   |  lname   |           email           |    phone     |        streetname        |     city      | state |  zip  | compname
----------+-----------+----------+---------------------------+--------------+--------------------------+---------------+-------+-------+----------
        4 | Gabriello | Cantrell | gcantrell3@xrea.com       | 415-395-6080 | 9 Havey Street           | San Francisco | CA    | 94137 | Quatz
        5 | Tomi      | Sirrell  | tsirrell4@examiner.com    | 707-160-8061 | 38289 Loftsgordon Place  | Petaluma      | CA    | 94975 | Mymm
        9 | Micky     | Cantrell | mcantrell8@squidoo.com    | 916-395-6954 | 19 Moose Lane            | Sacramento    | CA    | 95852 | Dynabox
        6 | Maisey    | Hadenton | mhadenton5@friendfeed.com | 714-515-3857 | 31 Arrowood Trail        | Anaheim       | CA    | 92825 | Kare
        7 | Aimee     | Pittel   | apittel6@furl.net         | 559-307-4377 | 29749 New Castle Parkway | Fullerton     | CA    | 92640 | Centimia
        3 | Clarance  | Grubbe   | cgrubbe2@yelp.com         | 310-654-8928 | 4 American Ash Circle    | Los Angeles   | CA    | 90071 | Vitz
        2 | Freddi    | Floris   | ffloris1@nytimes.com      | 951-240-5919 | 9948 Sundown Drive       | Riverside     | CA    | 92519 | Gigabox
        1 | Garald    | De Cruz  | gdecruz0@bloomberg.com    | 805-346-7783 | 65 Elka Circle           | San Mateo     | CA    | 94405 | Mydo
(8 rows)
```

5. **List all Machines that stock at least two ItemTypes.**

SELECT m1.* from vendingmachine m1, Invoice i, ItemsSold s1, ItemsSold s2
WHERE m1.machineid = i.machineid AND i.invoiceid = s1.invoiceid AND
s2.invoiceid = i.invoiceid AND s1.itemtypeid <> s2.itemtypeid
GROUP BY m1.machineid;

Output:

```
vendingmachine=# select m1.* from vendingmachine m1,Invoice i, ItemsSold s1, ItemsSold s2
where m1.machineid = i.machineid and i.invoiceid = s1.invoiceid and
s2.invoiceid = i.invoiceid and s1.itemtypeid <> s2.itemtypeid group by m1.machineid;
 machineid | clientid | locationid | build | itemsperslot | capacity
-----------+----------+------------+-------+--------------+----------
         1 |        1 |         10 |       |           10 |      610
         2 |        7 |         10 |       |           15 |      467
         4 |        8 |          8 |       |           20 |      631
         5 |        5 |          8 |       |           15 |      786
         6 |        8 |          1 |       |           10 |      639
         7 |        2 |          5 |       |           15 |      678
         8 |       10 |          7 |       |           20 |      205
         9 |        8 |          9 |       |           15 |      544
        10 |       10 |          8 |       |           15 |      540
(9 rows)
```

6. **List all Drivers who have delivered to every location.**

```
SELECT e.EmployeeID, e.fname, e.lname, e.LNum FROM employee e
NATURAL JOIN (SELECT r.LicenseNumber FROM route r
NATURAL JOIN (SELECT d.RouteID FROM Delivery d
        WHERE NOT EXISTS (SELECT * FROM Location l
        WHERE NOT EXISTS (SELECT * FROM Delivery d1
        WHERE d1.RouteID = d.RouteID AND d1.LocationID = l.locationid))) AS p)
        AS ep WHERE e.lnum = ep.LicenseNumber
        GROUP BY e.employeeid;
```

Output:

```
 employeeid | fname  |    lname     | lnum
------------+--------+--------------+-------
          1 | Nadiya | Sleightholme | 12000
(1 row)
```

7. **List all Dispatchers who have ordered from all suppliers.**

```
SELECT e.employeeid, e.fname, e.lname, e.bnum FROM employee e
NATURAL JOIN  (SELECT o.BadgeNumber FROM OrdersFrom o
        WHERE NOT EXISTS (SELECT * FROM Supplier s
WHERE NOT EXISTS (SELECT * FROM OrdersFrom o1
    WHERE o1.BadgeNumber = o.BadgeNumber AND o1.SupplierID = s.SupplierID)))
AS p WHERE e.bnum = p.badgenumber GROUP BY e.employeeid;
```

Output:

```
 employeeid | fname  |    lname     | bnum
------------+--------+--------------+------
          1 | Nadiya | Sleightholme |    1
(1 row)
```

8. **List all Driver who delivered to CSUB between Jan 2018 and Jan 2019.**

SELECT e.employeeid, e.fname, e.lname, e.ssn, d1.odate, d1.locationid, l.description FROM employee e
NATURAL JOIN route
NATURAL JOIN delivery d1,
(SELECT * FROM location
WHERE description = 'csub') AS l WHERE d1.locationid = l.locationid
AND l.description = 'csub' AND odate >= '2018-01-01'::date AND odate <= 2019-01-01'::date
GROUP BY e.employeeid, d1.odate, d1.locationid, l.description ORDER BY e.employeeid;

Output:

employeeid	fname	lname	ssn	odate	locationid	description
1	Nadiya	Sleightholme	444-87-3220	2018-02-23	1	csub
1	Nadiya	Sleightholme	444-87-3220	2018-07-08	1	csub
1	Nadiya	Sleightholme	444-87-3220	2018-07-27	1	csub
1	Nadiya	Sleightholme	444-87-3220	2018-08-03	1	csub
1	Nadiya	Sleightholme	444-87-3220	2018-08-28	1	csub
1	Nadiya	Sleightholme	444-87-3220	2018-09-05	1	csub
1	Nadiya	Sleightholme	444-87-3220	2018-09-15	1	csub
1	Nadiya	Sleightholme	444-87-3220	2018-09-20	1	csub
1	Nadiya	Sleightholme	444-87-3220	2018-10-28	1	csub
1	Nadiya	Sleightholme	444-87-3220	2018-12-04	1	csub
2	Giulia	McSorley	221-52-7822	2018-02-23	1	csub
2	Giulia	McSorley	221-52-7822	2018-07-08	1	csub
2	Giulia	McSorley	221-52-7822	2018-07-27	1	csub
2	Giulia	McSorley	221-52-7822	2018-08-03	1	csub
2	Giulia	McSorley	221-52-7822	2018-08-28	1	csub
2	Giulia	McSorley	221-52-7822	2018-09-05	1	csub
2	Giulia	McSorley	221-52-7822	2018-09-15	1	csub
2	Giulia	McSorley	221-52-7822	2018-09-20	1	csub
2	Giulia	McSorley	221-52-7822	2018-10-28	1	csub
2	Giulia	McSorley	221-52-7822	2018-12-04	1	csub
3	Pascal	Radbourne	384-48-7106	2018-02-23	1	csub
3	Pascal	Radbourne	384-48-7106	2018-07-08	1	csub
3	Pascal	Radbourne	384-48-7106	2018-07-27	1	csub
3	Pascal	Radbourne	384-48-7106	2018-08-03	1	csub
3	Pascal	Radbourne	384-48-7106	2018-08-28	1	csub
3	Pascal	Radbourne	384-48-7106	2018-09-05	1	csub
3	Pascal	Radbourne	384-48-7106	2018-09-15	1	csub
3	Pascal	Radbourne	384-48-7106	2018-09-20	1	csub
3	Pascal	Radbourne	384-48-7106	2018-10-28	1	csub
3	Pascal	Radbourne	384-48-7106	2018-12-04	1	csub
4	Brittani	Tuxsell	588-41-0157	2018-02-23	1	csub
4	Brittani	Tuxsell	588-41-0157	2018-07-08	1	csub
4	Brittani	Tuxsell	588-41-0157	2018-07-27	1	csub
4	Brittani	Tuxsell	588-41-0157	2018-08-03	1	csub
4	Brittani	Tuxsell	588-41-0157	2018-08-28	1	csub
4	Brittani	Tuxsell	588-41-0157	2018-09-05	1	csub
4	Brittani	Tuxsell	588-41-0157	2018-09-15	1	csub
4	Brittani	Tuxsell	588-41-0157	2018-09-20	1	csub
4	Brittani	Tuxsell	588-41-0157	2018-10-28	1	csub
4	Brittani	Tuxsell	588-41-0157	2018-12-04	1	csub
5	Alexis	Treece	241-07-2426	2018-02-23	1	csub
5	Alexis	Treece	241-07-2426	2018-07-08	1	csub
5	Alexis	Treece	241-07-2426	2018-07-27	1	csub
5	Alexis	Treece	241-07-2426	2018-08-03	1	csub
5	Alexis	Treece	241-07-2426	2018-08-28	1	csub
5	Alexis	Treece	241-07-2426	2018-09-05	1	csub
5	Alexis	Treece	241-07-2426	2018-09-15	1	csub
5	Alexis	Treece	241-07-2426	2018-09-20	1	csub
5	Alexis	Treece	241-07-2426	2018-10-28	1	csub
5	Alexis	Treece	241-07-2426	2018-12-04	1	csub
6	Jobey	Shemmans	114-33-9719	2018-02-23	1	csub
6	Jobey	Shemmans	114-33-9719	2018-07-08	1	csub
6	Jobey	Shemmans	114-33-9719	2018-07-27	1	csub
6	Jobey	Shemmans	114-33-9719	2018-08-03	1	csub
6	Jobey	Shemmans	114-33-9719	2018-08-28	1	csub
6	Jobey	Shemmans	114-33-9719	2018-09-05	1	csub
6	Jobey	Shemmans	114-33-9719	2018-09-15	1	csub
6	Jobey	Shemmans	114-33-9719	2018-09-20	1	csub
6	Jobey	Shemmans	114-33-9719	2018-10-28	1	csub

9. **Invoice that contains the least amount of items sold.**

```
SELECT * FROM invoice
NATURAL JOIN (SELECT s.* FROM itemssold s EXCEPT
    SELECT s2.* FROM itemssold s2, itemssold s1
    WHERE s2.numsold > s1.numsold)
    AS sla ORDER BY sla.numsold;
```

Output:

invoiceid	machineid	datecreated	timecreated	itemtypeid	numsold	pricesold
2	2	2018-01-12	05:11:00	8	1	$3.56
10	5	2018-11-09	08:32:00	8	1	$3.66

(2 rows)

10. **Invoice that contains the second least amount of items sold.**

```
SELECT * FROM invoice
NATURAL JOIN (SELECTION s.* FROM itemssold s EXCEPT
    SELECT s3.* FROM itemssold s3, itemssold s2, itemssold s1
    WHERE s3.numsold > s2.numsold
    AND s2.numsold > s1.numsold) AS sla
    ORDER BY sla.numsold;
```

Output:

invoiceid	machineid	datecreated	timecreated	itemtypeid	numsold	pricesold
2	2	2018-01-12	05:11:00	8	1	$3.56
10	5	2018-11-09	08:32:00	8	1	$3.66
35	1	2018-02-05	11:25:00	2	2	$4.04
3	6	2018-07-21	15:41:00	3	2	$5.66

(4 rows)

4 Stored Subprograms, Packages and Triggers

In this phase, we will discuss the implementation of PL/pgSQL and how it is used to implement complex database operations. We will cover the purpose of PL/pgSQL, some features provided by PostgreSQL and their syntax, and operations used for our PostgreSQL database.

4.1 Postgres PL/pgSQL

In this section we will take an in depth look into PL/pgSQL and the components implemented to make the database easier to fill with data. We will cover benefits, program structure, control statements, stored procedures and functions and their syntax, and more.

4.1.1 What is PL/pgSQL

PL/pgSQL is a Procedural Language for the PostgreSQL database management system. PostgreSQL was designed to create functions and trigger procedures, add control structures to the SQL language, and perform complex computations. It can inherit all user-defined types, functions, and operators, be defined to be trusted by the server. Overall, it is a simple to learn but powerful system that is great to master.

The Benefits of PL/pgSQL are the ability to implement stored procedures, functions and triggers. Stored procedures and functions can be implemented in place of writing long queries that can have errors, take long amounts of time to execute, and could potentially mess up the information already in the database. Queries must be compiled every time they are executed while stored procedures are compiled only the first time and stored in cache memory. These advantages make stored procedures very powerful as they can greatly speed up data gathering when millions of records are involved. Triggers can be used to activate another function, e.g. INSERT or DELETE, when a certain event occurs. Triggers can be used modify data so that constraints may not be violated, can be used to catch data changes and move them to new relations, help with cascading deletes, and many other convenient procedures.

4.1.2 PL/pgSQL Program Structure, Control Statements, and Cursors

Program Structure

PL/pgSQL is a block structured language which results in functions or stored procedures being organized in blocks. The blocks consist of a declaration and a body. The declaration section is where all the variables that will be used within the body section are declared and the body section is where the code manipulates data.

Syntax:

[<<**label**>>]
[DECLARE
declarations]
BEGIN
statements;
...
END [***label***];

Control Statements

Control statements can be used to manipulate data in a PostgreSQL database system. The different types of structures are: return, conditional, loop, and error trapping.

The **return** statements allow you to return data from a function and its syntax is:

RETURN ***expression***;

and

RETURN NEXT ***expression***;

RETURN QUERY ***query***;

RETURN QUERY EXECUTE ***command-string*** [USING ***expression*** [, ...]];

The **conditional** statements are IF and CASE statements that allow you to execute alternative commands based on certain conditions.

IF types:

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

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

IF ***boolean-expression*** THEN ***statements***

[ELSIF ***boolean-expression*** THEN ***statements***

[ELSIF ***boolean-expression*** THEN ***statements***

...]]

[ELSE ***statements***]

END IF;

CASE types:

CASE ***search-expression***

WHEN ***expression*** [, ***expression*** [...]] THEN ***statements***

[WHEN ***expression*** [, ***expression*** [...]] THEN ***statements***

...]

[ELSE ***statements***]

END CASE;

CASE

WHEN ***boolean-expression*** THEN ***statements***

[WHEN ***boolean-expression*** THEN ***statements***

...]

[ELSE ***statements***]

END CASE;

The **loop** statements are LOOP, EXIT, CONTINUE, WHILE, and FOR; they give you the ability to repeat a series of commands.

LOOP:

[<<***label***>>]

LOOP

statements

END LOOP [***label***];

EXIT:

EXIT [***label***] [WHEN ***boolean-expression***];

CONTINUE:

CONTINUE [***label***] [WHEN ***boolean-expression***];

WHILE:

[<<***label***>>]

WHILE ***boolean-expression*** LOOP

statements

END LOOP [***label***];

FOR:

[<<***label***>>]

FOR ***name*** IN [REVERSE] ***expression*** .. ***expression*** [BY ***expression***] LOOP

statements

END LOOP [***label***];

Cursors

A cursor can be setup to encapsulate a query and read the results a few rows at a time to avoid memory overrun if the result contains a large number of rows. You can also return a reference to a cursor that a function has created, which allows the caller to read the rows. It's a special data type named refcursor.

Syntax:

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

4.1.3 Stored Procedure and Syntax

A stored procedure is a set of SQL and procedural statements stored in a database server and can be activated using the SQL interface. These procedures can take parameters and return them as OUT parameters and return single and multiple result sets. Stored procedures are compiled the first time they are executed and then stored in cache memory, avoiding recompiling until removed from cache.

Syntax:

```
CREATE OR REPLACE PROCEDURE procedure_name(parameter_list)
    RETURNS void AS $$
    BEGIN
        stored_procedure_body;
    END;
$$ LANGUAGE language_name;
```

4.1.4 Stored Function and Syntax

A stored function is a user defined function of SQL and procedural statements that are stored in the database server and can be activated using the SQL interface much like a stored procedure. It can be used in an expression and return a value or single result set. They behave just like stored procedures.

Syntax:

```
CREATE OR REPLACE FUNCTION function_name(parameter_list)
    RETURNS void AS $$
    BEGIN
        stored_function_body;
    END;
$$ LANGUAGE language_name;
```

4.1.5 Trigger and Syntax

A trigger is an object associated with a specific table that executes when a certain event occurs. It can be specified to fire before or after the operation is attempted or completed. INSTEAD OF triggers must be marked FOR EACH ROW and can only be defined on views.

FOR EACH ROW: For every row of a relation that a trigger is called upon, fire the trigger

FOR EACH STATEMENT: If a relation fired a trigger, only fire that trigger once per trigger call, instead of once per row.

<u>Syntax:</u>

CREATE [CONSTRAINT] TRIGGER ***name*** { BEFORE | AFTER | INSTEAD OF } { ***event*** [OR ...] }

ON ***table***

[FROM ***referenced_table_name***]

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

[FOR [EACH] { ROW | STATEMENT }]

[WHEN (***condition***)]

EXECUTE PROCEDURE ***function_name*** (***arguments***)

where ***event*** can be one of:

INSERT

UPDATE [OF ***column_name*** [, ...]]

DELETE

TRUNCATE

4.2 Postgres PL/pgSQL Subprograms

In this section we will explore the syntax of creating stored procedures, functions and triggers. Stored procedures, functions and triggers are significant to database systems because they make certain functions easier to write and manage. For instance, if something needs to be added into a table, a stored procedure or function can be used as a shorter version of the query needed to insert the new data.

4.2.1 Stored Procedures

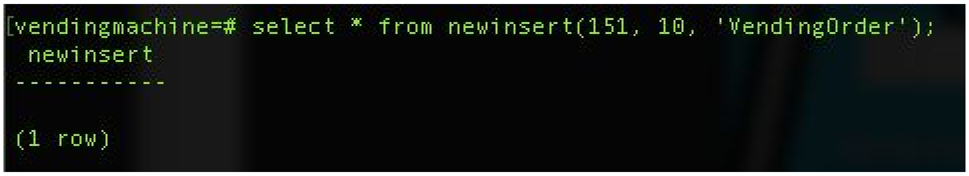
The following are stored procedures and functions that insert, delete and calculate the average of the number items sold.

<u>Insert:</u>

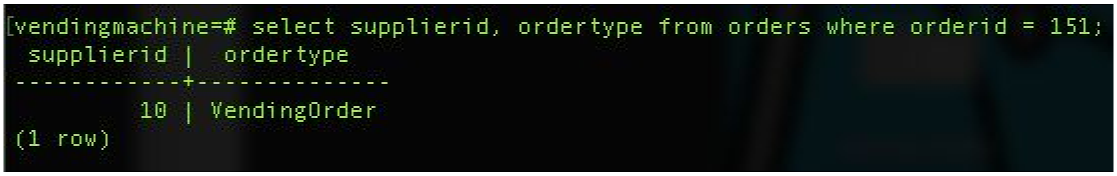
This function inserts new data into the Orders table.

```
CREATE FUNCTION NewInsert(_OrderID integer, _SupplierID integer, _OrderType text)
  RETURNS void AS
  $BODY$
    BEGIN
      INSERT INTO Orders(OrderID, SupplierID, OrderType)
      VALUES(_OrderID, _SupplierID, _OrderType);
    END;
  $BODY$
LANGUAGE 'plpgsql' VOLATILE
```

This screenshot shows the insertion of a new order.



This screenshot shows that the new order was in fact placed in the Orders table.



Delete:

This procedure deletes an employee from the table.

CREATE PROCEDURE DeleteSelect(_DeleteID integer) AS
$BODY$
BEGIN
DELETE FROM Employee
WHERE EmployeeID = _DeleteID;
END;
$BODY$
LANGUAGE 'plpgsql'

To execute a procedure, we must use CALL. This screenshot shows the demonstration of the procedure at use.

```
vendingmachine=# call deleteselect(2);
CALL
```

In this screenshot, you can see that the employee where employeeid = 2 has been deleted.

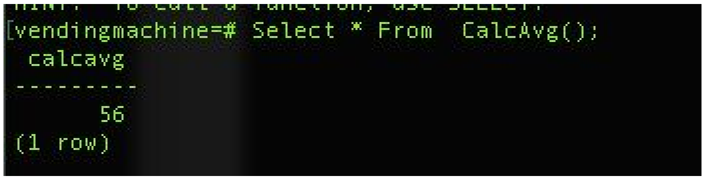
```
vendingmachine=# select employeeid, fname, lname, position from employee
vendingmachine-# ;
 employeeid |  fname   |    lname     |  position
------------+----------+--------------+------------
          1 | Nadiya   | Sleighthol me| Dispatcher
          3 | Pascal   | Radbourne    | Driver
          4 | Brittani | Tuxsell      | Dispatcher
          5 | Alexis   | Treece       | Driver
          6 | Jobey    | Shemmans     | Driver
          7 | Maurits  | Chrismas     | Dispatcher
          8 | Huntlee  | Huntress     | Driver
          9 | Etienne  | Waddington   | Manager
         10 | Cheston  | Thompkins    | Dispatcher
(9 rows)
```

Calculate Average:

This procedure calculates the average number of items sold.

CREATE OR REPLACE FUNCTION CalcAvg()
RETURNS integer AS
$$
BEGIN
RETURN (SELECT AVG(NumSold) FROM ItemsSold);
END;
$$
LANGUAGE plpgsql;

In this screenshot you can see the avg of the number of items sold.



4.2.2 Triggers

This section will explore writing triggers as well as cascading deletions. For simplicity and for the purposes of demonstration, three tables are created: employee, department, and job. The employee and job tables have a DID attribute that reference the department table's primary key, DID.

DID is Department ID

The following syntax creates three tables, two which refer to department. Department must be created first in order to satisfy the foreign key referential constraints set on employee and job. On delete cascade is used to delete a row in a parent table and all rows in other child tables that refer to the parent row.

```
CREATE TABLE department (
        DID serial PRIMARY KEY,
        dname text
);


CREATE TABLE employee(
        eid serial PRIMARY KEY,
        did integer REFERENCES department(did) on DELETE CASCADE,
        ename text
);


CREATE TABLE job(
        jid serial PRIMARY KEY,
        did integer REFERENCES department(did) on DELETE CASCADE,
        jname text
);
```

1st Trigger: Before Update

The BEFORE UPDATE trigger is used to call a procedure before the update statement begins execution. This is beneficial when attempting to work around constraints. For this example we update a department id but before doing so, we set the employee DID to null to avoid any constraint issues.

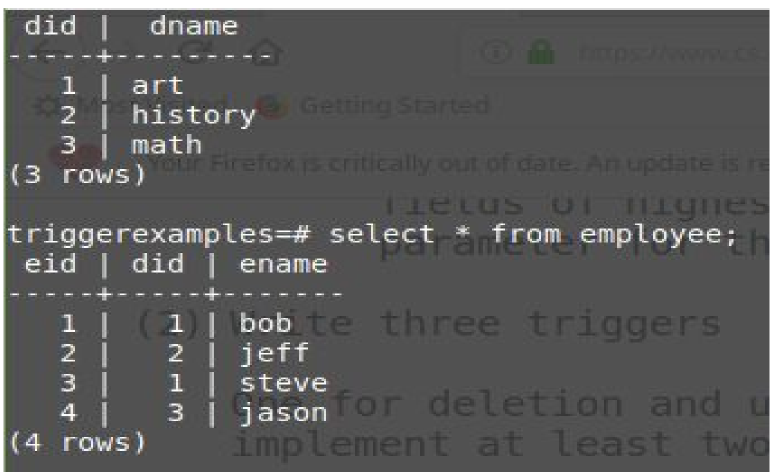
Update department function:

```
CREATE OR REPLACE FUNCTION update_department() RETURNS TRIGGER AS
	$$
	BEGIN
		UPDATE employee set did = null where did = old.did;
		RETURN new;
	END;
	$$
LANGUAGE plpgsql;
```

Trigger:

```
CREATE TRIGGER update_dep
	BEFORE UPDATE ON department
	FOR EACH ROW
	EXECUTE PROCEDURE update_department ();
```

Screenshot: Department and Employee. Department DID = 3 is updated to DID = 4 and employee jason who worked that department has his value to set null.



```
 did |  dname
-----+---------
   1 | art
   2 | history
   3 | math
(3 rows)

triggerexamples=# select * from employee;
 eid | did | ename
-----+-----+-------
   1 |   1 | bob
   2 |   2 | jeff
   3 |   1 | steve
   4 |   3 | jason
(4 rows)
```

```
triggerexamples=# update department set did = 4 where did = 3;
UPDATE 1
triggerexamples=# select * from department;
 did |  dname
-----+---------
   1 | art
   2 | history
   4 | math
(3 rows)

triggerexamples=# select * from employee;
 eid | did | ename
-----+-----+-------
   1 |   1 | bob
   2 |   2 | jeff
   3 |   1 | steve
   4 |     | jason
(4 rows)
```

Example 2: Department DID = 1 is set to 3. Employee Bob and Steve, who worked in that department, have their DID set to null.

```
triggerexamples=# update department set did = 3 where did = 1;
UPDATE 1
triggerexamples=# select * from employee;
 eid | did | ename
-----+-----+-------
   2 |   2 | jeff
   4 |     | jason
   1 |     | bob
   3 |     | steve
(4 rows)

triggerexamples=# select * from department;
 did |  dname
-----+---------
   2 | history
   4 | math
   3 | art
(3 rows)
```

<u>Cascade Delete:</u>

CASCADE DELETE is used to delete a row and all other rows in different tables that reference the parent row. This is used to bypass all constraints and wipe data quickly. This is more of a nuke on the data rather than a surgical deletion.

Below a view is shown crossing employee,department, and job:

```
triggerexamples=# select * from employee natural join department natural join job;
 did | eid | ename | dname | jid |  jname
-----+-----+-------+-------+-----+----------
   1 |   1 | steve | psych |   1 | teacher
   1 |   2 | lucy  | psych |   1 | teacher
   4 |   3 | bill  | math  |   2 | assitant
(3 rows)
```

Two employees, Steve and Lucy work in department 1 as teachers. A delete is used on department with did = 1 and the cascade deletes both lucy and steve. The following screenshot shows the leftover rows in the view:

```
triggerexamples=# delete from department where did = 1;
DELETE 1
triggerexamples=# select * from employee natural join department natural join job;
 did | eid | ename | dname | jid |  jname
-----+-----+-------+-------+-----+----------
   4 |   3 | bill  | math  |   2 | assitant
(1 row)
```

<u>3rd Trigger: INSTEAD OF</u>

Views are a way to combine many relations into one view point. For example, creating a view consisting of relations employee, department, and job would avoid the need to consistently query with joins across multiple tables.

<u>Creating a view:</u>
CREATE VIEW employee_department as SELECT * FROM employee natural join department;

```
triggerexamples=# create view employee_department as select * from employee natural join department;
CREATE VIEW
triggerexamples=# select * from employee_department ;
 did | eid | ename |  dname
-----+-----+-------+---------
   1 |   1 | bob   | art
   2 |   2 | jeff  | history
   1 |   3 | steve | art
(3 rows)
```

As we can see, an initial join needs to be made, but from now on the statement
SELECT * FROM employee_department;
can be used to bring this table up.

Views that contain different tables cannot be used to insert data directly into the base tables. **An INSTEAD OF trigger** can be used to insert data into the appropriate base tables in the appropriate order.

For this example, the insert function must ensure department info is inserted first to avoid any constraint conflicts.

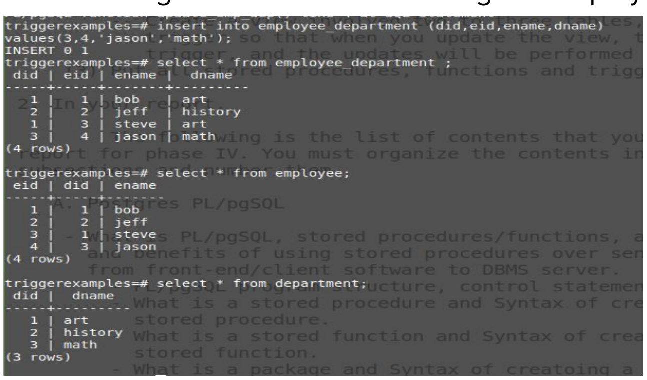
Insert Function:

```
CREATE OR REPLACE FUNCTION update_emp_dep() RETURNS TRIGGER AS $$
        BEGIN
                INSERT INTO department (did,dname) VALUES (new.did,new.dname);
                INSERT INTO employee(eid,did,ename) VALUES(new.eid,new.did,new.ename);
                RETURN new;
        END;
$$ LANGUAGE plpgsql;
```

Trigger

```
CREATE TRIGGER insert_emp_dep
        INSTEAD OF INSERT ON employee_department
        FOR EACH ROW EXECUTE PROCEDURE update_emp_dep ();
```

To demonstrate this, an employee by the name of jason is inserted into the view using
Insert into employee_department(did,eid,ename,dname)
values(3,4,'jason','math');
The INSTEAD OF trigger fires and two inserts are performed.
The following shows a view containing the employee and department.



```
triggerexamples=# insert into employee_department (did,eid,ename,dname)
values(3,4,'jason','math');
INSERT 0 1
triggerexamples=# select * from employee_department ;
 did | eid | ename |  dname
-----+-----+-------+---------
   1 |   1 | bob   | art
   2 |   2 | jeff  | history
   1 |   3 | steve | art
   3 |   4 | jason | math
(4 rows)

triggerexamples=# select * from employee;
 eid | did | ename
-----+-----+-------
   1 |   1 | bob
   2 |   2 | jeff
   3 |   1 | steve
   4 |   3 | jason
(4 rows)

triggerexamples=# select * from department;
 did |  dname
-----+---------
   1 | art
   2 | history
   3 | math
(3 rows)
```

4.3 PL/pgSQL-Like Languages in Microsoft SQL Server, MySQL and Oracle DBMS

A language is an expression of information structured by syntax, which includes keywords and identifiers as well as ways to manipulate data. Database management system (DBMS) language is a domain specific language to describe desired set of behaviors in a database. The discussion of such language is divided as follows:

SQL core syntax set:

select, insert, delete, update tables ...

Other syntax set, which describe the dynamic behavior of data:

- 4.3.1 Selective statements
- 4.3.2 Repetitive statements
- 4.3.3 Statements for creating Stored procedures/functions/triggers.

It is worth discussing the design philosophy of three powerful DBMS and their languages to gain knowledge and become better database engineers. Oracle DBMS was the first Relational implemented Database System in 1980s, it has been considered a high-end commercial DBMS product with supreme quality and can be quite expensive. For over 25 years, it has been implemented mostly in C and some assembly language for efficiency purpose. As such, its language feature, Oracle PL-SQL was developed from ADA. While C/C++ was developed to simplify ADA syntax in the 1980s, Oracle PL-SQL had decided to keep most ADA language features to allow Database developers the maximum freedom to squeeze every last ounce of performance. Full edition of ADA features in PL-SQL also allow a better software maintenance for Database developers. With its full feature including package, exception handling, in/out parameter setting, Oracle PL-SQL developer may tailor the semantics of language at a fine-grained level, while making it reusable at the same time.

Microsoft SQL-Server was originally designed by Sybase and later improved by Microsoft. The goal was to make a "competitive market share against IBM". The software developing cost per feature is significantly lower than Oracle, making it possible for massive distributions while remaining budget friendly for customers.

The language feature is designed to be as possible. The language, Transact-SQL is heavily coupled with a user-friendly graphic interface, "Microsoft SQL Server Management Studio". One can create the complete database, with all its constraints, simply by mouse drag and click. The studio may automatically generate the corresponding T-SQL statements base on mouse actions.

MYSQL Database System was a complete open-source DBMS, but its license was purchased by Sun Microsystems, and later Oracle Corporates for marketing purpose. Oracle Corporates has been intentionally limiting the scope of open source packages, making it a "reduced DBMS system" with the competitive price range of Microsoft SQL Server. The language currently lacks a few new features in Oracle Enterprise DBMS and PostgreSQL.

The most current SQL standard, SQL:2016, may be considered as an intersection set among all DBMS languages. There is proprietary software available to convert an DMBS language from one to another.

4.3.1 Selective Statements

Oracle DBMS:

```
IF condition1 THEN
 {...statements to execute when condition1 is TRUE...}
ELSIF condition2 THEN
 {...statements to execute when condition1 is FALSE and condition2 is TRUE...}
ELSE
 {...statements to execute when both condition1 and condition2 are FALSE...}
END IF;
```

MySQL DBMS:

Note: it is identical to PL-SQL syntax

```
IF condition1 THEN
    {...statements to execute when condition1 is TRUE...}
    [ ELSEIF condition2 THEN
    {...statements to execute when condition1 is FALSE and condition2 is TRUE...} ]
[ ELSE
    {...statements to execute when both condition1 and condition2 are FALSE...} ]
END IF;
```

Microsoft SQL-Server DBMS: T-SQL

Note: T-SQL does not support else if condition statement

```
IF condition
    {...statements to execute when condition is TRUE...}
ELSE
    {...statements to execute when condition is FALSE...}
```

4.3.2 Repetitive Statements

Oracle DBMS, has Loop, For loop, cursor for loop, while loop, repeat until loop, exit statement

In PL-SQL, loop is used instead of while

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

Note: inside the statements, an exist is required to end this loop

Or, use a while loop

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

Or, The for loop may be used when the number of iteration is certain

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

MySQL DBMS:

Mysql has loop, while, repeat until, iterate, leave, and return. Iterate Statement: The ITERATE statement is used when you are want a loop body to execute again. It is used within the LOOP statement, WHILE statement, and REPEAT statement. Leave statement; same as “exit” in PL-SQL.

Microsoft SQL-Server DBMS, has While loop, break, and continue.

Note: There is no for loop in T-SQL.

Alternative simulation by while syntax:

```
DECLARE @cnt INT = 0;
WHILE @cnt < cnt_total
BEGIN
    {...statements...}
SET @cnt = @cnt + 1;
END;
```

4.3.3 Statements for Creating Stored Procedures/Functions/Triggers.

Oracle DBMS

Procedures/Functions:

```
CREATE [OR REPLACE] PROCEDURE procedure_name
    [ (parameter [,parameter]) ]
    IS
     [declaration_section]
    BEGIN
        executable_section
    [ EXCEPTION
        exception_section ]
    END [procedure_name];
```

Triggers:

PL-SQL has {before,after} {insert,update,delete} trigger, and all 6 are implemented in PL-SQL.

```
CREATE [ OR REPLACE ] TRIGGER trigger_name
    BEFORE {insert, update, delete}
    AFTER { insert, update, delete}
            ON table_name
            [ FOR EACH ROW ]
    DECLARE
                    -- variable declarations
```

```
BEGIN
        -- trigger code
EXCEPTION
    WHEN ...
        -- exception handling
END;
```

Notable Restrictions on PL-SQL Triggers:

You can not create before, or after trigger on a view. Before triggers - You can update the :NEW values, not the :OLD values. After triggers you cannot update the :NEW or the :OLD values.

MySQL DBMS

Procedures/Functions:

```
CREATE FUNCTION function_name
[ (parameter datatype [, parameter datatype]) ]
RETURNS return_datatype
BEGIN
    declaration_section
    Executable_section
END;
```

Triggers:

```
CREATE TRIGGER trigger_name
 BEFORE ...
AFTER   ...
        ON table_name FOR EACH ROW
BEGIN
```

```
-- variable declarations

-- trigger code

END;
```

Notable Restrictions Triggers:

Same as PL-SQL

Microsoft SQL-Server DBMS

Procedures/Functions:

```
CREATE FUNCTION [schema_name.]function_name

( [ @parameter [ AS ] [type_schema_name.] datatype

  [ = default ] [ READONLY ]

 , @parameter [ AS ] [type_schema_name.] datatype

  [ = default ] [ READONLY ] ]

)

RETURNS return_datatype

[ WITH { ENCRYPTION

     | SCHEMABINDING

     | RETURNS NULL ON NULL INPUT

     | CALLED ON NULL INPUT

     | EXECUTE AS Clause ]

[ AS ]

BEGIN

     [declaration_section]

     executable_section
```

```
        RETURN return_value
    END;
```

Triggers:

```
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
```

Notable Restrictions Triggers:

Same as PL-SQL

It is important to keep in mind, DBMS is a market heavily under the influence of marketing. Although, there are SQL Standard being published**, most DBMS vending chose not to comply 100% for the following reasons:**

1. SQL itself is a complex language set, means that most implementers do not necessary support the entire standard.
2. The standard does not standardized database behavior, such as schema vs. user management, data ownership and segregation, indexing, file storage at disk-level.
3. Most DBMS vendors have large pool of customers, using pre-existing edition of Database related software, where the most current SQL standard may conflict with the prior behavior of old database system.
4. For commercial reason, “vendor lock-in” is preferred from vendor’s perspective.

Phase 5 Graphical User Interface

5.1 General Description

The preceding chapters have described in depth the implementation of a database designed for a vending machine company. The design process has gone through multiple stages to form a final database schema. A graphical user interface (GUI) is a computer application that is meant to be accessible and easy to use for users. The simpler the design, the better off the user will be. We as designers have decided to implement a web-based application that is meant to support three user groups:

A dispatcher, a client, and a driver.

I, Edwin Gonzalez, implemented the Dispatcher user group GUI.

5.1.1 Dispatcher User Group

The dispatcher is in charge of placing orders, reviewing orders, performing general bookkeeping, and creating routes for the drivers. At the time of this publication, the current application supports ordering, order reviewing, and report generation.

The following breaks down the generalized interface of the application.

- User logs in through a pop-up login menu.
- The user is greeted with a homepage and a utility side panel.
  This side panel is used to perform most major tasks the user needs.
- Two main buttons provide a quick report generation and the side panel.
  supports a customization report generation.

A later section in this chapter will break down the user experience more clearly.

To understand the application better, it is important to know what a dispatcher generally does.

General Work of a Dispatcher

- A dispatcher is able to look up a history of every order placed by the company and find a detailed report of each and every order.
- They must be able to place an order of any number of items from a select group of suppliers and be given some receipt of the transaction for bookkeeping.
- The dispatcher must be able to generate reports as well as look up previous order reports. This is useful when checking expenditures or the accuracy of past/current orders.
- The above general descriptions all blend together to create a workflow that can be generalized as such → Place an Order, review order, manage orders.

5.2 Functionalities of the Application

5.2.1 Itemized description of the application

This user-interface is implemented as a web application. Three key components are used to develop a web-based application: server-side programming, middle-tier programming, and client-side programming. The database management system Postgresql is used to store the applications data and is hosted on the California State University of Bakersfield's server, delphi.cs.csub.edu.

To help speed up the development process, HTML templates are used to layout generic pages that the are modified to fulfill the needs of the dispatcher user-group. The section below will provide screenshots with brief descriptions of the application.

5.2.1.2 Screenshots

Figure 1. User Login Screen. A session variable holding the username is created

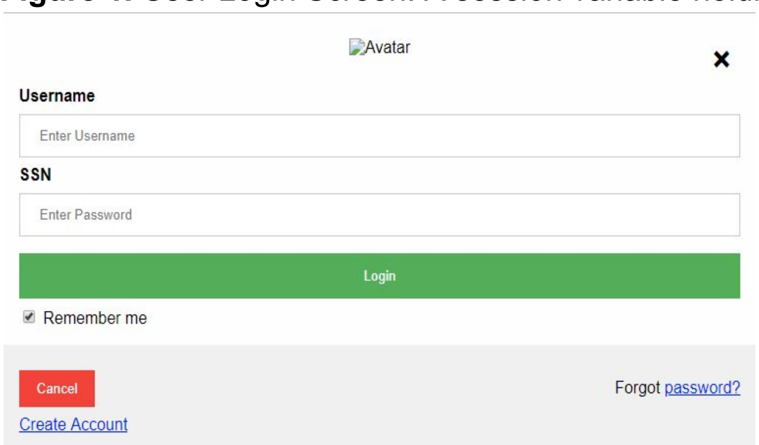


Figure 2. Main Screen displaying utility side panel and report generation buttons

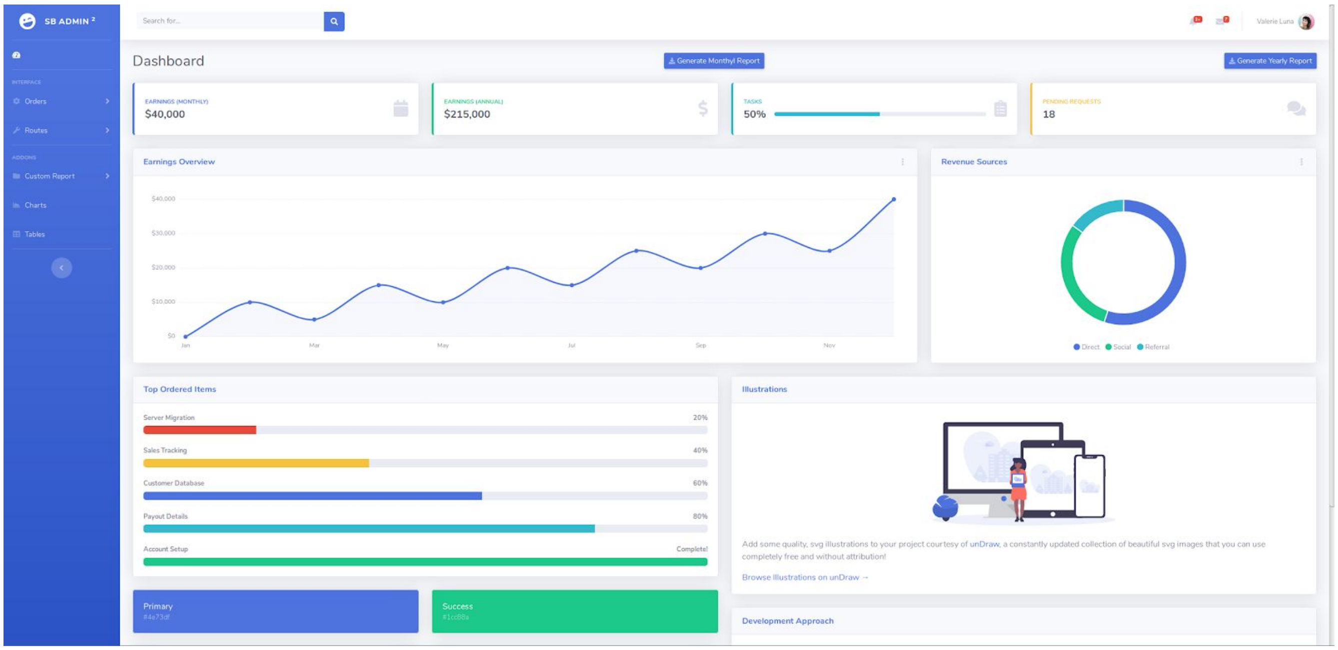
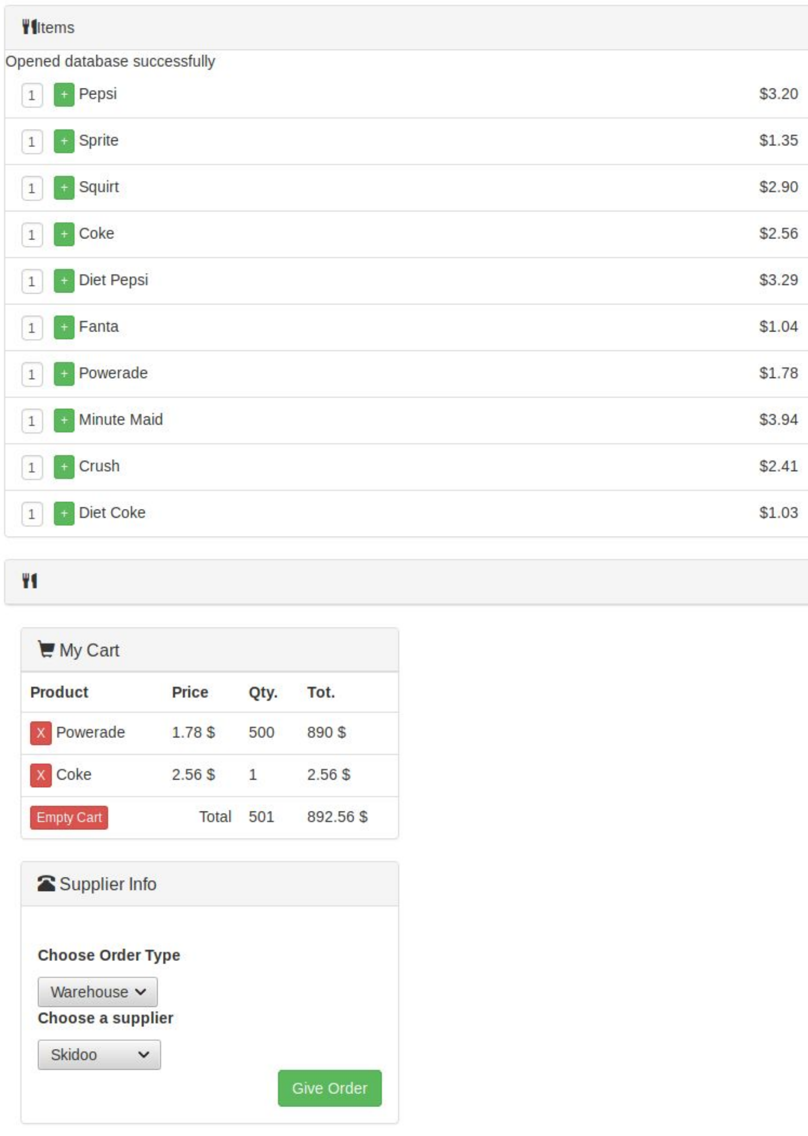


Figure 3. Drop down Menu: Order Placement

A user can enter the number of items they need, choose an order type, and choose from a supplier



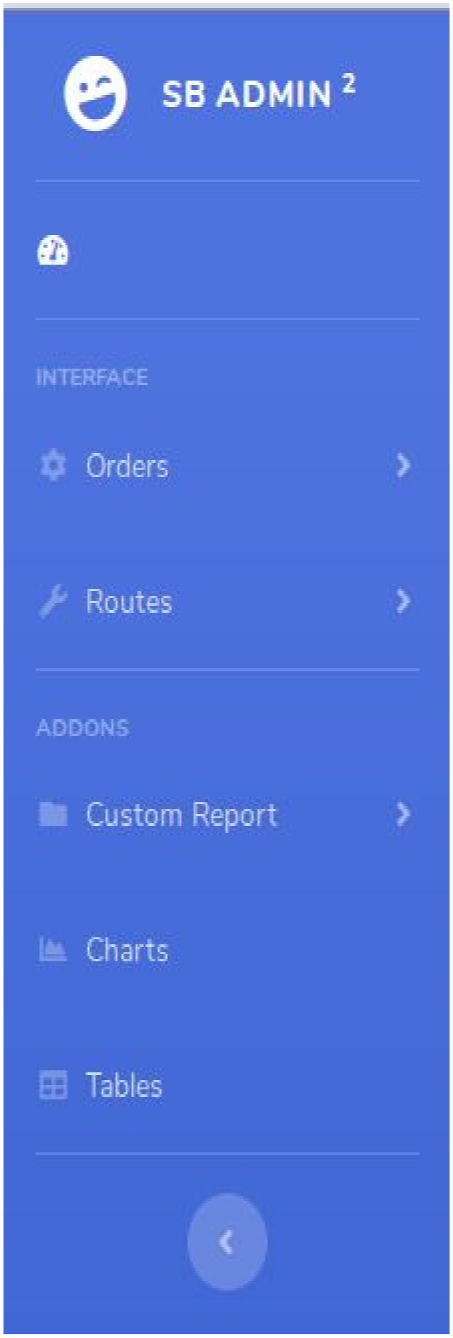


Figure 4. Side Panel

Orders - Brings up a drop down menu to place or view an order as well as generate an item report. Can also bring up an order history table.

Routes - used to bring up a route history

Custom Report - User can create two custom reports. An Items report or a overall expenditure report.

Figure 5. A receipt for a single placed order. Shows the supplier information at the top let, the items and the purchase information on the bottom and a grand total at the top right.

ARCHIVIST

CAPITAL

B i l l o f S a l e

Supplier Information:
Company ID: 7
Phone: 818-175-5739
Address: Packers, Glendale, CA
Zip: 11117

Company Name: Skidoo

Invoice #	191
Date	May 20, 2019
Amount Due	$892.56

Item	Description	Unit Cost	Quantity	Price
Powerade	Test Description Here	1.78	500	$890
Coke	Test Description Here	2.56	1	$2.56

T E R M S

NET 30 Days. Finance Charge of 1.5% will be made on unpaid balances after 30 days.

Figure 6. A general overall Items Report. The report details each item and the amount bought from each supplier. A total for each supplier is given as well as a grand total.

I t e m s / S u p p l i e r   R e p o r t

Archivist Capital
Address: 5555 Bakersfield, CA
93304
Phone: 661-555-5555
Summary:
A report on the expenditures per
supplier per item.

Company Name: Skidoo

Month of	May 2019
Date Generated	05-20-19
Grand Total Spent	$48625.43

Supplier	-------------Items---------	Total Purchases	MSRP	Expendetures
Lajo	Pepsi	394	$3.20	$1260.80
Lajo	Sprite	6	$1.35	$8.10
Lajo	Squirt	17	$2.90	$49.30
Lajo	Coke	14	$2.56	$35.84
Lajo	Diet Pepsi	28	$3.29	$92.12
Lajo	Fanta	8	$1.04	$8.32
Lajo	Powerade	59	$1.78	$105.02
Lajo	Minute Maid	185	$3.94	$728.90
Lajo	Crush	3	$2.41	$7.23
Lajo	Diet Coke	36	$1.03	$37.08
Total Price	$2332.71			
-----------	-----------	-----------	-----------	-----------
Cogidoo	Pepsi	34	$3.20	$108.80
Cogidoo	Sprite	301	$1.35	$406.35
Cogidoo	Squirt		$2.90	$0.00
Cogidoo	Coke		$2.56	$0.00
Cogidoo	Diet Pepsi		$3.29	$0.00
Cogidoo	Fanta		$1.04	$0.00
Cogidoo	Powerade	207	$1.78	$368.46
Cogidoo	Minute Maid	148	$3.94	$583.12
Cogidoo	Crush	137	$2.41	$330.17
Cogidoo	Diet Coke		$1.03	$0.00
Total Price	$1796.90			
-----------	-----------	-----------	-----------	-----------
Quaxo	Pepsi	114	$3.20	$364.80
Quaxo	Sprite	348	$1.35	$469.80
Quaxo	Squirt	311	$2.90	$901.90
Quaxo	Coke		$2.56	$0.00
Quaxo	Diet Pepsi		$3.29	$0.00

Figure 7. An overall monthly expenditures report. This report shows all suppliers that have sold to the vending machine as well as the total amount spend on each supplier. The most ordered item is displayed and the total expenditure are given.

ARCHIVIST

CAPITAL

Monthly Report

Archivist Capital
Address: 5555 Bakersfield, CA
93304
Phone: 661-555-5555
Summary:
A report on the expenditures per
monthly supplier.

Company Name: Archivist

Month of	May 2019
Date Generated	05-20-19
Amount Due	$14732.35

Supplier	-------------Information----------	Total Purchases	Most Ordered Item	Expendetures
Lajo	__213-882-2592	275	Squirt	$818.76
Cogidoo	__209-341-5236	100	Minute Maid	$394.00
Youspan	__818-544-7711	2275	Minute Maid	$6,459.70
Skidoo	__818-175-5739	1506	Powerade	$3,245.89
Kwinu	__213-538-2484	1100	Minute Maid	$3,814.00

TERMS

NET 30 Days. Finance Charge of 1.5% will be made on unpaid balances after 30 days.

Figure 8. Yearly Report. Same as monthly but over a year span from the date generated.

ARCHIVIST
CAPITAL

Y e a r l y   R e p o r t

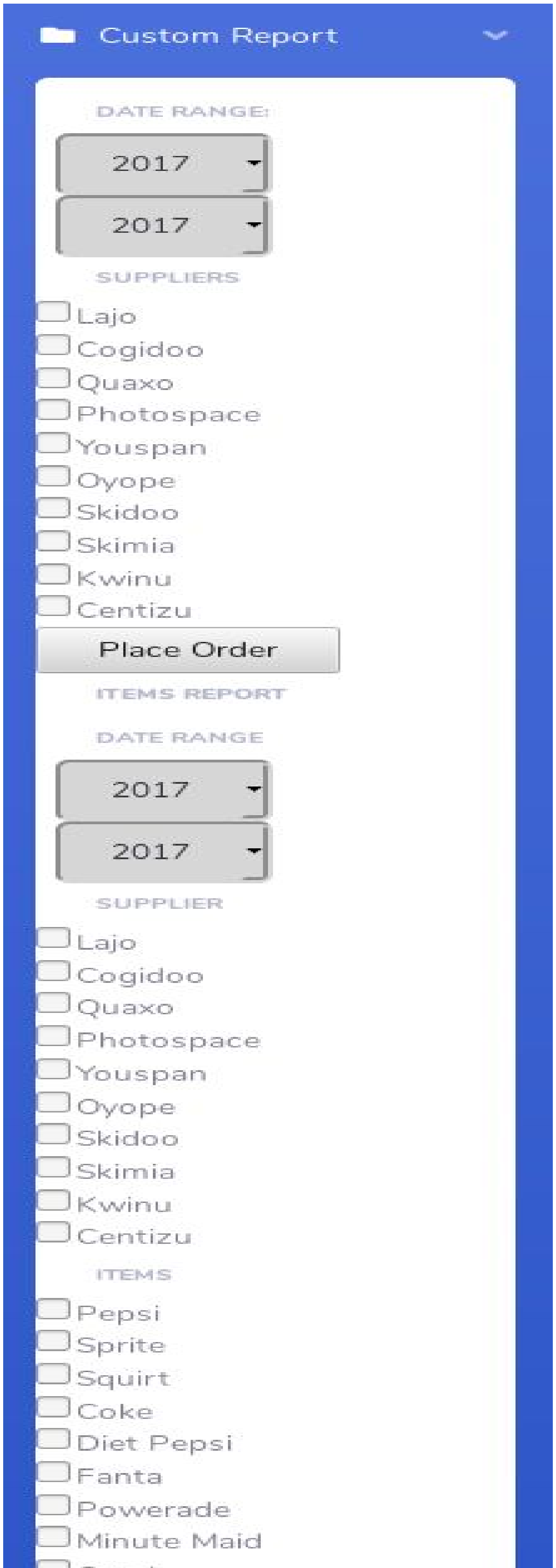
Archivist Capital
Address: 5555 Bakersfield, CA
93304
Phone: 661-555-5555
Summary:
A report on the expenditures per
yearly supplier.

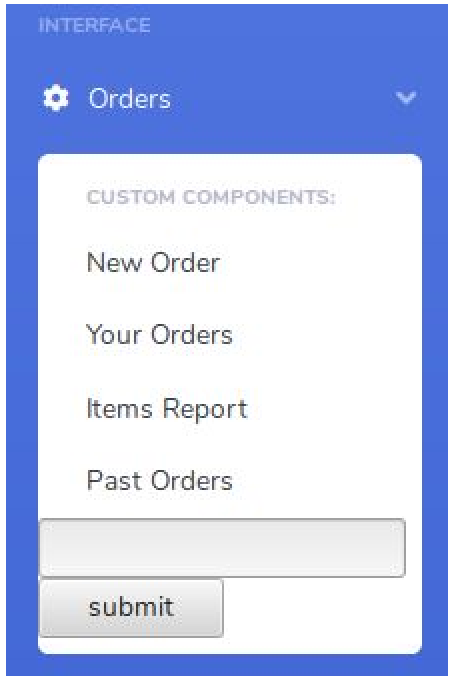
Company Name: Archivist

Year of	May 2019
Date Generated	05-20-19
Grand Total	$28024.09

Supplier	-------------Information----------	Total Purchases	Most Ordered Item	Total Expendetures
Lajo	213-882-2592	275	Squirt	$818.76
Cogidoo	209-341-5236	100	Minute Maid	$394.00
Quaxo	818-647-8909	354	Powerade	$876.56
Photospace	559-779-8191	866	Fanta	$2,875.18
Youspan	818-544-7711	2751	Powerade	$7,927.02
Oyope	310-529-6251	894	Fanta	$2,353.64
Skidoo	818-175-5739	1753	Squirt	$3,688.21
Skimia	858-574-6504	678	Diet Pepsi	$2,038.76
Kwinu	213-538-2484	1100	Minute Maid	$3,814.00
Centizu	559-270-0717	1452	Diet Coke	$3,237.96

Figure 9. Drop down Menus for Orders and custom orders





On Left:
The Custom Report has two options. The first is to generate a revenue report based on a custom date range and a specified supplier list.

The second report is an Items Report. A user can select a date range, choose a specific amount of suppliers, and choose which items they want to see have been purchased.

Above:
The drop down Orders box allows a user to place a new order, view the order history, or type in a specific order number to view that order receipt.

Figure 10. Order History. The user can search for any order by a main column attribute or can click the main column attribute to sort by ascending or descending order.

Show 10 entries Search:

OrderID	SupID	Type	SupName	Phone	Street	City	State	Zip	BadgeNum	TimePlaced	DatePlaced
191	7	TESTTYPE	Skidoo	818-175-5739	Packers	Glendale	CA	11117	1	17:45:25.176221	2019-05-20
190	5	TESTTYPE	Youspan	818-544-7711	Walton	North Hollywood	CA	11115	1	13:20:10.111286	2019-05-19
189	1	TESTTYPE	Lajo	213-882-2592	Mosinee	Los Angeles	CA	11111	1	13:19:19.206606	2019-05-19
188	5	TESTTYPE	Youspan	818-544-7711	Walton	North Hollywood	CA	11115	1	08:47:39.813206	2019-05-19
187	2	TESTTYPE	Cogidoo	209-341-5236	Daystar	Stockton	CA	11112	1	08:35:22.194278	2019-05-19
186	5	TESTTYPE	Youspan	818-544-7711	Walton	North Hollywood	CA	11115	1	08:03:01.972224	2019-05-19
185	9	TESTTYPE	Kwinu	213-538-2484	Chive	Los Angeles	CA	11119	1	07:02:26.114274	2019-05-19
184	1	TESTTYPE	Lajo	213-882-2592	Mosinee	Los Angeles	CA	11111	1	03:00:20.267123	2019-05-19
183	1	TESTTYPE	Lajo	213-882-2592	Mosinee	Los Angeles	CA	11111	1	03:45:39.708665	2019-05-18
182	1	TESTTYPE	Lajo	213-882-2592	Mosinee	Los Angeles	CA	11111	1	00:32:20.39303	2019-05-18

Showing 1 to 10 of 96 entries

Previous 1 2 3 4 5 … 10 Next

5.2.1.3 Tables, Views, Stored Subprograms

Tables

Supplier(supplierid, name, phone, streetname, city, state, zip)

Orders(orderid, supplierid, ordertype)

placesOrder(orderid, badgenumber, timeplaced, dateplaced)

orderContains(itemtypeid, orderid, numitemtype, itemtypeprice, expdate)

itemtype(itemtypeid, itemtypename, msrp)

employee(...)

Views

- **realYear** - a natural join of supplier, orders, placesOrder, orderContains, and itemType for the current year.
- **monthlyComplete** -the same above for the current month
- **allOrderData** - the same as above but for all of the data's history.

Subprograms

- **customTotal**
- **customTotalItemsBought**
- **mostBought**

5.2.2 Programming Sections

5.2.1 Server-side programming

Server-side programs run on the hosting server and are not accessible or seen by the client-side. These programs deal with manipulating data on the server, ensuring data integrity, and making sure the correct data is sent back to the client-side.

The Postgresql database management system is hosted on the server and contains queries and views that are requested by the client-side. Bellow are code snippets of these views/subprograms.

View: realyear
Purpose: Creates a view of an entire order and its relationships in the current year
Query:

```
View definition:
 SELECT ordercontains.itemtypeid, placesorder.orderid, orders.supplierid, placesorder.badgenumber, placesorder.timeplaced, placesorder.d
ateplaced, orders.ordertype, supplier.name, supplier.phone, supplier.streetname, supplier.city, supplier.state, supplier.zip, orderconta
ins.numitemtype, ordercontains.itemtypeprice, ordercontains.expdate, itemtype.itemtypename, itemtype.msrp
   FROM placesorder
NATURAL JOIN orders
NATURAL JOIN supplier
NATURAL JOIN ordercontains
NATURAL JOIN itemtype
  WHERE placesorder.dateplaced >= (now() - '1 year'::interval);
```

View: monthylcomplete
Purpose: Creates a view of an entire order and its relationships for a month
Query:

```
View definition:
 SELECT ordercontains.itemtypeid, placesorder.orderid, orders.supplierid, placesorder.badgenumber, placesorder.timeplaced, placesorder.d
ateplaced, orders.ordertype, supplier.name, supplier.phone, supplier.streetname, supplier.city, supplier.state, supplier.zip, orderconta
ins.numitemtype, ordercontains.itemtypeprice, ordercontains.expdate, itemtype.itemtypename, itemtype.msrp
   FROM placesorder
NATURAL JOIN orders
NATURAL JOIN supplier
NATURAL JOIN ordercontains
NATURAL JOIN itemtype
  WHERE placesorder.dateplaced >= date_trunc('month'::text, 'now'::text::date::timestamp with time zone) AND placesorder.dateplaced >= d
ate_trunc('year'::text, 'now'::text::date::timestamp with time zone)
  ORDER BY placesorder.orderid DESC;
```

View: allorderdata
Purpose: Creates a view of an entire order and its relationships for all time frames.
Query: Same as the realyear but without the date constraints.

The views are used in conjunction with functions to return specific queries.
A few examples are given below to demonstrate

<u>Functions</u>

Function: customTotal
Purpose: Returns the amount spent during a given timeframe on a supplier
Query:

```
begin

return query

select sum (sum) from (select orderid, sum(numitemtype * itemtypeprice)
from testall where supplierid = sid and extract(year from dateplaced)::int >= sdate
and extract(year from dateplaced)::int <= edate

group by orderid, numitemtype, itemtypeprice) as customtotal;

end
```

Function: customTotalItemsBought
Purpose: Returns the total number of items bought for a specific item type from a supplier
Query:

```
begin

return query

select sum (sum) from (select orderid, sum(numitemtype)

from testall where supplierid = sid and itemtypeid = id

and extract(year from dateplaced)::int >= sdate

and extract(year from dateplaced)::int <= edate

group by orderid, numitemtype, itemtypeprice) as customtotalitemsbought;
end
```

Function: mostBought
Purpose: Returns the most bought itemtype from a supplier
Query:

```
begin
return query
select itemtypename from itemtype natural join (select itemtypeid, count(*) as frequency
from testall where supplierid = sid and extract(year from dateplaced)::int >= sdate
and extract(year from dateplaced)::int <= edate group by itemtypeid
order by count(*) desc
fetch first 1 row only) as mostbought;
end
```

5.2.2 Middle-Tier Programming

PHP is a server scripting language that allows server connection. HTML forms send http request which are handled by the PHP code. The php code is used to query a server, retrieve data, and send it back to the clients that need them. PHP is also used to create session variables. These variables are stored by the browser's cache and allow variable values to exist across multiple webpages. Session variables can be online shopping carts, user login credentials, or other values that need to be shared across pages.

The following code snippet establishes a database connection

```php
<?php
$host        = "host = localhost";
$port        = "port = 5432";
$dbname      = "dbname = egonzale";
$credentials = "user = egonzale password=Yid8lav";

$db = pg_connect( "$host $port $dbname $credentials"  );
if(!$db) {
    echo "Error : Unable to open database\n";
} else {
    echo "Opened database successfully\n";
}

?>
```

This code snippet shows functions which contains views being used to calculate data for a customs items report

```
while($row = pg_fetch_row($ret)) {
    foreach($_SESSION['csname'] as $a) {
        if($row[1] == $a) {
            $suptotal = 0;
            $displayitem = "select * from itemtype"; $displayitemq = pg_query($db, $displayitem);
            while($displayrow = pg_fetch_row($displayitemq)){
                foreach($_SESSION['ciname'] as $b){
                    if($displayrow[1] == $b) {
                        $id = $displayrow[0];
                        $itemid = $displayrow[0];
                        //Query monthyl most bought item
                        $item = "select * from mostboughtitem($id,$intsd,$inted)";
                        $itemquery = pg_query($db,$item);
                        $itemrow = pg_fetch_row($itemquery);

                        $total = "select * from customtotal($id,$intsd,$inted)";
                        $totalquery = pg_query($db, $total);
                        $totalrow = pg_fetch_row($totalquery);
                        $num = preg_replace('/[^0-9]/','',$row[6]);
                        $totals = $row[2] * $num;
                        $sum =  money_format('%i', ($totals/100));

                        $totalm = "select * from customtotalitemsbought($id,$intsd,$inted,$id)";
                        $totalmq = pg_query($db, $totalm);
                        $retq = pg_fetch_row($totalmq);

                        //add msrp and expend
                        $msrp = preg_replace('/[^0-9]/', '', $displayrow[2]);
                        $totalpurchases = $retq[0];
                        $msrp = $msrp * $totalpurchases;
                        $expend = money_format('%i', ($msrp/100));
                        echo"<tr class=\"item-row\">";
                        echo"<td class=\"item-name\"><div class=\"delete-wpr\"><p>$row[1]</p></div></td>";
                        echo "<td class=\"description\"><p>$displayrow[1]</p></td>\n";

                        echo"<td><p class=\"cost\">                           $retq[0]</p></td>";
                        echo   "<td><p class=\"qty\">                         $displayrow[2]</p></td>";
                        echo              "<td><span class=\"price\">$$expend</span></td>";
                        echo "\n";
                        echo "</tr>";

                        $suptotal = $suptotal + $msrp;

                        $numbers = preg_replace('/[^0-9]/', '', $totalrow[0]);
                        $month = $month + $numbers;
                    }
                }
            }
            $suptotal = money_format('%i', ($suptotal/100));
```

5.2.3 Client-side programming

The client-side programming consist of HTML/CSS combined with the scripting language JavaScript. HTML is a standard markup language that is used to create web pages/applications. CSS style-sheets are used style a generic HTML page. Javascript is a scripting language that is used with HTML to make web pages more dynamic. Javascript can add clickListeners, can create dynamic boxes, and add animations.

Bellow is a snippet that uses bootstrap and javascript to create a searchable, paginated table

```
<script>
$(document).ready(function(){
    $('#myTable').dataTable();
});
</script>
```

5.3 Survey Questions

Promt

In this course, in what degree do you think you have the achieved each of following 4 outcomes? Your answer is between 1 (lowest) and 10 (highest). Each member of the team should have his/her own answers.

Outcome	Answer (1-10)
An ability to analyze a problem, and identify and define the computing requirements and specifications appropriate to its solutions.	8
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.	10
An ability to communicate effectively with a range of audiences. An ability to write a technical document such as a software specification white paper or a user manual.	9
An ability to apply mathematical foundations, algorithmic principles, and computer science theory in the modeling and design of computer-based systems in a way that demonstrates comprehension of the tradeoffs involved in design choices.	7