Bakersfield Flower Shop

CS 3420 Spring 2020

Dr. Huaqing Wang

Group 4

Joseph Shafer

Matthew Gonzales

Jacqueline Peralta

Table of Contents

Phase 1: Data Collection and Conceptual Database Design

Our project's focus will be to design a database system for Bakersfield Flowershop. Prior to designing its database system, we will have to do research to determine the business's day to day operations which will give us the necessary knowledge to design a database that will best serve Bakersfield Flowershop and their customers.

This phase of our database report will go into detail how we learned about our organization and translated it into conceptual database design. We'll talk about how we identified the necessary entities to run a flower shop and the ways these entities interact with each other. We will conclude this phase with a Entity-Relationship (ER) model to visually explain our conceptual database design.

1.1 Fact-Finding and Data Collection

We, as the designers, need to have full understanding as to how our business operates. This will allow us to accurately design a database that will suit our business's needs. This section will be about our research methods used to gather and collect information about flower shops.

A brief description of the organization and an explanation of the research process will be given. Itemized descriptions of the entities and relationships that build our design will also be discussed.

1.1.1 Introduction to Organization

Bakersfield Flowershop is a fictitious flower shop for which we are designing a database for. Flower shops supply customers with flowers, bouquets, and other floral arrangements for all types of occasions. Additional services such as delivery are also offered. Availability of certain flower products depends on the seasons. They are handled with care and constantly maintained to ensure freshness to achieve customer satisfaction.

Our customers will have the option to place an order online or in-person. They will also have the option to create an account on our website or proceed with guest checkout. Employees at our flower shop will take customers' orders for our florists to fulfill them. Delivery services are an option only for recipients living in Bakersfield.

1.1.2 Description of Fact-Finding Techniques

In order for our group to create the Database for Bakersfield Flowershop, we needed to get a greater understanding of the business flow for a flower shop. We decided to search for information from already established shops in the Bakersfield area. One of our group members interviewed employees of Mt. Vernon Florist and managed to obtain a better insight as to how they handle their orders, particularly those made in-person. We also researched online and looked at several flower shops' websites to learn more about how online orders are handled.

In addition, we emailed surveys out to the owners of several flower shops in Bakersfield. The list of questions we emailed to flower shops are as follows: If you have a database, what kind of information do you store in your current database (i.e customer addresses, phone numbers, ect)? If you do not currently have a database, what kind of information would you want to keep track of? Do you have reports you generate, and what kind of information is required to generate this report? If you do not, what kind of information would you want in a report? What do you like about the design of your current website? What don't you like and would you want to change? From the shops that responded we learned valuable information that helped in the conceptual design of our database.

1.1.3 Scope of Database

Our database's model represents what a Customer to Bakersfield Flowershop would need to make orders of our products, and what an employee would need to fulfill those orders. This includes designing a system for deliveries customers can make that store employees can then fultill.

The end goal of the design of our database is to create a front end application that customers can use to accomplish this. Our database will also model what an employee of Bakersfield Flowershop needs to refill our products we sell to customers which we will also integrate into our front end. The scope of our database is to cater to customers and allow employees to smoothly do their job.

1.1.4 Itemized Descriptions of Entity Types and Relationship Types

Once all the information has been gathered on the structure and organization of the company, the data will be represented as entity sets and relationship sets to create an Entity-Relationship (ER) model. The entity and relationship sets will be explained in further detail.

For each entity type, there will be a description of every attribute and an example has to how each entity type relates to others. For each relationship set, the related entity types and constraints of the relationship will be listed and an example of the relationship will also be given.

Descriptions of Entity Types

Entity Name: Customer

Attribute: customer_id

- Type: integer
- Meaning: Unique number associated with each customer.
- Instance: 12345

Attribute: name

- Type: String
- Meaning: Holds the customer’s name.
- Instance: John Doe

Attribute: address

- Type: String
- Meaning: Holds the customer’s address.
- Instance: 123 Main St. Bakersfield, CA 93301

Attribute: username

- Type: String
- Meaning: Holds the customer’s username
- Instance: JohnDoe70

Attribute: password

- Type: String
- Meaning: Holds the hashed password of the customer.
- Instance: gHje1J09pK

Attribute: email

- Type: String
- Meaning: Holds the customer’s email address.
- Instance: JDoe70@email.com

Attribute: acc_creation_date

- Type: Datetime
- Meaning: Holds the date and time the account was created.
- Instance: 10/20/2018 13:25:11

Attribute: phone_number

Type: String

Meaning: Holds the customer’s phone number.

Instance: (661)555-5555

A **Customer** visits our store to buy products or is able to make orders for products from the Bakersfield Flowershop website.

Entity Name: Delivery Address

Attribute: address_id

Type: Integer

Meaning: Holds a unique id for each address for deliveries.

Instance: 0534701

Attribute: address

Type: String

Meaning: Holds the recipient’s address

Instance: “4615 Polo View Drive Bakersfield, CA, 93312”

An online customer inputs a **Delivery Address** for their product order.

Entity Name: Employee

Attribute: employee_id

Type: Integer

Meaning: Holds the employee’s unique id number.

Instance: 1191021

Attribute: name

Type: String

Meaning: Holds the employee’s name.

Instance: Jane Doe

Attribute: address

Type: String

Meaning: Holds the employee’s address

Instance: 181 24th St. Bakersfield, CA 93302

Attribute: phone_number

Type: String

Meaning: Holds the employee’s phone number.

Instance: (661) 555-5050

A **Employee** of the store’s responsibilities include taking orders from in-store customers, packaging products for deliveries, and delivering orders.

Entity Name: Flower Product

Attribute: product_id

Type: Integer

Meaning: Unique id for each flower product

Instance: 101

Attribute: product_name

Type: String

Meaning: Holds the name of product/flower

Instance: “Rose”

Attribute: sell_price

Type: decimal

Meaning: Holds the current selling price of the product to the customer.

Instance: “9.99”

Attribute: purchase_price

Type: decimal

Meaning: Holds the price the product was purchased at from the supplier.

Instance: “2.99”

Attribute: color

Type: String

Meaning: Holds the color of the flower

Instance: “Red”

Attribute: length

Type: String

Meaning: Holds the length of the flower

Instance: “6 in.“

Attribute: product_image

Type: String

Meaning: Holds a filename to the image

Instance: “redrose.png”

Attribute: description

Type: String

Meaning: Holds a brief description of the product.

Instance: “A beautiful red rose, thornless.”

A **Flower Product** is a product available to customers of Bakersfield Flowershop.

Entity Name: Incoming Payment

Attribute: incoming_id

Type: Integer

Meaning: Holds the id associated with the transactions made online or in the store.

Instance: “10039214”

Attribute: sales_tax

Type: Float

Meaning: Holds the percentage of sales tax charged in Bakersfield.

Instance: “8.25”

A **Incoming Payment** is a more specific payment related to customers paying Bakersfield Flowershop for product orders.

Entity Name: Order Status

Attribute: status_id

Type: Integer

Meaning: Holds a unique id for each status.

Instance: “405392342”

Attribute: status

Type: String

Meaning: Holds the current order status.

Instance: “In Process”

Customers can check the **Order Status** to see the current status of their order.

Entity Name: Package

Attribute: package_id

Type: Integer

Meaning: A unique identifier for each package.

Instance: 14312

Attribute: expected_time

Type: Datetime

Meaning: The date and time a customer can expect their delivery.

Instance: 02/14/2020 11:30:00

Attribute: message

Type: String

Meaning: This will hold a message relating to the delivery.

Instance: “Happy Valentines Day!”

A **Package** is for a product order that is sent out of the store.

Entity Name: Payment

Attribute: payment_id

Type: Integer

Meaning: Holds a unique id associated with each transaction.

Instance: “190123411”

Attribute: date

Type: datetime

Meaning: Holds the date and time of the payment.

Instance: “02/14/2020 11:34:52”

Attribute: amount

Type: decimal

Meaning: Holds the amount paid.

Instance: “49.99”

A **Payment** is an exchange of money in Bakersfield Flowershop.

Entity Name: Payment Type

Attribute: payment_type_id

Type: Integer

Meaning: Holds a unique id for every form of payment.

Instance: “1”

Attribute: description

Type: String

Meaning: Holds the name of the payment type.

Instance: “Cash”

A **Payment Type** is the method of paying used for a Payment.

Entity Name: Product Order

Attribute: p_order_number

Type: Integer

Meaning: Holds an order number associated with the product order.

Instance: “109222341”

Attribute: date

Type: Datetime

Meaning: Holds the date and time the order was placed.

Instance: 02/10/2020 16:22:12

A **Product Order** is made by customers and contains items from Flower Products that the customer has ordered.

Entity Name: Recipient

Attribute: recipient_id

Type: Integer

Meaning: Holds a unique id for every recipient.

Instance: “ 1001234483”

Attribute: name

Type: String

Meaning: Holds the recipient’s name.

Instance: “James Trickington”

Attribute: phone_number

Type: String

Meaning: Holds the recipient’s phone number.

Instance: “(661)555-0505

A **Recipient** is who receives a delivery.

Entity Name: Supplier

Attribute: supplier_id

Type: Integer

Meaning: Holds a unique id for each supplier.

Instance: “909011”

Attribute: vendor_name

Type: String

Meaning: Holds the name of the supplier.

Instance: “Flower Farm”

Attribute: address

Type: String

Meaning: Holds the address to the supplier.

Instance: 123 Flower Lane Arvin, CA 93203

Attribute: phone_number

Type: String

Meaning: Holds the supplier’s phone number.

Instance: “(661)505-0055”

A **Supplier** is who Bakersfield Flowershop purchases products from to then sell.

Entity Name: Outgoing Payment

Attribute: outgoing_id

Type: Integer

Meaning: Holds a unique id for transactions that occur with a supplier.

Instance: “99000123”

Attribute: supplier_invoice_id

Type: Integer

Meaning: Holds the suppliers invoice id.

Instance: “110092341”

An Outgoing **Payment** is a more specific type of payment, for when Bakersfield Flowershop pays a supplier to refill our products.

Entity Name: Supply Purchase Order

Attribute: supply_purchase_id

Type: Integer

Meaning: Holds a unique id for each purchase from a supplier.

Instance: “100012413”

Attribute: supply_purchase_time

Type: Datetime

Meaning: Holds the date and time that a supply purchased was placed.

Instance: “02/11/2020 18:01:11”

A **Supply Purchase Order** is the purchase of products an employee makes to refill products from a supplier.

Entity Name: Work History

Attribute: history_id

Type: Integer

Meaning: Holds a unique id of the employee work history.

Instance: “10003”

Attribute: start_date

Type: Datetime

Meaning: Holds the date the employee was hired”

Instance: “05/10/2018 00:00:00”

Attribute: end_date

Type: Datetime

Meaning: Holds the date the employee stopped working for the company.

Instance: “07/17/2019 00:00:00”

Attribute: job_title

Type: String

Meaning: Holds the employee’s job title.

Instance: “Sales Associate”

Attribute: pay_rate

Type: decimal

Meaning: Holds the employee’s current pay rate per hour.

Instance: “14.50”

Each employee has a **Work History** to track when they have worked for Bakersfield Flowershop.

Entity Name: Work Shift

Attribute: shift_id

Type: Integer

Meaning: Holds a unique id of the shift workedd.

Instance: “10003”

Attribute: shift_date

Type: date

Meaning: Holds the date the employee worked”

Instance: “05/10/2018”

Attribute: start_time

Type: time

Meaning: Holds the starting time of an employees shift.

Instance: “08:00:00”

Attribute: end_time

Type: time

Meaning: Holds the ending time of an employees shift

Instance: “11:00:00”

An employee works a **work shift** during a specific date and time within bakersfield flowershop.

Description of Relationship Types

Relationship name: **Assigned**

- Meaning: An employee is **assigned** to make a delivery.
- Related Entity Types: Employee, Delivery
- Cardinality: 1..M; Participation: Partial, Total
- Example of relationship: Employee bob is assigned to take a delivery for 3:00 PM.

Relationship name: **Classifies**

- Meaning: Payment type **classifies** how the customer has provided their payment.
- Related Entity Types: Payment Type, Payment
- Cardinality: 1..M; Participation: Total, Total
- Example of relationship: Customer John made a payment using his credit card.

Relationship name: **Contains**

- Meaning: Product order **contains** a flower product.
- Related Entity Types: Product Order, Flower Product
- Cardinality: M..M; Participation: Total, Partial
- Example of relationship: Customer Jamie has made an order of two bouquets one containing only roses and the other daffodils.

Relationship name: **Has**

- Meaning: A product orders **has** an order status.
- Related Entity Types: Product Order, Order Status
- Cardinality: 1..1; Participation: Total, Total
- Example of relationship: The order of Crimson Passion bouquet has an order status which tells the customer that it is currently still in the process of making.

Relationship name: **Makes**

- Meaning: Customer **makes** a product order.
- Related Entity Types: Customers, Product Order
- Cardinality: 1..M; Participation: Partial, Total
- Example of relationship: Customer Ann makes an order of two Red Rose bouquets.

Relationship name: **Needs**

- Meaning: A Supply Purchase Order **needs** Payment.
- Related Entity Types: Supply Purchase Order, Payment
- Cardinality: M..M; Participation: Total, Partial
- Example of relationship: We pay for Supply Purchase Order 144433 with $500.

Relationship name: **Places**

- Meaning: An Employee **places** a Supply Purchase Order.
- Related Entity Types: Employee, Supply Purchase Order
- Cardinality: 1..M; Participation: Partial, Total
- Example of relationship: Bakersfield Flowershop is low on Lilies so the employee Francis creates a supply purchase order to refill them.

Relationship name: **Processes**

- Meaning: An employee **processes** product order.
- Related Entity Types: Employee, Product Order
- Cardinality: 1..M; Participation: Partial, Total
- Example of relationship: Employee Chris has processed a total of 5 orders this morning.

Relationship name: **Refills**

- Meaning: A Supply Purchase Order **refills** flower products.
- Related Entity Types: Supply Purchase Order, Flower Products
- Cardinality: M..M; Participation: Total, Partial
- Example of relationship: The Supply Purchase Order refills our stock of red roses.

Relationship name: **Requires**

- Meaning: Product order **requires** a payment from the customer.
- Related Entity Types: Product Order, Payment

- Cardinality: M..M; Participation: Total, Partial
- Example of relationship: The order of one Crimson Passion bouquet still requires a payment before the florist may proceed to arrange it.

Relationship name: **Satisfies**

- Meaning: A Supplier **satisfies** a supply order.
- Related Entity Types: Supplier, Supply Purchase Order
- Cardinality: M..M; Participation: Partial, Total
- Example of relationship: Golden Valley Rose Distributors satisfies an order for white roses.

Relationship name: **Shipped To**

- Meaning: A product order is **Shipped to** an address in the database.
- Related Entity Types: Product order, Delivery Address
- Cardinality: M..1; Participation: Total, Partial
- Example of relationship: Product order 12354453 is shipped to 1234 Elmo Street, Bakerfield, CA, 93309.

Relationship name: **Packed**

- Meaning: A product order is **Packed for** a package.
- Related Entity Types: Product Order, Package
- Cardinality: 1..M; Participation: Partial, Total
- Example of relationship: An order of Red Roses is packaged and scheduled to be delivered at 2:00 PM.

Relationship name: **Sent To**

- Meaning: A delivery is **sent to** a recipient.
- Related Entity Types: Delivery, Recipient
- Cardinality: M..1; Participation: Total, Total

- Example of relationship: A delivery is sent to Stacy Diamonds from her husband Chad Diamonds.

Relationship name: **Tracked By**

- Meaning: The times an employee has worked for Bakersfield Flowershop is **tracked by** their Work History.
- Related Entity Types: Employee, Work History
- Cardinality: 1..M; Participation: Total, Total
- Example of relationship: Employee George started working for Bakersfield Flowershop on 01/03/2015 and quit 05/04/2017.

Relationship name: **Works**

- Meaning: An employee **works** a shift in Bakersfield Flowershop
- Related Entity Types: Employee, Work Shift
- Cardinality: 1..M; Participation: Total, Partial
- Example of relationship: The employee stacy **works** on July 14th as a cashier in Bakersfield Flowershop.

1.1.5 User Groups, Data Views and Operations

The database for Bakersfield Flowershop will have three user groups. One will be for customers, one is for the manager of the database, and one is for employees. Separating what is accessible to each user ensures the information in our database is secure, protected, and maintained.

The customers of Bakersfield Flower Shops interactions with the database will involve their account information, viewing past orders, and creating new orders. Customers will not have any direct interaction with the database, but only be able to access it through what our website allows them to. Employees of our database will be able to view customer orders, make orders with suppliers, and add new products as they are made.

They can also assist customers with account creation in store, but customers will be unable to input sensitive information until they access our website and change their login credentials. The manager will be able to insert information related to making new employees, their histories, and the shifts employees work into the database.

1.2 Conceptual Database Design

Before creating a database, you must first figure out how your data will be stored. An Entity-Relationship (ER) model will be used to represent the data we have collected and the relationships they have.We can represent our data with two properties, an entity which will represent an object such as our flower_product or employee and a relationship which will show how different entities relate to each other.

Section 1.2 contains detailed information about the entity sets and relationships that are part of our database scheme. For every entity, information is given for it primary key, the type of entity is it and a table explaining all of its attributes. Similarly, for every relationship, a description of the entity is given that explains its purpose, what entities that it relates, and the multiplicities of the relationship. The section concludes with our database E.R. model.

1.2.1 Entity Type Description

In a database an entity is a collection of data meant to model an object in our system. Each entity has a descriptive name meant to generalize the kind of object it is and is described by the attributes found in it’s fields. For example: one such entity in our database is the Employee Entity. Each employee has a name, address, and phone number that describe who this employee is. Due to possible collisions in data we also have a generated id associated with each of the employees to uniquely identify them.

This section will go into detail of each of the entities in our conceptual database and their respective attributes. We will go over it’s chosen name, whether it is a strong or weak entity type, the Primary key of the entity to identify each instance, and a description of the entity and what it represents. Each entity also has a table of attributes and descriptive traits for each.

Entity Name: Customers

Entity Type: Strong

Primary Key: Customer ID

Description: The purpose of the customer entity is to store information about the customers who purchase flowers from the website. This entity will contain common information such as the customer’s name, email address, street address, phone number, as well as a username, password, and date the account was created.

The customer entity will have frequent insertions of new tuples because new customers could make a purchase at any time. Updates will be somewhat frequent as the customer can change addresses, phone numbers, passwords, and sometimes email addresses. Deletion of tuples will be very infrequent and only occur after an error has occurred.

Attributes:

Attribute Name	**customer_id**	**name**	**address**	**username**
Description	Used to uniquely identify the customers of the shop.	Name of a customer (First, Middle, Last).	Customer’s street address, city, state, and zip code.	The username of a customer's account.
Domain/Type	Integer	Varchar, Varchar, Varchar	Varchar, Varchar, Varchar, Integer	Varchar
Value/Range	All positive n-digit numbers	Any, Any, Any	Any, Any, Any, 00000-99999	Any
Default Value	None	None	None	None
Null Value Allowed	No	No	No	No

Unique	Yes	No	No	Yes
Single or Multi-value	Single	Single	Single	Multi
Simple or Composite	Simple	Composite	Composite	Single

Customer Continued…

Attribute Name	**password**	**email**	**acc_creation_ date**	**phone_number**
Description	This will contain a hash of the users password.	This holds the email of the user and will be used to update the customer on the status of their order.	This will hold the date the account was created.	This will hold the phone number of the customer in case they need to be contacted in regard to their order.
Domain/Type	char[64]	Varchar	Date	Varchar
Value/Range	[a-z][0-9] valid in size of char	All valid email addresses	All dates	All valid phone numbers
Default Value	None	None	01/01/1970	(000)000-0000
Null Value Allowed	No	No	No	No
Unique	Yes	Yes	No	No
Single or Multi-value	Single	Single	Single	Multi
Simple or Composite	Simple	Simple	Simple	Simple

Entity Name: Package

Entity Type: Strong

Primary Key: package_id

Description: The delivery entity represents each delivery made by our company to the recipient. This entity has three attributes, package id, expected delivery time, and message. Each id will be used for each package being delivered, expected delivery time will hold when the package is expected to be delivered to the recipient, and the message attribute will hold the message that will accompany the flower order.

Insertions will be very frequent as many customers prefer to have the flowers delivered. Updates will be semi-frequent as the expected delivery time may change or the customer may request to change their message. Deletions will be very rare.

Attributes:

Attribute Name	**package_id**	**expected_delivery_time**	**message**
Description	Unique ID for each package delivered by employees to recipients.	This will display when the customer can expect the delivery to arrive.	This will hold a message related to the delivery
Domain/Type	All positive n-digit numbers	datetime	varchar
Value/Range	All	All valid datetimes	Any
Default Value	00000000000	01/01/1970 00:00:00	None
Null Value Allowed	No	No	Yes
Unique	Yes	No	No
Single or Multi-value	Single	Multi	Single
Simple or Composite	Simple	Simple	Simple

Entity Name: Delivery Address

Entity Type: Strong

Primary Key: address_id

Description: The Delivery address entity represents addresses Bakersfield Flower Shop will deliver too. It will store as a composite attribute the addresses input from customers and employees where product orders will be delivered.

This entity will keep all addresses input from our front end and instances of the entity will only be removed under special circumstances, like a customer requesting an address be removed from the database. The entity will keep a record of all addresses input.

Attributes:

Attribute Name	address_id	address
Description	This is an id to uniquely identify addresses.	The street address, city, state, and zip code where product orders will be delivered to.
Domain/Type	All positive n-digit numbers	Varchar, Varchar, Varchar, Integer
Value/Range	All	Any, Any, Any, 00000-99999
Default Value	00000000000	None
Null Value Allowed	No	No
Unique	Yes	No
Single or Multi-value	Single	Multi
Simple or Composite	Simple	Composite

Entity Name: Employee

Entity Type: Strong

Primary Key: employee_id

Description: The employee entity represents each unique employee that is currently employed by the company. This entity stores information about the individuals who work for the company. This entity will include some basic information such as the employee's name, address, and phone number and the employee's id number.

The employee entity will have frequent insertions due to hiring new employees and having to input their data into the database.Updates will be frequent because an employee may change addresses or phone numbers during their employment. Deletions may also be somewhat frequent because when an employee quits or is fired, the company may no longer need their information.

Attributes:

Attribute Name	employee_id	name	address	phone_number
Description	This is an id number used to	This contains the employee's	The current address of the	The current phone number of

	distinguish employees from each other.	full legal name. (First, Middle, Last)	employee which includes street address, city, state, and zip code.	the employee.
Domain/Type	Integer	varchar, varchar, varchar	varchar, varchar, varchar, Integer	varchar
Value/Range	All positive n-digit numbers	All names	Any, Any, Any, 00000-99999	All valid phone numbers
Default Value	00000	None	None	“(000)000-0000”
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	Composite	Simple

Entity Name: Flower Product

Entity Type: Strong

Primary Key: product_id

Description: This entity represents each flower product that can be added to the order. This is an important entity in our diagram as it is the only product being sold through our website. It will hold information on the type of flower, the flower name, current selling price, and the price it was purchased at from suppliers.

Attributes:

Attribute Name	**product_id**	**product_name**	**sell_price**	**purchase_price**
Description	An id that identifies each of the products sold by the shop.	The name of the product/flower	Current selling price of the product to customer	Price purchased at from the supplier.
Domain/Type	Integer	Varchar	decimal	decimal
Value/Range	All n-digit numbers	All flower names	All positive values	All positive values

Default Value	0000000000	None	0.00	0.00
Null Value Allowed	No	No	No	No
Unique	Yes	Yes	No	No
Single or Multi-value	Single	Single	Multi	Multi
Simple or Composite	Simple	Simple	Simple	Simple

Flower Product Continued…

Attribute Name	**color**	**length**	**product_image**	**description**
Description	The color of the specific flower product.	The length of the flower product.	An image of the product.	A brief description of the flower product
Domain/Type	Varchar	Varchar	Varchar	Varchar
Value/Range	All valid colors	All positive lengths	All valid image files	Any
Default Value	White	None	None	Flower
Null Value Allowed	No	No	Yes	No
Unique	No	No	No	No
Single or Multi-value	Single	Single	Single	Single
Simple or Composite	Simple	Simple	Simple	Simple

Entity Name: Incoming Payment

Entity Type: Strong

Primary Key: incoming_id

Description: This entity will contain a sell id that is unique to each sale and the current sales tax for the city of Bakersfield. This is a child entity resulting from the disjunction on the Payment Entity. This table tracks transactions involving sales between the store and customers.

Attributes:

Attribute Name	incoming_id	sales_tax
Description	Id associated with transactions that occur from sales within the store or on our website.	The percentage of sales tax charged in Bakersfield.
Domain/Type	Integer	Float
Value/Range	All positive n-digit numbers	0.0-100.0
Default Value	000000000000	0.0
Null Value Allowed	No	No
Unique	Yes	No
Single or Multi-value	Single	Multi
Simple or Composite	Simple	Simple

Entity Name: Order Status

Entity Type: Strong

Primary Key: status_id

Description: The purpose of the status entity is to indicate how far along an order is. There will be a brief description about the status and the date.

Attributes:

Attribute Name	status_id	description
Description	A unique id to each description of status for orders.	This will store the current status of the order.
Domain/Type	Integer	Varchar
Value/Range	All n-digit numbers	Any
Default Value	000000	None
Null Value	No	No

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

Entity Name: Payment

Entity Type: Strong

Primary Key: payment_id

Description: The purpose of this entity is to hold information about base payment used for an order. The information held in this entity is basic information about the payment such as the date and the amount paid.

For this entity there will be frequent insertions because it will track all payments from every customer. Updates will not occur unless a customer decides to add on to their order. Deletions will only occur if a customer cancels their order.

Attributes:

Attribute Name	**payment_id**	**payment_time**	**amount**
Description	A unique id associated with each transaction that occurs both in store and on the website.	The date and time a payment occurs.	The total amount paid for the transaction.
Domain/Type	Integer	Datetime	decimal
Value/Range	All positive n-digit numbers	All valid datetimes	All positive values
Default Value	None	01/01/1970 00:00:00	None
Null Value Allowed	No	No	No
Unique	Yes	No	No
Single or Multi-value	Single	Single	Single
Simple or	Simple	Simple	Simple

Composite			

Entity Name: Payment Type

Entity Type: Strong

Primary Key: payment_type_id

Description: The payment type entity is used to track the different forms of payment used by customers. It will contain a payment type id and a description of the payment used.

Payment type id will be a unique id for every possible form of payment, such as cash, check, or credit card. The description attribute will store the actual name of the payment as it relates to the id number.

Attributes:

Attribute Name	**payment_type_id**	**description**
Description	Unique Id for each of the possible forms of payment transaction could be.	Name of the type of payment for a transaction.
Domain/Type	Integer	Varchar
Value/Range	All positive n-digit numbers	Any
Default Value	00000000	None
Null Value Allowed	No	No
Unique	No	Yes
Single or Multi-value	Single	Single
Simple or Composite	Simple	Simple

Entity Name: Product Order

Entity Type: Strong

Primary Key: p_order_number

Description: The purpose of this entity is to keep track of each purchase made through the website and storefront. It contains two attributes, an order number and a datetime for when the order was placed.

There will be frequent insertions as new orders will be constantly coming in, but an order will only be deleted if an order is cancelled by the customer. There may be some updates to the order, such as when a customer wants to change the type of product being added or removed from the order.

Attributes:

Attribute Name	**p_order_number**	**order_time**
Description	An order number associated with the product order.	The date and time the order was placed.
Domain/Type	Integer	Datetime
Value/Range	All positive n-digit numbers	All valid datetimes
Default Value	00000000	01/01/1970 00:00:00
Null Value Allowed	No	No
Unique	Yes	No
Single or Multi-value	Single	Single
Simple or Composite	Simple	Simple

Entity Name: Recipient

Entity Type: Strong

Primary Key: recipient_id

Description: The purpose of this entity is to keep track of to whom the order is going to. It contains an id for the recipient, as well as their name, address, and phone number.

There will be frequent insertions as customers may want to have the product delivered to a specific person throughout the year. Deletions will be very infrequent and updates will be somewhat frequent as a recipient can change addresses or phone numbers.

Attributes:

Attribute Name	**recipient_id**	**name**	**address**	**phone_number**
Description	An id number that is unique to each recipient.	The recipient's name.	The recipient's address.	The recipient's phone number.
Domain/Type	integer	Varchar	Varchar	Varchar
Value/Range	All positive n-digit numbers	Any	All valid addresses	All valid phone numbers
Default Value	000000000	None	None	"(000)000-0000"
Null Value Allowed	No	No	No	No
Unique	Yes	No	No	No
Single or Multi-value	Single	SIngle	Multi	Multi
Simple or Composite	Simple	Composite	Composite	Simple

Entity Name: Supplier

Entity Type: Strong

Primary Key: supplier_id

Description: The purpose of this entity is to record information about the companies supplying flower products to our company. These suppliers will typically be purchased from flower farms. This entity will hold the suppliers id, name, address, and phone number.

Insertions will be infrequent since adding a new supplier will only happen when we purchase new flowers from a new farm. Updates will be infrequent as well since farms may rarely change locations. A deletion would not occur very often if at all.

Attributes:

Attribute Name	**supplier_id**	**vendor_name**	**address**	**phone_number**
Description	A unique id for each supplier.	The name of the supplier.	The supplier's current address.	The supplier's current phone

		Name identifies company, not person's name.		number.
Domain/Type	integer	varchar	Varchar, varchar, varchar, Integer	varchar
Value/Range	All positive n-digit numbers	Any	Any, Any, Any, 00000-99999	All valid phone numbers
Default Value	000000	None	None	(000)000-0000
Null Value Allowed	No	No	No	No
Unique	Yes	No	No	No
Single or Multi-value	Single	Single	Multi	Multi
Simple or Composite	Simple	Simple	Composite	Simple

Entity Name: Outgoing Payment

Entity Type: Strong

Primary Key: buy_id

Description: The purpose of the outgoing payment entity is to record all the payment made to the suppliers. This entity will have two attributes, buy id and supplier invoice id.

Insertions will be frequent as our flower shop will constantly be ordering flowers to keep inventory fresh. Deletions will only occur when a payment has been cancelled and updates will be non-existent.

Attributes:

Attribute Name	**outgoing_id**	**supplier_invoice_id**
Description	A unique identifier for transactions that occur between Bakersfield Flowershop and suppliers.	The suppliers invoice id we apply payment to.
Domain/Type	Integer	Integer
Value/Range	All positive n-digit numbers	All positive n-digit numbers
Default Value	0000000000	None

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

Entity Name: Supply Purchase Order

Entity Type: Strong

Primary Key: supply_id_purchase

Description: The purpose of this entity is to keep track of all purchases made from the suppliers. This entity will contain the supply purchase id and the date and time the purchase was made.

This entity will be frequently updated as purchases from the supplier are made. Deletions will only happen when an order is cancelled by us. Updates will not happen.

Attributes:

Attribute Name	**supply_purchase_id**	**supply_purchase_time**
Description	A unique id for each purchase from the supplier.	The date and time the Supply Purchase Order was placed.
Domain/Type	Integer	Datetime
Value/Range	All positive n-digit numbers	All valid date times
Default Value	00000000	01/01/1970 00:00:00
Null Value Allowed	No	No
Unique	Yes	No
Single or Multi-value	Single	Single
Simple or Composite	simple	simple

Entity Name: Work History

Entity Type: Strong

Primary Key: history_id

Description: This entity will hold the work history of each individual employee hired by our company. It will contain a unique id, the start_date and if the employee has quit, an end date. It will also contain that employee's job title and salary.

With the employee entity, we can track when a specific employee has started working and for how long they have been with the company. The job title will store what the worker's current position within the company is and the pay rate will be the employee's current salary.

Attributes:

Attribute Name	**history_id**	**start_date**	**end_date**	**job_title**	**pay_rate**
Description	Unique identifier of employee work history.	This will track when an employee began working for our flower shop.	This will store when an employee has stopped working for us.	This stores the various job titles for all employees.	This will store the current pay rate of each employee.
Domain/Type	Integer	Datetime	Datetime	varchar	decimal
Value/Range	All positive n-digit numbers	All valid dates	All valid dates	Any	All positive values
Default Value	00000	01/01/1970 00:00:00	Null	Cashier	13.00
Null Value Allowed	No	No	Yes	No	No
Unique	Yes	No	No	No	No
Single or Multi-value	Single	Single	Single	Single	Single
Simple or Composite	Simple	Simple	Simple	Simple	Simple

Entity Name: Work Shift

Entity Type: Strong

Primary Key: history_id

Description: This entity will hold the day that an employee will work in Bakersfield Flower Shop. It will contain a shift_id to identify the shift uniquely. There will be a column to identify the date they will work in the store, a column that identifies the starting time of their shift, and a column identifying the ending time of their shift.

In future phases of our document we will go over how this will reference the employee entity and allow our store to assign employees to shifts easily. With this table it will be used to simplify scheduling for a store.

Attributes:

Attribute Name	**Shift_ID**	**shift_date**	**begin_time**	**End_time**
Description	Unique identifier of the shift an en employee is working.	Day Employee works	Starting time of the shift	Ending time of the shift
Domain/Type	Integer	date	time	time
Value/Range	All positive n-digit numbers	All valid dates	All valid times	All valid times after begin_time
Default Value	00000	01/01/1970 00:00:00	Null	Null
Null Value Allowed	No	No	No	No
Unique	Yes	No	No	No
Single or Multi-value	Single	Single	Single	Single
Simple or Composite	Simple	Simple	Simple	Simple

1.2.2 Relationship Type Description

Relationships illustrate how entities are related with one another. Relationships help describe how two different entities will interact with each other and why they may be important to another entity in the database. Relationships may have an attribute that helps describe how the entities are connected.

In this section, we detail each relationship type in our conceptual database by listing the following: a description of the relationship types' purpose and the entities involved, mapping cardinality, descriptive field, and participation constraints

Relationship: Assigned
Description: An employee will be assigned to make deliveries. Employees can also take numerous orders, so they can make more than one delivery before returning back to the flower shop.
Entity Sets Involved: Employee, Delivery
Mapping Cardinality: 1..M
Descriptive Field: None
Participation Constraint: Partial participation for Employee. Total participation for delivery. Each delivery must be made by an employee, but there are only some employees designated to make these deliveries.

Relationship: Classifies
Description: A customer can purchase flowers with a debit/credit card, cash or check. Payment type will classify how the customer has provided their payment. Payments are also done by employers when doing a supply purchase order.
Entity Sets Involved: Payment Type, Payment, Product Payment, Supply Payment
Mapping Cardinality: 1..M
Descriptive Field: None
Participation Constraint: Total participation for both Payment Type and Payment. Payment type is recorded only when a payment has been made. For every payment made, the type of payment that is used should be identified.

Relationship: Contains

Description: As a flower shop, every one of our product orders will contain a flower product. A customer can choose a variety of flower products to be made into a flower arrangement of their choice. The price at which those flower products were sold will allow for customers to find other flower products at equivalent price in case something goes wrong with the flowers they initially purchased. In the case of refunds, no more or less of the amount used to purchase will be refunded.

Entity Sets Involved: Product Order, Flower Product

Mapping Cardinality: M..M

Descriptive Field: quantity_item, point_of_sale_price

Participation Constraint: Total participation for Product Order. Partial participation for Flower Product. There will be flower products that have yet to be ordered. Meanwhile, every order will have a flower product.

Relationship: Has

Description: Product orders all have an order status. Order status will let customers know when their payment has been received and when their order is in process. In general, it will let customers know at what stage their orders are in up until it is ready to be delivered or to be picked up by the customer. An order status may be updated for more than one product order if it is made by the same customer.

Entity Sets Involved: Order Status, Product Order

Mapping Cardinality: 1..M

Descriptive Field: time_updated

Participation Constraint: Total participation for both Order Status and Product Order. Every order status must be associated with a product order that has been made. If there is no order status, then there is no product order. Every product order must have an order status.

Relationship: Makes

Description: Customers will have the option to make an order either through our website or in-store, and they can also make multiple orders. No matter what method a customer chooses to make an order, all orders are associated with the same customer based on the personal information they have provided.

Entity Sets Involved: Customers, Product Order

Mapping Cardinality: 1..M

Descriptive Field: None

Participation Constraint: Partial participation for Customers. Total participation for Product Order. There may exist customers who have not made an order. Every product order, however, must be linked to a customer.

Relationship: Needs

Description: A supply purchase order needs payment. The payment can only be done by an employee. Multiple payments can be made for more than one supply purchase order.

Entity Sets Involved: Supply Purchase Order, Payment

Mapping Cardinality: M..M

Descriptive Field: supply_purchase_id, date

Participation Constraint: Total participation for Supply Purchase Order. Partial participation for Payment. To purchase more supply of flower products, a payment is always needed. Payment may not be for a supply purchase order, but for a product order done by a customer.

Relationship: Places

Description: An employee will place a supply purchase order. Multiple flower products can be close to or completely out of stock. One employee can make more than one supply purchase order as suppliers only carry a certain number of flower types.

Entity Sets Involved: Employee, Supply Purchase Order

Mapping Cardinality: 1..M

Descriptive Field: None

Participation Constraint: Partial participation for employees. Total participation for Supply Purchase Order. Not every employee in our flower shop has made a supply purchase order. All supply purchase orders, on the other hand, need to be placed by an employee.

Relationship: Processes

Description: A single employee can process more than one product order. There can be more than one product order being processed by a single employee but no two or more employees can process the same product order.

Entity Sets Involved: Employee, Product Order

Mapping Cardinality: 1..M

Descriptive Field: None

Participation Constraint: Partial participation for Employee. Total participation for Product Order. Not all employees are assigned to process a product order, but every product ordered must be processed by an employee.

Relationship: Refills

Description: A supply purchase order refills the flower products. Since the condition and availability of certain flowers depends on the weather, multiple supply purchase orders are made because a supplier may not have all the flower products that are needed to restock.

Entity Sets Involved: Supply Purchase Order, Flower Product

Mapping Cardinality: M..M

Descriptive Field: quantity_item, supply_price

Participation Constraint: Total participation for Supply Purchase Order. Partial participation for Flower Product. When a supply purchase order is made for a flower shop, it must be refilling a flower product. A flower product can exist without having yet been refilled.

Relationship: Requires

Description: To proceed with an order, a payment is required from the customer. In addition, a customer may choose to pay part of their order with a card and then the rest with cash. Multiple payments can be used to pay for one or more product orders.

Entity Sets Involved: Product Order, Payment

Mapping Cardinality: M..M

Descriptive Field: None

Participation Constraint: Total participation for Product Order. Partial participation for Payment. For all product orders, a payment is required. Not all payments made will be for a product order but could for a supply purchase order.

Relationship: Satisfies

Description: When a supply purchase order is made, a supplier will be the ones to satisfy that purchase. One supplier could satisfy more than one supply purchase order.

Entity Sets Involved: Supplier, Supply Purchase Order

Mapping Cardinality: M..M

Descriptive Field: None

Participation Constraint: Partial participation for Supplier. Total participation for Supply Purchase Order. Every supply purchase order made must be linked to a supplier. A supplier may exist but have not yet satisfied a supply purchase order.

Relationship: Packed

Description: The product order schedules a delivery. A customer may have flowers delivered at the time and date of their choosing to more than one recipient. This may all be done within the same order. Also, there could be a failed attempt at delivery, so a new delivery is set for the same product order.

Entity Sets Involved: Product Order, Package

Mapping Cardinality: 1..M

Descriptive Field: None

Participation Constraint: Partial participation for Product Order. Total participation for Packed. Customers may choose to pick up their orders, so not all product orders are needed to be delivered. A Package, on the other hand, must be associated with a product order.

Relationship: Sent To

Description: A delivery is sent to a recipient. Multiple deliveries can be made to a single recipient. Separate individuals may have flowers or other flower arrangements sent to the same individual, thus multiple deliveries will be made to a single recipient . We can verify it is the same recipient by their personal information.

Entity Sets Involved: Delivery, Recipient

Mapping Cardinality: M..1

Descriptive Field: None

Participation Constraint: Total participation for both Delivery and Recipient. Every delivery requires a recipient. There is no need for a delivery to be made if there is no intended recipient. Each recipient is expected to have a delivery be made to them. A recipient is not recorded if the customer has chosen to pick up the order themselves.

Relationship: Shipped To

Description: Product orders in our database are shipped to an address saved in our database. Our front end will collect this data from customers who have accounts online, and employees will also collect this information from customers if they make an in store order that is going to be scheduled for delivery. This will allow Bakersfield Flowershop to save frequent delivery addresses and where many product orders go to.

Entity Sets Involved: Product Order, Delivery Address

Mapping Cardinality: M..1

Participation Constraint: Total participation for Delivery addresses and partial participation for product order. Every Delivery address will have a product order associated with it, but not every product order has a delivery address. Some product orders are in store orders so a delivery address is not needed.

Relationship: Tracked By

Description: Every employee is tracked by a work history. In case an employee quits and/or returns, their work history would already be stored. This may help them in obtaining old or new positions at the flower shop or to simply confirm their work experience. It will also let the flower shop keep track as to who is doing what. If something happens, we may turn to the right employee to ask the questions.

Entity Sets Involved: Employee, Work History

Mapping Cardinality: 1..M

Descriptive Field: None

Participation Constraint: Total participation for both Employee and Work History. Every employee will have a work history, as well as every work history must be associated with an employee. A work history doesn't exist without being linked to an employee.

Relationship: Works

Description: Describes the relationship between an employee and the shift that they are working. Each employee is able to work a work_shift within the store. This relationship helps for when a manager would like to schedule employees and they don't have to manually enter an employee each time, as the future design of the front end will show them the currently working employees.

Entity Sets Involved: Employee, Work Shift

Mapping Cardinality: 1..M

Descriptive Field: None

Participation Constraint: Total participation for work_shift but partial participation for employees. In general it will appear as if employees are total, but there are situations in which an employee will never appear in the relationship. If an employee is added to the store but they quit before being given a work shift they will never appear in work_shift.

1.2.3 Related Entity Types

Specialization is to essentially take an already existing entity and then create another entity from the one you have that has all the same attributes, in addition to its own set of attributes. In our own conceptual database model we have an Entity for payments, and we have two entities extending this for product payments and supply payments.

Performing generalization on entities is to do the opposite. Generalization is when you have multiple entities that have some of the same attributes, so you take the similar attributes and you create an entity to hold these same attributes. The entities before combination will still have their unique attributes, but they will inherit the attributes they shared from this new generalized entity. We considered making multiple entities for the different types of products that will be sold by Bakersfield Flowershop, like a Rose entity, Daisy entity, and Bouquet entity, then generalizing them to inherit from the Flower Product table. During the course of our design we decided it simplified our design to have fields for length, color, and name of flower within the products table instead of splitting them up in this way.

Specializations and generalizations can also be discussed in terms of a “IS-A” relationship. In our database an instance of a Product Payment “IS-A” instance of a Payment, but payments for products have sales tax so it required a separation. Not every transaction that occurs in Bakersfield Flowershop has sales tax, only the sale of products to our customers.

There are two kinds of constraints on specialization and generalizations, which are participation and disjointedness. The disjointedness constraint specifies entities that share attributes with their parent entity, aside from a unique attribute that is not shared with the parent entity. Participation is a constraint that specifies if an entity must be a child entity. The possible values for participation are total and partial, where total means that the entity must also be one of the children entities, and partial means that it may also be a child entity but it does not have too.

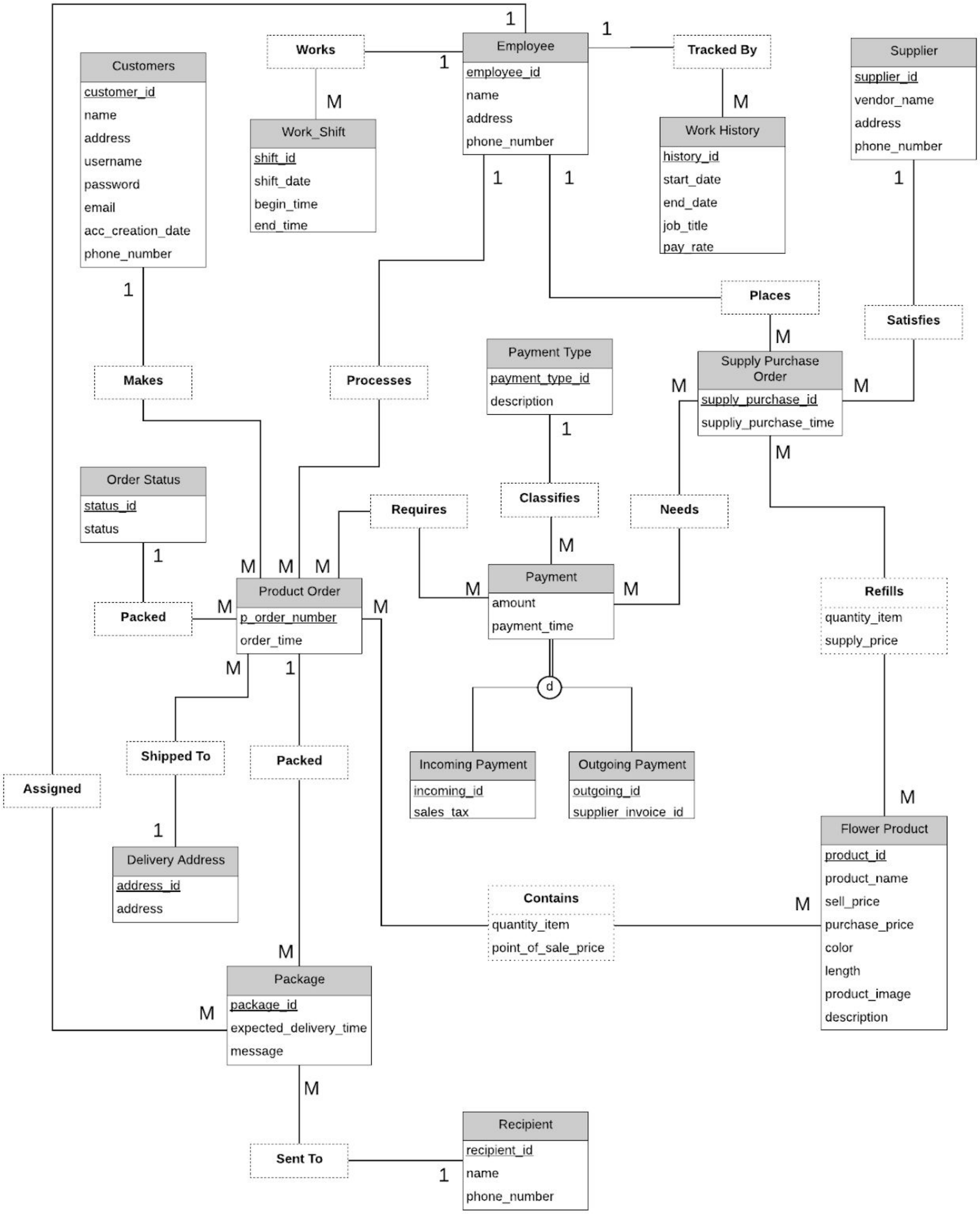
Aggregation is an abstraction concept where you build an object from component objects. This can be used to describe attributes combining to define an entity, a relationship between two entities that are tightly coupled, or also describe how multiple entities interact when they work together to describe one thing. This can also be described as a “HAS-A” relationship. In our design we have entities Order and Order Status, and payment. An order HAS-A order status, and an order HAS-A payment. This all describes an order at Bakersfield Flowershop being completed, but each entity is responsible for describing one part of the process.

1.2.4 E-R Diagram

An E-R Diagram is used to visually represent entities and the relationships between them. Entities are represented by the boxes with light blue headers. The relationship is denoted by the lines between entities.

The cardinality of a relationship is described by “1” and “M.” “1” next to an entity denotes the entity on the other end of the relationship is related to one instance of that entity. An “M” next to an entity denotes that there could be multiple instances of this entity related to the entity on the other side of the relationship. Each relationship is described by a combination of these two: 1:1 (one-to-one), 1:M (one-to-many), M:1 (many-to-one), M:M (many-to-many). Between each of these relationships is a description for the way these entities interact.

See next page for diagram.


Customers
customer_id
name
address
username
password
email
acc_creation_date
phone_number
Works
Work_Shift
shift_id
shift_date
begin_time
end_time
Employee
employee_id
name
address
phone_number
Tracked By
Work History
history_id
start_date
end_date
job_title
pay_rate
Supplier
supplier_id
vendor_name
address
phone_number
Places
Satisfies
Makes
Processes
Payment Type
payment_type_id
description
Supply Purchase Order
supply_purchase_id
suppliy_purchase_time
Order Status
status_id
status
Requires
Classifies
Needs
Refills
quantity_item
supply_price
Packed
Product Order
p_order_number
order_time
Payment
amount
payment_time
d
Incoming Payment
incoming_id
sales_tax
Outgoing Payment
outgoing_id
supplier_invoice_id
Shipped To
Packed
Assigned
Delivery Address
address_id
address
Contains
quantity_item
point_of_sale_price
Flower Product
product_id
product_name
sell_price
purchase_price
color
length
product_image
description
Package
package_id
expected_delivery_time
message
Sent To
Recipient
recipient_id
name
phone_number
1
1
1
M
M
1
1
1
1
M
M
M
1
M
M
1
M
M
M
M
M
M
M
M
M
1
1
M
M
M
M
M
1

Phase 2: Conceptual and Logical Database

Phase two focuses on the conversion of the Flower Shop database from the ER model to the Relational model. The relational model is another useful modeling tool for database design. Converting from the ER model to the Relational model helps ensure that a DBMS will be able to function with its data when the relational model is converted to an actual database.

Section 2.1 will give introductory information on these models, and will document the common techniques for the conversion of the ER model to the Relation model and presents our own database using relations. Section 2.2 will go over the conversion of entities and relations in our ER model to a Relational Model. Section 2.3 will focus on defining the relations in the Flower Shop with examples of those relations and their data. Section 2.4 will cover our database relations with ten sample queries.

2.1 E-R Model and Relational Model

An E-R model is used to show the relation between entities in our database in a conceptual way. A relational model represents our entities in tuple format and how the data within tables interact in a more direct way. [Probably need 1 more sentence]

Section 2.1.1 will go into detailed descriptions of E-R Model and Relational Models. Section 2.1.2 is going to discuss the similarities and differences between these two models. They are both tools to show the design of a database but each one is better at conveying different ideas in databases. Section 2.1.2 will go over the trade-offs of using one versus the other.

2.1.1 Description of E-R Model and Relational Model

Understanding the background of the models we use in database design is important to understanding why we use them. Prior to the ER model and Relational model there were many ways companies designed databases. The models they used may have made sense for their company only, but they did not have as strong of a foundation as the ER and relational models do.

This section will be about the background of the ER model and the Relational model. It will go over their history, what they are, their major features, and for what purposes they can be used for.

History

Dr. Peter Pin-Shan Chen first introduced the E-R model in 1976 in a paper titled “The Entity-Relationship Model: Toward a Unified View of Data” with the goal of defining a way to represent real world objects, ideas, and their relations in a natural way that could be translated to a database. His paper goes into details about how entity sets and value sets those entities contain, as well as defining the cardinality relations can take on; such as one-to-one, one-to-many, many-to-many, ect.

The relational model was invented by Edgar F. Codd in 1970 while he was working at IBM. Prior to Codd’s invention of the relational model databases did not have set standards for implementation. With his idea representing data in databases could be approached using principles of logic and mathematics. Codd’s model provided a way to design databases that could then be translated almost directly to any database management system.

What the Model Is

Chen’s E-R model idea is primarily concerned with the visual representation of entities and relations. It can be used to easily explain how data interacts to people who do not work on databases everyday. The appeal of his model is it’s simplicity and readability and makes designing a database accessible for non-technically inclined people.

The relational model is mainly for designing a database prior to implementing it into a database management system. The relational model can have the operations defined in relational mathematics done to them the same way a database management system can. It is a useful tool for those who plan to implement a database.

Major Features

The major features of the ER model are Entities, relations described by their cardinalities, and descriptions of the relation between entities. The ER model represents these visually to help describe a database in the planning phase of it’s design. It is not concerned with the allowed values of data types, as it leaves that to the relational model.

Major features of the relational model are relations. Relations are essentially a table of values, or a flat file of data. Using relations you can model a full database. A relation is made up of attributes and tuples, with attributes being constrained by a domain of values. Attributes describe data found within the relation and a tuple is a collection of attributes.

Purpose of the Models

The purpose of these models is to give ways to describe databases. An ER model helps create a bridge for a business owner to understand what their database needs, and a relational model is a way to model what the database will look like prior to implementing it into a database management service. ER model is simple and easy to understand,

and a relational model is descriptive enough to model a database, but generalized enough to be translated to any database management service.

2.1.2 Comparison of Two Different Models

Both the ER model and Relational model are useful, but they have different strengths. The ER model is used primarily for its ease to understand. It provides a visual medium for business people to see how the entities in the database relate to each other. It’s a useful design tool that can map a database quickly and be understood easily. It’s best to use as a first step in database design as the designer does not have to be concerned with implementation details and can focus on creating entities and mapping out how they relate. The disadvantage of the ER model is that it does not directly translate to a database as easily as the relational model. It is easily understood by humans, but it’s method of modeling does not translate perfectly to the strict logic of computers.

The relational model is also a useful tool for designing databases but has different strengths. It is more strict than the ER model in what it allows to be modeled using it, as it’s end goal is to be able to be translated to a DBMS. Modeling a database using the relational model constrains the designer into ensuring the database can be modeled using principles of logic and mathematics. Doing this allows query languages to perform operations on relations. The disadvantage of using the relational model is it is hard to understand for people not already familiar with the model. It is easily translated to any DBMS, but difficult for the general human to understand.

The models do have similarities. They can both express ideas of a database, but do so in different ways. The ER model has entities and relationships, while the relational model only has the relation. The relationships described by the ER model can be expressed in the relational model, it is just not as easy to understand. The ways you can convert the ER model’s relationships and entities will be discussed in the next few subsections. They are similar enough that one model can be converted to the other and vice versa.

2.2 From Conceptual Database to Logical Database

In this section, entity types and relation type from the E.R. model will be converted into relations for use in a logical database. Many different methods can be used to complete this process depending on the mappings.

This section will cover methods used to convert Entities to relations 2.2.1. It will also go over how to convert the different relationship types that exist in the ER model into relations in 2.2.2. The last section 2.2.3 will go over the different constraints that databases must follow.

2.2.1 Converting Entity Types to Relations

The ER model helped to set the groundwork of the business our database needs to represent and can easily be converted to the relational model. The ER model uses entities and relationships to model data and how the data interacts. The relational model uses only relations to represent a database.

Every entity from the ER model will be converted into a relation. Every attribute in an entity from the ER model will be an attribute in the corresponding relation in the relational model. Each relation will contain a primary key to uniquely identify it. This section will go over the ways Entities from the ER model can be converted to relations.

Strong Entity Type Conversion

To convert a strong entity type E from an ER model to relational model create a relation R that includes all the simple, single value, attributes from E. An attribute in R that can be used to uniquely identify each tuple should be designated as the primary key attribute for R. If the converted entity E had a multi-value attribute the attribute must be converted to multiple simple attributes in the relation.

Weak Entity Type Conversion

Weak entity conversion follows much of the same principles of the strong entity conversion. For each weak entity type W from the ER model with the owner entity type E create a relation R with all the simple attributes (including composite attributes broken down into simple attributes) of E. Choose an attribute in R that can uniquely identify each tuple and designate it as the partial key. The primary key of R is the combination of the primary key from the owner entity E and the partial key chosen.

Mapping of Simple and Composite Attributes

Mapping simple attributes from the ER model to the relational model involves designating an attribute field for each of the simple attributes in the ER model. Composite Attributes must be broken down into simple attributes and then they follow the same path as simple attributes and are converted to their attribute fields.

Mapping of Single and Multi-valued Attributes

Single value attributes directly translate to simple attributes of a relation in the relational model as described in the previous section. For multi-valued attributes there is a more involved method. For each multi-valued attribute, A, create a new relation R. R will have a unique attribute as part of it’s primary key to describe its relation to A. R will also contain the primary key of A as a foreign key. The unique attribute chosen in R and the foreign key from A will form the primary key of this relation.

2.2.2 Converting Relationship Types to Relations

In the ER model relationships can be described with a line connecting entities with the cardinality of the relationship expressed along the line. This is easy to understand, but the same cannot be done with the relational model. The relational model can describe relationships but it has to be done in relations already existing or by creating a relation to describe the relationship.

This section will go over how the relationships between entities described by an ER model can be converted to the relational model. Most conversions involve using a foreign key in another relation or creating a relation to describe the relationship. This section will compare the ways that work best for the relationship cardinality to be converted.

Mapping of Binary 1:1 (one-to-one) Relationships

For every binary 1:1 relationship type, referred to as R, in an ER diagram identify the relations A and B that correspond to the entity types participating in R. The three methods to convert are as follows:

1. **Foreign Key Approach**
   To achieve this you choose A or B, and make the primary key of A a foreign key in B. The best way to do this is to make the entity type with total participation in R the role of B.

2. **Merged Relation Approach**
   To achieve this you take the 1:1 relationship type and merge them into one relation. This can be done when both relations' participations in R are total, as they will always have the same number of tuples at all times.

3. **Cross-Reference or relationship relation approach**
   The final way to convert involves creating a third relation to cross reference A and B we will call C. This is also referred to as a relationship relation or a lookup table. To accomplish this C will contain the primary keys of A and B as foreign keys. The primary key of C may be one of the foreign keys from A or B.

The **Foreign Key Approach** is best used when one relation in the 1:1 relationship has total participation in the other relation. If one relation does not then the **cross-reference approach** should be used, as the relations will be able to reference each other without requiring total participation. The **merged relation approach** is best done in the design phase prior to mapping the ER model to the relational model, but is a good choice if at this point the designer is trying to decrease the number of relations in their model.

Mapping of 1:M (one-to-many) Relationships

For each regular binary 1:M relationship type in R, identify the relation A that represents the participating entity type at the M-side of the relationship type, and B will be the relation that represents the 1-side. The ways to convert these relationship types are as follows:

1. **Foreign Key Approach**

   To accomplish this conversion take A and include the primary key of B as a foreign key in A. The entity that is A or B matters here because the M-side of this conversion is related to at most one entity instance to the 1-side.

2. **Cross Reference Approach**

   Similar to the way described in binary relationship conversions create a relation C that includes the primary key of A and B as foreign keys. The primary key of C should be the foreign key obtained from including A.

The **foreign key approach** is best used when most of the tuples in the A participate in R. If few tuples from A participate in R then the **cross reference approach** is better as it avoids excessive NULL values.

Mapping of M:M (many-to-many) Relationships

This can only be accomplished using the **Cross Reference Approach** described in previous relationship conversions. For each M:M relationship R, create a relation to represent the relationship, which will call C. In C include the participating entities primary keys, and their combination will represent the primary key of C.

Mapping of Superclass and Subclass for “IsA” relationship

Entities that are disjointed from one parent entity only are described with the “IsA” relationship type. Essentially they are a more specialized version of the parent entity. We will refer to the superclass in these relationship types as C, and a subclass as S { $S_1, S_2, ..., S_m$ }. The methods for converting this relationship type from the ER model to the relational model are as follows:

1. **Multiple Relations - superclass and subclasses**

   To convert these types create a relation, L, for the superclass, C, and set the primary key of L as the primary key of C. For each subclass of C create a relation for each and set their primary keys to be the primary key of L.

2. **Multiple Relations - subclass only**

   This method only works for the disjointness constraint. Create relations only for the subclasses and not the superclass, but each of the relations in the subclass contain the attributes of the superclass. The primary key of each of the subclasses is the primary key from the superclass.

3. **Single relation with one type attribute**

   This method is achieved by creating a single relation L with the attributes in C, all the attributes from $S_1, S_2, ..., S_m$, and another attribute t denoting the **type** or **discriminating** attribute of the relation. The **type** attribute specifies which subclass the relation belongs to.

4. **Single relation with multiple type attributes**
   Not recommended for “IsA” relationship types but it is possible. The method to do these will be discussed in the “HasA” section where it is more useful for the relationship type.

The **multiple relations - superclass and subclasses** is the best option if the specializations involved are disjoint partial. If the specialization is disjoint total then it is better to use **multiple relations - subclass only**. The **single relation with one type attribute** is better than the previous two methods only when there are not many specific attributes defined for the subclass.

Mapping of Superclass and Subclass for “HasA” relationship

Relationships described by a “hasA” relationship type contain entities that can be described with and belong to multiple subclasses. The same as the previous section; we will refer to the superclass in these relationship types as C, and a subclass as S { $S_1, S_2, ..., S_m$ } and can convert them with the methods following:

1. **Multiple Relations - superclass and subclasses**
   Refer to the method used for “IsA” relationship type, as it is the same procedure.

2. **Single relation with multiple type attributes**
   Create a single relation L containing all the attributes of C, every attribute from $S_1, S_2, ..., S_m,$ and the attributes $t_1, t_2, ..., t_m.$ The attributes $t_1, t_2, ..., t_m$ are boolean types that denote whether or not the relation belongs to a subclass.

The **multiple relations - superclass and subclasses** mapping should be used when there are many specific attributes defined for a subclass. If there are not, the **single relation with multiple type attributes** method should be used instead.

Mapping Relationship types involving other Relationship Types

This conversion can be accomplished using what is most similar to the **foreign key approach** discussed in earlier sections. After converting the entities participating in this relationship type, the entity involved in all the relationship types will have a foreign key from each of the participating relations but will also contain its own attributes. You treat the relationships as an attribute that can be referenced by using the foreign keys from the relation the foreign key originated from.

Mapping Recursive Relationships

A recursive relationship in an ER diagram is an entity that has a relationship with itself. To convert these relationship types there are two options:

1. **Create a field for the foreign key referenced (foreign key approach)**
   This is achieved by creating an attribute in the relation set up to hold the foreign key of another tuple in the relation. An example would be a relation called "employee" with a "works for" attribute containing the primary key of another employee as a foreign key to denote that an employee reports to another employee.

2. **Cross Reference Approach**
   To do this conversion create a new relation L. L will be named to define the relation relationship. L will contain multiple foreign keys from a single table that denote their relationship. An example would be a "works for" relation with attributes of "supervisor" and "employee" where the relation exists only to define their connection.

The **foreign key approach** is sufficient if the primary key and foreign key only need to be defined once in the tuple, or if there is not a need to keep a record of previous primary and foreign key connections. If this approach is used the foreign key field is going to be overwritten. If this is not allowed the **cross reference approach** should be used instead.

Mapping of Relationships involving more than 2 entity types

To accomplish this conversion create a relation A to represent an entity created to connect all entity types. After converting all entity types within A to relations include their primary keys as foreign keys in A and also include attributes that are important to describing the relationship. The primary key of A will be the combination of foreign keys from all relations A is meant to connect except in cases where a relations cardinality constraint is 1.

Mapping of Category (or union) Types

A category, also called union, type relationship involves a subclass of the union of two or more superclasses. To map these relationship types from an ER model to relational model we have to specify a new attribute in the relation as a **surrogate key.** The surrogate key is a primary key of a relation that is created to uniquely identify a tuple and is not derived from any existing information. To make use of the surrogate key here the entities related to the category relationship type will have a foreign key attribute for the created relation. Tuples that are part of the same category will have the value from the same surrogate key, and if they belong to none of the values from the surrogate key the field will be set to NULL.

2.2.3 Database Constraints

A constraint in a database is a restriction used to make the database perform efficiently and provides a strong framework to ensure data's integrity is maintained. There are constraints implicitly and explicitly defined in the database design and in the applications that use the database.

Constraints are important to ensure queries can be performed efficiently on the data in a database. This section will discuss some of the constraints that the designer of a database must follow.

Entity Constraints

Also called the entity integrity constraint. This specifies that a primary key attribute of a tuple cannot have the value of NULL. A primary key is used to uniquely identify a tuple in the relational model, so if NULL values were allowed the tuples who have it as the value in their primary key's field would not be identifiable.

Primary Key and Unique Key Constraints

In the relational model each set of tuples in a relation must be distinct, meaning the combination of one tuple's attributes cannot be the same as another tuple within the same relation. The primary key constraint ensures that there is an attribute within a tuple that makes it uniquely identifiable. A primary key with a unique value for each tuple in a relation ensures that a tuple's combination of attributes is also unique, fulfilling this constraint.

Referential Constraints

The referential integrity constraint ensures that if a tuple in a relation refers to a tuple in another relation, the tuple being referenced must exist. In practice this means that a primary key in a tuple in relation A can exist as a foreign key in a tuple in relation B, with the foreign key of the tuple in relation B being the same value as the primary key from the tuple in relation A. Doing this allows the tuple in relation B to reference the tuple in relation A.

Check Constraints and Business Rules

A check constraint is a check on data being inserted or modified into a tuple to ensure that it fulfills a condition before being allowed to be inserted into a database. Business rules are additional constraints that must be implemented into application programs connecting to a database to make a business function. An example of one would be “a customer can only have an account if they have an email,” which would mean the tuple with the email field cannot be NULL in a database when the front end of an application takes the user through account creation.

2.3 Convert E-R Model to Relational Database

Section 2.3 focuses on defining the relations in the Flower Shop database and provides an example of a single state relation. Each relation table specifies each attribute and its domain and the primary key of each relation. After the table will be a list of each relation’s constraints.

2.3.1 Relation Schema for our Local Database

Contains

p_order_number	Integer
product_id	Integer
quantity_item	Integer
point_of_sale_price	decimal

Primary Key: p_order_number, product_id combination key

Primary Key Constraint: No two tuples can share the value of the combination of p_order_number and product_id together.

Entity Integrity Constraint: p_order_number and product_id cannot be null.

Not Null Constraint: The attributes quantity_item and point_of_sale_price cannot be null.

Customer

Customer_id	Integer

fName	Varchar(255)
lName	Varchar(255)
street	Varchar(255)
city	Varchar(255)
state	Varchar(255)
zip	Integer, 00000-99999
Username	Varchar(50)
Password	Varchar(255)
Email	Varchar(255)
Acc_creation_date	Datetime
Phone_number	Varchar(10)

Primary Key: “Customer_id”

Primary Key Constraint: No two tuples can have the same values for Customer_id.

Entity Integrity Constraint: The Customer_id attribute must not be null.

Not Null Constraint: The fName, lName, Username, Password, and Email attributes cannot be null.

Delivery Address

address_id	Integer
city	Varchar(255)
street	Varchar(255)
state	Varchar(255)
zip	Integer, 00000-99999

Primary Key: “address_id”

Primary Key Constraint: No two tuples can have the same values for address_id.

Entity Integrity Constraint: address_id must not be null.

Not Null Constraint: The attributes city, street, state, and zip cannot be null.

Package

Package_id	Integer
Expected_time	Datetime
Message	Text
p_order_num	Integer
employee_id	Integer

Primary Key: Package_id

Primary Key Constraint: No two tuples can have the same values for Package_id.

Entity Integrity Constraint: The Package_id attribute must not be null.

Not Null Constraint: The Expected_time attribute cannot be null.

Employee

Employee_id	Integer
fName	Varchar(255)
lName	Varchar(255)
Street	Varchar(255)
City	Varchar(255)
State	Varchar(255)
Zip	Integer, 00000-99999
Phone_number	Varchar(10)

Primary Key: Employee_id

Primary Key Constraint: No two tuples can have the same values for Employee_id.

Entity Integrity Constraint: The Employee_id attribute must not be null.

Not Null Constraint: None of the attributes should be null.

Flower Product

Product_id	Integer
Product_name	Varchar(255)
Sell_price	decimal
Purchase_price	decimal
Color	Varchar(50)
Length	Varchar(15)
Product_image	Varchar(255)
Description	Text
supply_purchase_id	Integer

Primary Key: Product_id

Primary Key Constraint: No two tuples can have the same values for Product_id.

Entity Integrity Constraint: The Product_id attribute must not be null.

Not Null Constraint: The Product_name, Sell_price, Purchase_price, Color, and Length attribute must not be null.

Incoming Payment

Incoming_id	Integer
Sales_tax	Float, 0.0-100.0

Primary Key: Incoming_id

Primary Key Constraint: No two tuples can have the same values for incoming_id.

Entity Integrity Constraint: The incoming_id cannot be null.

Not Null Constraint: The integer attribute cannot be null.

Needs

Supplier_purchase_id	Integer

payment_id	Integer

Primary Key: Supplier_purchase_id, payment_id combination

Primary Key Constraint: No two tuples can have the same values for Supplier_purchase_id with payment_id.

Entity Integrity Constraint: The Supplier_purchase_id and payment_id cannot be null.

Not Null Constraint: The Supplier_purchase_id and payment_id cannot be null.

Order Status

Status_id	Integer
Status	Varchar(255)

Primary Key: Status_id

Primary Key Constraint: No two tuples can have the same values for Status_id.

Entity Integrity Constraint: The Status_id must not be null.

Not Null Constraint: The Status attribute must not be null.

Payment

Payment_id	Integer
Payment_time	Datetime
Amount	decimal
payment_type_id	Integer

Primary Key: Payment_id

Primary Key Constraint: No two tuples can have the same values for Payment_id.

Entity Integrity Constraint: Payment_id cannot be null.

Not Null Constraint: The amount attribute cannot be null.

Payment Type

Payment_type_id	Integer
Description	Text

Primary Key: Description

Primary Key Constraint: No two tuples can have the same values for Description.

Entity Integrity Constraint: The Description cannot be null.

Not Null Constraint: Payment_type_id attribute cannot be null.

Product Order

P_order_number	Integer
Order_time	Datetime
customer_id	Integer
status_id	Integer
employee_id	Integer
address_id	Integer

Primary Key: P_order_number

Primary Key Constraint: No two tuples can have the same values for p_order_number.

Entity Integrity Constraint: The p_order_number cannot be null.

Not Null Constraint: Order_time attribute cannot be null.

Recipient

Recipient_id	Integer
fName	Varchar(255)
lName	Varchar(255)
Phone_number	Varchar(10)
package_id	Integer

Primary Key: Recipient_id

Primary Key Constraint: No two tuples can have the same values for recipient_id.

Entity Integrity Constraint: The recipient_id cannot be null.

Not Null Constraint: The fname, lname, city, street, state, zip, and phone_number attributes cannot be null.

Refills

supply_purchase_id	Integer
product_id	Integer
quantity_item	Integer
supply_price	decimal

Primary Key: Supply_purchase_id, product_id combination

Primary Key Constraint: No other combination of supply_purchase_id and product_id can have the same combined value as these tuples.

Entity Integrity Constraint: Neither Supply_purchase_id and product_id can be null.

Not Null Constraint: The attributes quantity_item and supply price cannot be null.

Requires

p_order_number	Integer
payment_id	Integer

Primary Key: p_order_number, payment_id

Primary Key Constraint: No other p_order_number and payment_id combination can have the same value as the combination of this tuple.

Entity Integrity Constraint: Neither p_order_number or payment_id can be null.

Not Null Constraint: Neither p_order_number or payment_id can be null.

Supplier

Supplier_id	Integer
Vendor_name	Varchar(255)
street	Varchar(255)

city	Varchar(255)
state	Varchar(255)
zip	Integer, 00000-99999
Phone_number	Varchar(10)

Primary Key: Supplier_id

Primary Key Constraint: No two tuples can have the same values for supplier_id.

Entity Integrity Constraint: The supplier_id cannot be null.

Not Null Constraint: The vendor_name, street, city, state, zip, phone_number attributes cannot be null.

Outgoing Payment

Outgoing_id	Integer
Supplier_invoice_id	Integer

Primary Key: Outgoing_id

Primary Key Constraint: No two tuples can have the same values for outgoing_id.

Entity Integrity Constraint: The outgoing_id cannot be null.

Not Null Constraint: The supplier_invoice_id attribute cannot be null.

Supply Purchase Order

Supply_purchase_id	Integer
Supply_purchase_time	Datetime
employee_id	Integer
supplier_id	Integer

Primary Key: Supply_purchase_id

Primary Key Constraint: No two tuples can have the same values for supply_purchase_id.

Entity Integrity Constraint: Supply_purchase_id cannot be null.

Not Null Constraint: The supply_purchase_time attribute cannot be null.

Work History

History_id	Integer
Start_date	Datetime
End_date	Datetime
Job_title	Varchar(255)
Pay_rate	decimal
employee_id	Integer

Primary Key: History_id

Primary Key Constraint: No two tuples can have the same values for history_id.

Entity Integrity Constraint: The history_id cannot be null.

Not Null Constraint: The start_date, job_title, and pay_rate attributes cannot be null.

Work Shift

Shift_ID	Integer
Start_date	date
Begin_time	time
End_time	time
employee_id	Integer

Primary Key: shift_id

Primary Key Constraint: No two tuples can have the same values for history_id.

Entity Integrity Constraint: The shift_id cannot be null.

Not Null Constraint: No fields in this relation may contain null values.

2.3.2 Sample Data of Relation

Customer

Customer Id	**fName**	**lName**	**Street**	**city**
1000921412	Seymour	Skinner	1234 Belle Terrace	Bakersfield
1000974562	Charlie	Day	999 Philadelphia St CA	Bakersfield
1000974125	Michael	Scott	5453 Business Park Blvd.	Bakersfield
1000452337	Jim	Halpert	91910 Quarry Way	Bakersfield
1000684362	Dwight	Schrute	1111 Farm Ave.	Shafter
1098784321	Dennis	Reynolds	5678 Pennsylvania Ct.	Bakersfield
1007096011	Charles	Boyle	34353 New York Ave.	Bakersfield
1004342343	Terry	Jeffords	7871 Justice Ct.	Bakersfield
1032442344	Andrew	Dwyer	1245 Indiana St.	Bakersfield
1934838822	Tom	Haverford	3432 Cologne Way	Bakersfield

state	**zip**	**username**
CA	93305	PrincipalS
CA	93314	Dayman111
CA	93311	MichaelGaryScott
CA	93301	Prankster80

CA	93263	BeetFarmer
CA	93315	GoldenGod
CA	93305	CookingLvr
CA	93311	TerryLovesYogurt
CA	93304	Champion1
CA	93312	TommyTom

Password	**Email**	**Acc_creation_date**	**Phone_number**
hdQw1mgW81H	SgtSeymour@gmail.com	02/13/2017 11:34:09	6615550001
S8hg62jJimwZ	Nightmancometh@gmail.com	06/19/2016 22:59:27	6615555079
g0HD4k8cEk	Worldsbestboss@gmail.com	12/29/2016 13:41:53	6615551214
9cB0vXehY71C	Prankster80@gmail.com	08/07/2017 10:19:35	6615558963
b9Kls4HP1cw0	Beetfarmer@gmail.com	10/15/20/17 05:01:06	6615554183
9Pc8Hw37Xz	Goldengod@gmail.com	01/19/2016 12:36:44	6615554862
Uf9ws71dGKq	Foodandwine@gmail.com	04/22/2018 09:16:29	6615552846
D1g13T7s2bvO	yogurtterry@gmail.com	07/28/2017 15:41:11	6615559712
pH1kA4jHh82Z	fellinthepit@yahoo.com	02/14/2016 14:48:56	6615551948
kJj4UaIP3I	tommyfresh@gmail.com	08/02/2017 10:55:13	6615551436

Delivery Address

address_id	street	city	state	zip
45678945	2042 Washington Way	Bakersfield	CA	93307
45687865	2100 Jump St.	Bakersfield	CA	93304
12312332	4178 Evergreen Terrace	Bakersfield	CA	93311
12312313	3333 Elm St.	Oildale	CA	93308
456456456	76129 Baker St.	Bakersfield	CA	93307
123489545	22256 Paper Ct.	Bakersfield	CA	93305
212345654	1011 Sesame St.	Shafter	CA	93263
156845651	5144 Wisteria Ln	Bakersfield	CA	93307
144845212	7712 Rainey St.	Bakersfield	CA	93307
012140548	4389 Power St.	Bakersfield	CA	93305

Employee

Employee_id	fName	lName	Street
01515494	Michael	Scott	5453 Business Park Blvd.
01598487	Jim	Halpert	91910 Quarry Way
19887871	Dwight	Schrute	1111 Farm Ave.

01877987	Pam	Beesley	91910 Quarry Way
00018077	Creed	Bratton	90010 Quarry Way
01984770	Stanley	Hudson	5908 Pacific St.
19848770	Ryan	Howard	8542 Stonetree Way
18800870	Kelly	Kapoor	98721 Windmill Ct.
28747462	Meredith	Palmer	1235 Decatur St.
54411223	Kevin	Malone	546 Christmas Tree Ln.

City	**State**	**Zip**	**Phone_number**
Bakersfield	CA	93311	6615551214
Bakersfield	CA	93301	6615558963
Shafter	CA	93263	6615554183
Bakersfield	CA	93301	6615558964
Bakersfield	CA	93311	6615550000
Bakersfield	CA	93314	6615551778
Bakersfield	CA	93305	6615551478
Bakersfield	CA	93309	6615557532
Oildale	CA	93308	6615551117
Bakersfield	CA	93306	6615559879

Flower Product

Product_id	**Product_name**	**Sell_price**	**Purchase_price**

00125441	Rose	9.99	6.99
00185812	Tulip	4.49	1.99
00188987	Baby’s Breath	1.99	0.49
00121141	Hydrangea	6.99	4.99
00177989	Daffodil	3.99	1.49
00178781	Lily	5.49	3.99
00117079	Chrysanthemum	1.49	0.49
00978740	Gerbera	4.49	3.49
00987112	Carnation	2.49	1.49
00185580	Carnation	3.49	1.99

Color	**Length**	**Product_image**	**Description**	**supply_purchase_id**
Red	12 in	redrose.png	A beautiful, thornless red rose.	01984770
Yellow	10 in	yellowtulip.png	A beautiful tulip with a large yellow bulb.	12312345
White	6 in	whitebreath.png	A common flower filler with small flowers coming off its branches.	12315489
Light Blue	8 in	bluehydrangea.png	Contains small flowers in bunches at the end of a long stem.	12313152
Yellow	10 in	yellowdaffodil.png	Contains a trumpet shaped petal	15654568

			surrounded by a star shaped petal.	
White	8 in	whitelily.png	Big flowers with a large petal span.	12354984
Yellow	10 in	yellowchrsanthemum.png	Blooms into a large beautiful flower.	78954654
Pink	8 in	pinkgerbera.png	A part of the sunflower and daisy family. Appears to look like a colorful sunflower.	12321434
Pink	6 in	pinkcarnation.png	A commonly known flower with branched or forked clusters.	00321422
Orange	8 in	orangebirdofparadise.png	Known for its distinct exotic look.	10320000

Incoming Payment

Incoming_id	Sales_tax
2003584124	0.0725
2349021343	0.0525
1234162132	0.0750
3143214532	0.0550
3512134643	0.0750
4321213554	0.0800

5231432143	0.0800
4321315123	0.0550
3123125321	0.0600
5321432143	0.0625

Order Status

Status_id	Status
01	new order
02	checked availability
03	credits checked
04	packed
05	out for delivery
06	delivered
07	delivery attempted - not received
08	contact customer
09	cancelled
-1	in store purchase

Outgoing Payment

Outgoing_id	Supplier_invoice_id
2000584534	26323452342
2045430345	234532
3494594333	32442345432
4060593054	4543234
0000012343	643243643

1234543233	462345454334
1234353234	64328676
1234232343	3234345
6432543223	2345432345
2345432123	3262454335

Package

Package_id	**Expected_delivery_time**	**Message**
2000587246	02/14/2020 11:30:00	“Happy Valentine’s Day”
2000587247	02/14/2020 11:40:00	“Happy Valentine’s Day”
2000541479	02/14/2020 08:00:00	null
2004688787	04/01/2020 13:05:00	“Happy Birthday!”
2008787997	09/04/2019 14:45:00	“Sorry for your loss”
2000148631	05/28/2018 11:25:00	“Congratulations!”
2000357498	08/04/2018 12:30:00	null
2007854123	11/22/2018 10:45:00	“Happy Thanksgiving”
2008569871	12/24/2018 12:50:00	“Merry Christmas”
2001547112	12/30/2017 13:15:00	“Happy New Year”

p_order_num	**employee_id**
45645645	01598487
07907011	00018077
45645678	28747462
45645678	01984770

09770454	01598487
45645645	28747462
45645677	01984770
44567895	54411223
11234595	01598487
45677785	00018077

Payment

Payment_id	**Payment_time**	**Amount**	**employee_id**	**payment_type_id**
017787700	02/09/2020 10:34:43	49.99	01598487	02
017787701	02/09/2020 10:36:18	34.99	00018077	01
017789781	02/12/2020 15:01:56	74.49	28747462	03
017988711	02/13/2020 16:58:08	64.24	01984770	02
017998712	02/14/2020 03:04:33	58.67	01598487	04
018000701	03/05/2020 12:14:48	33.58	28747462	05
018018070	04/18/2020 13:45:29	44.49	01984770	08
018070101	08/20/2020 09:18:45	14.44	54411223	07
018070711	09/01/2020 00:30:54	99.99	01598487	09

018870702	12/20/2020 19:54:06	128.53	00018077	01

Payment Type

Payment_type_id	Description
01	Cash
02	Credit Card - In store
03	Debit Card - In store
04	Check
05	Gift Card - In store
06	Coupon
07	Instore Credit
08	Credit Card - Online
09	Debit Card - Online
10	Gift Card - Online

Product Order

P_order_number	Order_time	customer_id
09098970	02/09/2020 10:34:43	1000921412
08970078	02/09/2020 10:36:18	1000974562
07907011	02/12/2020 15:01:56	1000974125
09877001	02/13/2020 16:58:08	1000452337
01264684	02/14/2020 03:04:33	1000684362
09770454	03/05/2020 12:14:48	1098784321
07061004	04/18/2020 13:45:29	1007096011
08987001	08/20/2020 09:18:45	1004342343

08899011	09/01/2020 00:30:54	1032442344
09870331	12/20/2020 19:54:06	1934838822

status_id	**employee_id**	**address_id**
02	01598487	45687865
04	00018077	12395651
06	28747462	45678954
02	01984770	12456545
06	01598487	78945623
04	28747462	12324532
02	01984770	45678954
06	54411223	12355545
02	01598487	11123548
06	00018077	44456458

Recipient

Recipient_id	**fName**	**lName**
00079456	Deandra	Reynolds
00045997	Liam	McPoyle
00012345	Maureen	Ponderosa
00787845	Matthew	Mara
00365556	Barbara	Reynolds
00148755	Margaret	McPoyle
00123545	Squilliam	Fancyson

00015154	Patrick	Star
01212154	Robert	Trousers
10000078	Sandy	Chi

Phone_Number	**package_id**
6614545875	01378970
6614548784	01537014
6619875642	01377755
6617842542	01256743
6614874525	01388132
6612354874	01837539
6616875309	01389166
6611234567	01160714
6612564897	01561986
6618975451	01221498

Supplier

Supplier_id	**Vendor_name**	**Street**	**City**
78945648	Kern Roses	12343 Taft Hwy,	Taft
78956123	Taft Daisies	15888 Taft Hwy,	Taft
78954562	Bakersfield Tulips	23453 Weedpatch Rd.	Bakersfield
78954452	Sun Valley Group	53243 Sycamore Rd.	Bakersfield
35489545	Luffa Farm	54324 Panama Rd.	Bakersfield

85462152	Rose Story Farm	13241 Ribier Rd.	Lamont
78954562	Kendall Farms	53234 Edmundson Acres	Arvin
12345678	Kilcoyne Lilac Farm	45453 E Bear Mountain Blvd.	Arvin
78945623	Ori’s Orchid’s	12343 Old River Rd.	Bakersfield
12345858	Mary’s Marigold’s	45434 Millux Rd	Bakersfield

State	**Zip**	**Phone_number**
CA	93268	6615889898
CA	91231	6612342343
CA	93312	6612345643
CA	93308	6619873452
CA	93234	6615837294
CA	90001	6612839219
CA	90012	6612727383
CA	90321	6612342322
CA	93312	6615555555
CA	93305	6615893275

Supply Purchase Order

Supply_purchase_id	**Supply_purchase_time**	**employee_id**	**supplier_id**
100000005	01/05/2010 08:00:01	54411223	100000069

100000234	12/05/2011 09:32:23	01984770	100000420
100005432	03/12/2012 10:33:12	28747462	100000656
100006443	04/12/2014 09:45:11	28747462	123456789
100007543	07/12/2015 08:40:54	01598487	105454585
100008625	11/12/2016 10:30:35	28747462	012345654
100009750	09/12/2017 11:20:45	28747462	001224555
100010800	06/12/2018 12:25:55	01984770	001215544
100011901	03/12/2019 14:34:59	54411223	012124545
100012925	02/12/2020 16:50:23	54411223	100450001

Work History

History_id	**Start_date**	**End_date**	**Job_title**	**Pay_rate**	**employee_id**
00043	10/12/2010 08:00:00	null	Florist	14.00	01692945
00323	01/23/2011 12:00:00	01/23/2014 16:00:00	Delivery Driver	13.00	01668596
00323	11/25/2015 14:00:00	null	Cashier	13.00	01431928
00142	10/12/2008 08:00:00	10/03/2010 16:00:00	Florist	10.00	01538090
00321	10/12/2008 08:00:00	05/20/2011 14:00:00	Delivery Driver	13.25	01353823

00334	11/23/2018 08:00:00	05/20/2019 14:00:00	Florist	13.50	01610434
00343	11/15/2007 08:00:00	null	Manager	18.00	01985315
00456	05/10/2018 08:00:00	null	Cashier	13.50	01135503
00123	02/12/2017 14:00:00	null	Delivery Driver	14.00	01473241
00212	01/12/2016 07:00:00	01/12/2016 20:30:00	Florist	13.75	01595995

Work Shift

shift_id	**shift_date**	**start_time**	**end_time**	**employee_id**
00043	10/12/2010	11:00:00	16:00:00	01692945
00323	01/23/2011	08:00:00	18:00:00	01668596
00323	11/25/2015	11:00:00	16:00:00	01431928
00142	10/12/2008	08:00:00	14:00:00	01538090
00321	10/12/2008	14:00:00	16:00:00	01353823
00334	11/23/2018	11:00:00	15:00:00	01610434
00343	11/15/2007	16:00:00	16:00:00	01985315
00456	05/10/2018	08:00:00	11:00:00	01135503
00123	02/12/2017	11:00:00	15:00:00	01473241
00212	01/12/2016	08:00:00	15:00:00	01595995

Contains

p_order_number	**product_id**	**quantity_item**	**point_of_sale_price**

78945648	00000301	6	40.22
12345678	00000300	1	31.69
45682184	00000303	4	52.12
48754125	00000307	8	56.23
12315652	00000312	9	85.23
15648954	00000314	5	56.23
21235489	00000324	7	26.45
21562126	00000345	4	45.22
21564556	00000325	5	32.11
12312348	00000354	8	45.12
45654123	00000356	5	15.25
00012154	00000375	6	12.02
15615645	00000380	3	22.35
12320151	00000396	6	32.56
00121445	00000401	4	12.25
00012102	00000411	2	45.20
00012789	00000402	44	96.45
00124555	00000423	46	65.23
00789545	00000425	42	80.23
00121455	00000430	21	50.21
00121547	00000423	15	45.60
00154578	00000467	5	45.50
20515456	00000445	4	32.65
10515651	00000470	7	32.12
01564895	00000475	9	60.12

01561546	00000480	5	50.12
08984512	00000578	6	45.21
01564895	00000534	8	45.62
01564895	00000562	4	12.12
01565456	00000402	5	45.12
01564891	00000502	6	21.30
20156489	00000702	9	30.12
30156785	00000802	8	40.52
30156546	00000734	5	25.41
40012105	00000750	1	9.99
01516456	00000800	10	24.12
01564898	00000802	15	30.23
90944452	00000890	16	40.25
45787895	00000902	45	60.65
10263542	00000913	45	30.21
12654895	00000945	8	21.45
45678954	00000950	30	60.23
84745621	00000952	20	45.21
04515654	00001051	15	30.25
89456123	00001053	14	23.12
12345678	00000345	11	20.45
21345621	00000502	3	16.25
47895462	00000804	6	23.45
12345621	00000325	4	20.12
47895456	00000450	8	40.56

12345678	00000202	9	70.65
01204574	00000043	8	40.25
11445545	00000001	5	45.23
66998545	00000443	1	13.23
74158852	00000342	2	15.23
15648954	00000524	5	20.45
01456521	00000414	10	60.25
01548795	00000415	13	70.25
01565489	00000412	14	73.25
01215645	00000490	20	50.23

Refills

supply_purchase_id	**product_id**	**quantity_item**	**supply_price**
12456212	00000412	60	300.00
45021548	00001262	45	200.00
45602456	00000012	80	325.00
01548795	00000456	60	350.00
45689051	00000490	50	200.00
21565456	00000500	70	125.00
15654565	00000512	90	300.00
15156512	00000812	56	250.00
51545621	00000912	90	300.00
26545621	00000402	60	200.00
26212421	00000415	50	250.00
15987985	00000416	65	225.00

15951945	00000420	75	300.00
54954954	00000500	80	350.00
15648954	00000985	45	225.00
89462154	00000789	60	300.00
48975466	00000812	90	350.00
78978456	00000905	60	325.00
12448454	00000900	80	325.00
48784545	00000995	90	350.00
11156788	00001000	60	375.00
14774441	00000412	65	400.00
45151489	00000789	70	425.00
14489545	00000800	100	400.00
11456545	00000812	90	320.00
15654895	00000850	75	330.00
11234595	00000950	80	312.00
49489789	00001000	90	333.00
98784561	00000500	70	325.00
49415652	00000612	75	225.00
15648985	00000712	80	325.00
45654562	00000234	60	300.00
77789545	00000789	65	300.00
77895456	00008456	90	225.00
11145987	00000500	85	250.00
47895456	00000601	90	400.00
12456897	00000705	60	332.00

48954652	00000800	75	325.00
48795456	00000850	60	400.00
12346578	00000900	65	425.00
45678954	00000412	70	325.00
41145687	00000500	90	352.00
12364568	00006004	100	400.00
55595789	00004564	150	400.00
11154895	00007895	120	425.00
44447895	00045685	125	400.00
44456898	00078954	130	400.00
66998545	00047895	145	400.00
44489789	00060078	150	425.00
44487895	00074589	165	450.00
11156785	00048789	75	325.00
11156489	00078954	80	225.00
54546845	00004567	90	300.00
17895545	00054895	95	325.00
00123459	00000789	100	350.00
45641235	00045685	90	300.00
48978955	00045654	65	325.00
11145685	00405689	60	300.00
21567895	00045687	120	325.00

Requires

p_order_number	payment_id
01154885	00089967
00273109	00000524
50706593	00053981
42118781	00000385
82758239	00078970
01444092	00005471
01555534	00037014
46252404	00003774
30402172	00031174
00672471	00000798
06822989	00000999
72801991	00038184
06809341	00009841
16683911	00003918
13356307	00033177
04763679	00009385
58007373	00000102
25841699	00090194
70960440	00000431
03650103	00058892
34595198	00000819
17245099	00007890
03628812	00064804
02953380	00005918

51515614	00008760
23010099	00001823
06508100	00000490
29294845	00002097
89078779	00000775
78345273	00003728
36618043	00007347
18418033	00000959
72828203	00030855
45169966	00000386
06849768	00003292
79450957	00022753
11294349	00000230
25792127	00003454
52331780	00011595
96427525	00000313
05070272	00001748
65345488	00051213
47402655	00000793
17571516	00001104
34135700	00016902
23047965	00000157
42264189	00009621
07904002	00035125
11423471	00093830

05059014	00000734
96899183	00009734
48199083	00095038
05564523	00000186
72275806	00005739
03904705	00036333
11665370	00000242
02775592	00033116
13399409	00009876
12884495	00045263

Needs

payment_id	supply_purchase_id
44487895	00000895
52870128	00001995
42956775	00004652
43265380	00001219
88767226	00000830
50694627	00004898
03223264	00006273
01363692	00004328
29908550	00002521
31963428	00001610
72932329	00000537

33095363	00017091
41612477	00023930
87026834	00009042
29128746	00045546
91995701	00007091
47866227	00001807
38789436	00000998
66887163	00009444
17340937	00005502
80138058	00004131
40917255	00000178
99478788	00034603
10965503	00000220
69150139	00003638
61277560	00005480
93282735	00008861
44222702	00000713
85349223	00002348
62871288	00011137
43073255	00022129
98378602	00000263
57881841	00068101
92951147	00000620
33946750	00074382
45818244	00000378

46026598	00004211
45598572	00006899
85345308	00031891
58789606	00018845
35146304	00071199
36657559	00002170
30964915	00004805
29820344	00015002
40427714	00064461
83006157	00000202
80125694	00000498
41194749	00008603
13271489	00029158
87008154	00006567
18089890	00005428
70724171	00007000
90772528	00090376
74015760	00001435
90184792	00094130
68321105	00000749
53716319	00032145
10931456	00005489
86980245	00009899

2.4 Sample Queries to Our Database

Section 2.4 will focus on a few sample queries that can be used to retrieve certain data and information from our database. Our sample queries will be presented as relational algebra, tuple relational calculus, and domain relational calculus.

The sample queries provided will demonstrate a couple of operations. For example, operations such as select, project, cartesian product, and join will be used in relational algebra. We'll also show examples of the division operation used in relational algebra

2.4.1 Design of Queries

We will discuss the three main formal query languages: relational algebra and relational calculus. Relational calculus consists of two calculi which are tuple relational calculus and domain relational calculus. We will be using relational algebra, tuple and domain relational calculus to express our sample queries.

There will be a total of ten sample queries. In the following three sections, we will express all ten samples in the three main formal query languages as explained previously.

2.4.2 Relational Algebra for Queries of 4.1

Relational algebra is an algebra whose operations are designed to retrieve tuples from our database. A tuple is one record (a row). A relational algebra expression combines fundamental operations to return a set of tuples and describes the process of doing so from a relational database. This language is procedural, so the order and how these expressions are nested matters.

Some operations for relational algebra are as follows: select (σ), project (Π), cartesian product (X), set different (-), union ($\cup$), and join($\bowtie$). Selection picks rows. Projection picks columns. Cartesian products join two relations. Union can only be used if two relations are union compatible to give a relation with tuples which are either in one relation or in the other. Join operation allows joining variously related tuples from different relations. There are different types of joins.

1. List customers who have made at least 2 product orders between 1/18/20 and 2/18/20.

$P1 \leftarrow \sigma_{\text{order\_time >= 1/18/20} \wedge \text{order\_time <= 2/18/20}}(\text{Product Order})$

$P2 \leftarrow \sigma_{\text{order\_time >= 1/18/20} \wedge \text{order\_time <= 2/18/20}}(\text{Product Order})$

$\Pi_{\text{customer\_id, name}}(\text{ Customers * } (\sigma_{\text{P1.p\_order\_number} \neq \text{ P2.p\_order\_number}} (\text{P1 x P2}))$

2. List customers with accounts on our website that have not made a product order in the past 6 months.

$P1 \leftarrow \sigma_{\text{order\_time >= currentDate - 6 months}} (\text{Product Order})$

$\Pi_{\text{customer\_id, name}}(\ \sigma_{\text{username != NULL}} \text{ Customers * } (\Pi_{\text{product\_order\_number}}(\text{Product Order - P1})))$

3. List employees who purchased flower products from every supplier.

$\Pi_{\text{employee\_id, employee\_name}}(\text{Employee * } (\Pi_{\text{supply\_purchase\_id}}\text{Supply Purchase Order } \div \Pi_{\text{supplier\_id}}\text{Supplier}))$

4. List product orders with a payment greater than $100 that have been delivered.

$\Pi_{\text{product\_order\_number}} (\sigma_{\text{status = ‘delivered’}} (\text{Product Order * Order Status}) \text{ * } \Pi_{\text{payment\_id}}(\sigma_{\text{amount > 100}} \text{ Payment}))$

5. List current employees who have processed all John Doe’s purchases.

$\Pi_{\text{empolyee\_id, employee\_name}} (\sigma_{\text{end\_date = NULL}} (\text{Employee * Work History}) \div \Pi_{\text{p\_order\_number}}(\sigma_{\text{name = ‘John Doe’}} (\text{Product Orders * Customers})$

6. List the package(s) that has the second least expensive product order.

$P_2 \leftarrow (\text{Payment - } \Pi_{\text{p1.*}} (\sigma_{\text{p1.amount > p2.amount}}(\text{Payment x Payment}))$

Package * ($\Pi_{product_order_number}$Product Order * ($\sigma_{p2.amount\ != p3.amount}P_2$ - (Payment - $\Pi_{p1.*}$($\sigma_{p1.amount > p3.amount}$(Payment x Payment))))

7. List recipients who have never received red roses.

P1 ← Packages * ($\sigma_{product\_name = \text{‘Red Roses’}}$ (Flower Product * Product Orders)))

$\Pi_{recipient\_id,\ R.name}$ (Recipients * (Packages - P1))

8. List the suppliers that have no supply purchase order with more than 1 flower product.

F1 ← $\Pi_{f1.product\_id,\ f2.product\_id}$($\sigma_{f1.product\_name\ != f2.product\_name}$(Flower Product x Flower Product))

Supplier * ($\Pi_{supply_purchase_id}$Supply Purchase Order - ($\Pi_{suppy\_purchase\_id}$(Supply Purchase Order * F1))

9. List customers who have purchased all flower products.

$\Pi_{customers\_id,\ c.name}$(Customers * ($\Pi_{product\_order\_number}$ Product Order ÷ $\Pi_{product\_name}$ (Flower Product)))

10. List the cheapest package delivered by John Doe.

$\Pi_{package\_id}$ ($\sigma_{name = \text{‘John Doe’}}$ Employee * (Package * $\Pi_{p1.*}$($\sigma_{p1.amount < p2.amount\ \wedge p1.payment\_id\ != p2.payment\_id}$ (Payment x Payment))))

2.4.3 Tuple Relational Calculus Expressions for Queries

Tuple relational calculus depends on the use of tuple variables. The language is non-procedural. This means the order of operations needed to retrieve a set of tuples does not matter. It is declarative, so it does not explain how to solve a query, instead it only provides a description. It uses existential and universal quantifiers in the declarative expressions to check if a condition is true or false. It will check if every possible tuple meets the conditions to make the declarative expression true or false.

An example as to how a query in tuple relational calculus is expressed is as follows: { t | P(t) }. A variable associated with a existential (∃) and universal variables (∀) are known as bounded variables. Bounded variables are any tuple variable with a “for all” or “there exists” condition. Free variables are any tuple variable without ∃ or ∀ .

1. List customers who have made at least 2 product orders between 1/18/20 and 2/18/20.

{ c | Customers(c)∧(∃ p_1) (Product Order (p_1) ∧(∃ p_2) (Product Order (p_2)
∧p_1.product_order_number != p_2.product_order_number
∧p_1.customer_id = c.customer_id∧p_2.customer_id = c.customer_id
∧p_1.order_time <= 2/18/20∧p_1.order_time >= 1/18/20
∧p_2.order_time <= 2/18/20∧p_2.order_time >= 1/18/20))
}

2. List customers with accounts on our website that have not made a product order in the past 6 months.

{ c | Customers(c)∧c.username != NULL ∧(∃ po_1)(Product Order(po)
∧po_1.customer_id = c.customer_id
∧ ¬(∃ po_2)(Product Order(po_2)∧po_2.order_time >= currentDate - 6months))
}

3. List employees who purchased flower products from every supplier.

{ e | Employees(e) ∧(∀ s)(Supplier (s) → (∃ sp)(Supply Purchase Order (sp)
∧(∃ f) Flower Products (f) ∧sp.supply_purchase_order =
f.supply_purchase_order∧sp.employee_id = e.employee_id
∧s.supply_id = sp.supply_id))
}

4. List product orders with a payment greater than $100 that have been delivered.

{ po | Product Order(po) ∧(∃ os)(∃ p)(Payment(p) ∧Order Status (os)
∧po.p_order_number = os.p_order_number
∧po.payment_id = p.payment_id

∧os.status = ‘delivered’ ∧p.amount > 100)

}

5. List current employees who have processed all John Doe’s purchases.

{ e | Employee(e) ∧(∀ po)(∃ c) (Product Orders(po)

∧ Customers(c) ∧c.name = ‘John Doe’ ∧c.customer_id = po.customer_id

→ (∃ w)(Work History(w)∧e.employee_id = w.employee_id

∧w.end_date = NULL))

}

6. List the package(s) that has the second least expensive product order.

{ p | Package(p) ∧(∃ po)(∃ pt_1)(Product Order(po) ∧Payment(pt_1)

∧po.p_order_number = pt_1.p_order_number

∧p.package_id = po.package_id

∧(∃ pt_2)(Payment (pt_2) ∧pt_2.amount < pt_1.amount

∧ ¬(∃ pt_3) (Payment(pt_3) ∧$pt_{3.}$amount < pt_1.amount

∧pt_2.amount ≠ pt_3.amount)))

}

7. List recipients who have never received red roses.

{ r | Recipient(r) ∧(∃ p_1) (Packages(p_1)∧

∧ ¬(∃ p_2)(Packages(p_2) ∧(∃ f)(∃ po) (Flower Product(f) ∧ Product Orders(po)

∧po.p_order_number = f.p_order_number

∧p_2.package_id = r.package_id

∧f.product_name = ‘red roses’))

}

8. List suppliers that have no supply purchase order with more than 1 flower product.

{ s | Supplier (s) ∧(∃ sp)(Supply Purchase Order(sp)

∧sp.supplier_id = s.supplier_id

∧ ¬(∃ f_1)(∃ f_2) (Flower Product(f_1)∧Flower Product(f_2)

∧f_1.product_name != f_2.product_name))

}

9. List customers who have purchased all flower products.

{ c | Customers(c) ∧(∀ f) (Flower Products (f) → (∃ po) (Product Order(po)

∧po.customer_id = c.customer_id

∧po.product_id = f.product_id))

}

10. List the cheapest package delivered by John Doe.

{ p | Package(p)∧(∃ pt_1)(∃ pt_2) (Payment(pt_1) ∧Payment(pt_2)

∧pt_1.amount < pt_2.amount ∧pt_1.payment_id != pt_2.payment_id

∧p.package_id = pt.package_id

∧(∃ e) (Employee(e) ∧e.name = ‘John Doe’

∧p.employee_id = e.employee_id))

}

2.4.4 Domain Relational Calculus Expressions for Queries

Domain relational calculus uses variables that will take their values from domains of attributes rather than tuples of relations like in tuple relational calculus. Each variable represents a single value with a tuple instead of a list of values. It is also non-procedural and uses existential and universal quantifiers. It gives a description of the query but does not give ways to solve it.

Domain relational calculus has the following format: { d_1, d_d, …, d_n | F(d_1, d_2, …, d_n) }. The < d_1, d_d, …, d_n > represents resulting domains variables. The F(d_1, d_2, …, d_n) represents the condition equivalent to the Predicate calculus - a boolean expression over d_1, d_2, …, d_n. The predicate has a set of comparison operators, connectives, and quantifiers.

1. List customers who have made at least 2 product orders between 1/18/20 and 2/18/20.

{ <c, nm> | Customers(c, nm, _, _, _, _, _, _)
∧(∃ pn_1)(∃ pn_2)(Product Order(pn_1, >= 1/18/20)∧Product Order(pn_2, <=
2/18/20)∧Product Order(pn_2, >= 1/18/20) ∧Product Order(pn_2, <= 2/18/20)∧pn_1
!= pn_2)
}

2. List customers with accounts on our website that have not made a product order in the past 6 months.

{<c, nm> | Customers(c, nm, _ ,!=NULL , _, _, _, _)
∧(∃ pn_1)(Product Order(pn, _)
∧ ¬(∃ pn_2)(∃ ot) (Product Order(pn_2, ot) ∧ot >= currentDate - 6 months)
}

3. List employees who have purchased flower products from every supplier.

{ < e, nm > | Employee(e, nm, _, _) ∧(∀ s) (Supplier(s,_ ,_ ,_)
→ (∃ f) Flower Products(f, _, _, _, _, _, _, _)
∧(∃ sp)(Supply Purchase Order (sp, _,)))
}

4. List product orders with a payment greater than $100 that have been delivered.

{ <pn> | Product Orders (pn, _) ∧Payment(_, > 100, _)
∧Order Status(_, ‘delivered’)
}

5. List current employees who have processed all John Doe’s purchases.

{ <e, nm> | Employee(e, nm, _, _) ∧(∀ pn)(Product Order(pn, _)
∧Customers(_, ‘John Doe’, _, _, _, _, _, _)
→ (∃ h)(Work History (h, _, NULL, _, _)))
}

6. List the package(s) that has the second least expensive product order.

{ <pi> | Package(pi, _, _)∧Order Status(_ , ‘delivered’)

∧(∃ pn)(Product Order(pn, _)

∧(∃ a_1)(Payment(_, a_1, _)∧(∃ a_2)(Payment(_, a_2, _) ∧$a_1 > a_2$

∧ ¬(∃ a_3)(Payment(_ , a_3, _) ∧$a_1 > a_3$ ∧a_2 != a_3))))

}

7. List recipients who have never received red roses.

{ <r, nm> | Recipient(r, nm, _)

∧(∃ pi)(Package(pi, _, _)∧ ¬(∃ pn) (Product Order(pn, _)

∧Flower Product(_, ‘red roses’, _, _, _, _, _, _)))

}

8. List suppliers that have no supply purchase order with more than 1 flower product.

{ <i, vn> | Supplier (i, vn, _ , _) ∧ ¬ (∃ sp)(∃ f_1)(Supply Purchase (sp, _)

∧Flower Product (f_1, _, _, _, _, _, _, _)

∧ ¬(∃ f_2) Flower Product (f_2, _, _, _, _, _, _, _))

}

9. List customers who have purchased all flower products.

{ <c, nm> | Customers(c, nm, _, _, _, _, _, _)

∧(∀ p) (Flower Product(p, _, _, _, _, _, _, _)

→ (∃ o) (Product Order(o, _)))

}

10. List the cheapest package delivered by John Doe.

{ < p> | Package(p, _, _) ∧(∃ a_1)(Payment(_, a_1, _)

∧(∃ a_2)(Payment(_, a_2, _) ∧(a_1< a_2)

∧Employee (_, ‘John Doe’, _, _, _)))

}

Phase 3: Relational Database Normalization and Implementation

Up to this point we have only defined our database in conceptual terms and have not implemented it into a physical database. Using what we have learned from making a conceptual database using the ER model and honing it's logical design using the relational model we will now implement it into a physical database.

This first section of this phase will go over the ways we ensure our data is normalized. The next section will go over the DBMS we are using postgres and some of the advantages it offers. The next section will discuss the schema object allowed by postgres. The last section will display the results of the queries we designed in phase 2 and how we translated them into SQL queries.

3.1 Normalization of Relations

Normalization is the method of arranging the data in a database to prevent data duplication or redundancy. Redundancy is where information in a database is being stored in more than one place. Eliminating data repetition helps improve the data integrity of a database. Useless data should be removed and only related data is stored in each table. There is a series of normal forms used to achieve normalization.

3.1.1 Normalization and Anomalies

The concept of normalization will be discussed prior to analyzing the quality of the relation schemas in the Bakersfield Flower Shop database. In this section, we will be determining how to measure normalization using the following normal forms: First Normal Form, Second Normal Form, Third Normal Form, and Boyce-Codd Normal Form. It is critical that we find relations that may not be normalized before implementing our physical database. A poorly designed database will lead to additional problems called update anomalies.

Update anomalies can be classified into three types: insert anomaly, delete anomaly, and modification anomaly. Modification anomaly is also known as update anomaly. Satisfying the series of normal forms will help us avoid these anomalies we may encounter when we would want to modify our physical database. If there is such a relation that does not satisfy up to the Third Normal Form and/or Boyce-Codd Normal Form, we would have to “break apart” the relation schemas so that this redundancy is removed. Oftentimes, we can resolve normalization issues by using NULL .It can waste space at storage level and may cause some misunderstanding with the meaning of attributes as well with specifying JOIN operations at the logical level. However, there are times where NULLs will be unavoidable.

Description of Normal Forms

As mentioned previously, we can’t implement our logical database as a physical database until we check the quality of the relations. Each relation must be normalized and go through a number normal form tests. There are four normal forms and we will be redesigning our database, if necessary, to ensure that it satisfies those four. The higher the normal form a relation schema satisfies, the more normalized it is.

First Normal Form

A relation in First Normal Form, or 1NF, satisfies the following two conditions: the domain of each attribute contains only atomic values, and the value of each attribute contains only a single value from that domain. An atomic value is a value that can’t be split into smaller pieces. In other words, a column can’t be broken down into sections with more than one type of data, therefore, one part is dependent on another part of the same column. A relation in 1NF allows there to be no repeating groups in individual tables, seperate table for each set of related data can be created, and each set of related data with a primary key can be identified.

Let’s make the table Customer have attributes Customer ID, First Name, Last Name, and Phone Number. Let Phone Number be a multi-value attribute in this case. That means there exists more than one phone number separated by a comma for each customer. Column values are not atomic. To comply with 1NF, customers can be duplicated each associated with only one phone number, but that would make

Customer ID no longer unique. To resolve this so it may satisfy both 1NF requirements, Customer may be split into two tables: Customer Name and Customer Phone Number Table. Customer Name would have Customer ID, First Name, and Last Name as attributes. Meanwhile, Customer Phone Number would have an ID, the Customer ID and Phone Number as attributes.

Second Normal Form

A relation in its Second Normal Form, or 2NF, satisfies the following two requirements: it is in First Normal Form and all non-prime attributes are not functionally dependent on any proper subset of any candidate key of the relation. A non-prime attribute of a relation is an attribute that isn’t part of any candidate key of the relation. If attribute B is functionally dependent on A but not functionally dependent on a proper subset of A, then B is considered fully functional dependent on A. All non-key attributes can’t be dependent on a subset of the primary key.

Let’s make the table Flower Product consist of Supplier, Product Name, Product Full Name, and Supplier Country as attributes. Candidate key is {Supplier, Product Name}. The table is not in 2NF since Supplier Country is a non-prime attribute functionally dependent on a part of a candidate key which is Supplier. To conform to 2NF, the table can be split into two. One table named Flower Product Suppliers with Suppliers and Supplier country as attributes. The other table is named Flower Product Name with Supplier, Product Name, and Product Full Name as attributes.

Third Normal Form

A relation in its Third Normal Form, or 3NF, satisfies the following two conditions: it is in Second Normal Form and there can’t be any non-prime attributes of R that are transitively dependent on every key of R. A non-prime attribute of R is an attribute or column that doesn’t belong to any candidate key of R. Transitive dependency in simple terms means that a column’s value is dependent on another column through a second intermediate column. To achieve 3NF, a relation schema can be broken down into relations where the left side of a functional dependency is always a primary key attribute.

Let's make the table Employee consist of the following attributes: Employee ID, Name, Street, City, State, Zip. The candidate key, in this case, is {Employee ID} as it will help uniquely identify a row. Although this table satisfies one of the conditions for 3NF, it does not satisfy the second condition. The non-prime attributes Street, City, and State are transitively dependent on the candidate key through the non-prime attribute Zip. The table can then be split into two to satisfy 3NF. Table Employee can consist of ID, Name, and Zip, while the second table Employee Zip can consist of Street, City and State.

Boyce-Codd Normal Form

Boyce-Codd Normal Form is a simpler form of Third Normal Form, but stricter. It is an extension of 3NF where a relation must satisfy the following two conditions: it is in Third Normal Form and any existing dependencies (A → B) A cannot be a non-prime attribute and B is a prime-attribute. In other words, A being on the left side of the functional dependency is a primary key. BCNF does not allow any prime attributes to be dependent on non-prime attributes. A BCNF table is also in 3NF, but a 3NF table can't be in BCNF.

Let us make the table Product Order consist of attributes Product Order ID, Order Time, Payment No, Payment Type, and Payment Type Description. Candidate keys are Product ID, Payment No. Functional dependencies are Product Order ID to Order Time, as well as Payment No. to Payment Type and Payment Description. This is not in BCNF because neither Payment No and Product ID alone are keys. To convert the table into BCNF, tables must be decomposed for the left side of both the functional dependencies is a key.

Anomalies due to Poor Normalization

Insertion anomaly, update anomaly, and deletion anomaly are the three types of anomalies that can occur when a database is not normalized and poorly designed. There may exist tuples that contain redundant data and if we are to ever change that data, inconsistencies can happen. Normalization allows us to remove these anomalies and avoid running into these issues with our physical database.

Update Anomalies

Updating a table won't be possible if a table is not normalized. If we are to update one copy of such repeated data, an inconsistency is created unless all copies are similarly updated. If two copies of the same data do not match, wanting to update all copies we would be unable to decide which copy is correct. The data is said to become inconsistent. It essentially defeats the reason for one of the benefits of a database over a spreadsheet if we update one entry and copy this update to possibly many other fields.

For example, Employee information for the Employee table would have to be duplicated to avoid having the attribute phone number be a multi-value which would not satisfy 1NF. If we look into changing the address for a particular employee, this change would have to be applied to multiple records in the case where the employee has more than one phone number listed. If this update is only partially successful, the employee's address is updated on only some records but not others. The relation is left in an inconsistent state which provides conflicting answers to the question of what this particular employee's address is.

Insertion Anomalies

In a table that is not normalized, it won't be possible to insert new data due to existing dependencies in the table. We cannot store data unless some other information is stored well. This is an insert anomaly. If a tuple is inserted in referencing relation and referencing attribute value is not present in referenced attribute, it will not allow inserting in referencing relation.

For example, there is a table called Work History. It lists Employee ID, name, start date, job title code. It has the ability to record any employee that has at least one job. However, a newly hired employee’s work history cannot be recorded until they are assigned a job title except by setting the job title to null.

Deletion Anomalies

Deletion of data that represents certain facts may require deleting other data that may represent different facts. It won’t be possible to delete certain information without deleting other information too in a non-normalized table.

For instance, let’s recall the Work History table. If an employee temporarily stops being assigned to any work, the last of the records on which that employee appears must be deleted. This means not just deleting or setting the job title code to null but also deleting the employee’s ID, name, and start date in the case where the table was not normalized.

3.1.2 Normal Forms for Bakersfield Flower Shop

We will now be checking and documenting each of every of our relations to see if it at least satisfies Third Normal Form or Boyce-Codd Normal Form. This will help us determine any anomalies that may occur within each of our relations if they are not normalized. We will also be listing the original relation and updating those that were not in 3NF or in BCNF.

Customer

Functional Dependencies:

FD1: {customer_ID} → {fName, lName, street, city, state, zip, username, password, email, acc_creation_date, phone_number}

FD2: {username} → {password, email, acc_creation_date}

Normal Form**:**

1NF is satisfied because all attributes are single value, atomic domains.

2NF is satisfied because all non-prime attributes are not functionally dependent on any proper subset of any candidate.

3NF is satisfied because no non-prime attributes depend on other non-prime attributes.

BCNF is satisfied because the left side of all functional dependencies is a candidate key.

Since the BCNF is satisfied, there should be no modification anomalies.

Delivery Address

Functional Dependencies:

FD1: {address_id} → {city, street, state, zip}

Normal Form**:**

1NF is satisfied because all attributes are single value, atomic domains.

2NF is satisfied because the primary key consists of only a single attribute.

3NF is satisfied because no non-prime attributes depend on other non-prime attributes.

BCNF is satisfied because our dependency has the primary key on the left-hand side.

Since the BCNF is satisfied, there should be no modification anomalies.

Package

Functional Dependencies:

FD1: {package_id} → { expected_delivery_time, message}

Normal Form**:**

1NF is satisfied because all attributes are single value, atomic domains.

2NF is satisfied because the primary key consists of only a single attribute.

3NF is satisfied because no non-prime attributes depend on other non-prime attributes.

BCNF is satisfied because our dependency has the primary key on the left-hand side.

Since the BCNF is satisfied, there should be no modification anomalies.

Employee

Functional Dependencies:

FD1: {employee_id} → {fname, lname, street, city, state, zip, phone_number}

Normal Form**:**

1NF is satisfied because all attributes are single value, atomic domains.

2NF is satisfied because the primary key consists of only a single attribute.

3NF is satisfied because no non-prime attributes depend on other non-prime attributes.

BCNF is satisfied because our dependency has the primary key on the left-hand side.

Since the BCNF is satisfied, there should be no modification anomalies.

Flower Product

Functional Dependencies

FD1: {product_id} → {product_name, sell_price, purchase_price, color, length, product_image, description}

Normal Form**:**

1NF is satisfied because all attributes are single value, atomic domains.

2NF is satisfied because the primary key consists of only a single attribute.

3NF is satisfied because no non-prime attributes depend on other non-prime attributes.

BCNF is satisfied because our dependency has the primary key on the left-hand side.

Since the BCNF is satisfied, there should be no modification anomalies.

Incoming Payment

Functional Dependencies:

FD1: {incoming_id} → {sales_tax}

Normal Form**:**

1NF is satisfied because all attributes are single value, atomic domains.

2NF is satisfied because the primary key consists of only a single attribute.

3NF is satisfied because no non-prime attributes depend on other non-prime attributes.

BCNF is satisfied because our dependency has the primary key on the left-hand side.

Since the BCNF is satisfied, there should be no modification anomalies.

Order Status

Functional Dependencies:

FD1: {status_id} → {status}

Normal Form**:**

1NF is satisfied because all attributes are single value, atomic domains.

2NF is satisfied because the primary key consists of only a single attribute.

3NF is satisfied because no non-prime attributes depend on other non-prime attributes.

BCNF is satisfied because our dependency has the primary key on the left-hand side.

Since the BCNF is satisfied, there should be no modification anomalies.

Payment

Functional Dependencies:

FD1: {payment_id} → {amount, payment_time}

Normal Form**:**

1NF is satisfied because all attributes are single value, atomic domains.

2NF is satisfied because the primary key consists of only a single attribute.

3NF is satisfied because no non-prime attributes depend on other non-prime attributes.

BCNF is satisfied because our dependency has the primary key on the left-hand side.

Since the BCNF is satisfied, there should be no modification anomalies.

Payment Type

Functional Dependencies:

FD1: {description} → {payment_type_id}

Normal Form**:**

1NF is satisfied because all attributes are single value, atomic domains.

2NF is satisfied because the primary key consists of only a single attribute.

3NF is satisfied because no non-prime attributes depend on other non-prime attributes.

BCNF is satisfied because the left side of all dependencies are a candidate key.

Since the BCNF is satisfied, there should be no modification anomalies.

Product Order

Functional Dependencies:

FD1: {p_order_number} → {order_time}

Normal Form:

1NF is satisfied because all attributes are single value, atomic domains.

2NF is satisfied because the primary key consists of only a single attribute.

3NF is satisfied because no non-prime attributes depend on other non-prime attributes.

BCNF is satisfied because our dependency has the primary key on the left-hand side.

Since the BCNF is satisfied, there should be no modification anomalies.

Recipient

Functional Dependencies:

FD1: {recipient_id} → {fName, lName, phone_number}

Normal Form:

1NF is satisfied because all attributes are single value, atomic domains.

2NF is satisfied because the primary key consists of only a single attribute.

3NF is satisfied because no non-prime attributes depend on other non-prime attributes.

BCNF is satisfied because our dependency has the primary key on the left-hand side.

Since the BCNF is satisfied, there should be no modification anomalies.

Supplier

Functional Dependencies:

FD1: {supplier_id} → {vendor_name, street, city, state, zip, phone_number}

Normal Form**:**

1NF is satisfied because all attributes are single value, atomic domains.

2NF is satisfied because the primary key consists of only a single attribute.

3NF is satisfied because no non-prime attributes depend on other non-prime attributes.

BCNF is satisfied because our dependency has the primary key on the left-hand side.

Since the BCNF is satisfied, there should be no modification anomalies.

Outgoing Payment

Functional Dependencies:

FD1: {outgoing_id} → {supplier_invoice_id}

FD2: {supplier_invoice_id} → {outgoing_id}

Normal Form**:**

1NF is satisfied because all attributes are single value, atomic domains.

2NF is satisfied because the primary key consists of only a single attribute.

3NF is satisfied because no non-prime attributes depend on other non-prime attributes.

BCNF is satisfied because our dependency has the primary key on the left-hand side.

Since the BCNF is satisfied, there should be no modification anomalies.

Supply Purchase Order

Functional Dependencies:

FD1: {supply_purchase_id} → {supply_purchase_time}

Normal Form**:**

1NF is satisfied because all attributes are single value, atomic domains.

2NF is satisfied because the primary key consists of only a single attribute.

3NF is satisfied because no non-prime attributes depend on other non-prime attributes.

BCNF is satisfied because our dependency has the primary key on the left-hand side.

Since the BCNF is satisfied, there should be no modification anomalies.

Work History

Functional Dependencies:

FD1: {work_history} → {start_date, end_date, job_title, pay_rate}

Normal Form**:**

1NF is satisfied because all attributes are single value, atomic domains.

2NF is satisfied because the primary key consists of only a single attribute.

3NF is satisfied because no non-prime attributes depend on other non-prime attributes.

BCNF is satisfied because our dependency has the primary key on the left-hand side.

Since the BCNF is satisfied, there should be no modification anomalies.

Work Shift

Functional Dependencies:

FD1: {work_shift} → {shift_date, begin_time, end_time}

Normal Form**:**

1NF is satisfied because all attributes are single value, atomic domains.

2NF is satisfied because the primary key consists of only a single attribute.

3NF is satisfied because no non-prime attributes depend on other non-prime attributes.

BCNF is satisfied because our dependency has the primary key on the left-hand side.

Since the BCNF is satisfied, there should be no modification anomalies.

Contains

Functional Dependencies:

FD1: {p_order_number, product_id} → {quantity_item, point_of_sale_price}

Normal Form**:**

1NF is satisfied because all attributes are single value, atomic domains.

2NF is satisfied because all non-prime attributes are not functionally dependent on any proper subset of any candidate.

3NF is satisfied because no non-prime attributes depend on other non-prime attributes.

BCNF is satisfied because our dependency has the primary key on the left-hand side.

Since the BCNF is satisfied, there should be no modification anomalies.

Needs

Functional Dependencies:

FD1: {supplier_purchase_id} → {payment_id}

FD2: {payment_id} → {supply_purchase_id}

Normal Form:

1NF is satisfied because all attributes are single value, atomic domains.

2NF is satisfied because all non-prime attributes are not functionally dependent on any proper subset of any candidate.

3NF is satisfied because no non-prime attributes depend on other non-prime attributes.

BCNF is satisfied because our dependency has the primary key on the left-hand side.

Since the BCNF is satisfied, there should be no modification anomalies.

Refills

Functional Dependencies:

FD1: {supply_purchase_id, product_id} → {quantity_item, supply_price}

Normal Form**:**

1NF is satisfied because all attributes are single value, atomic domains.

2NF is satisfied because all non-prime attributes are not functionally dependent on any proper subset of any candidate.

3NF is satisfied because no non-prime attributes depend on other non-prime attributes.

BCNF is satisfied because our dependency has the primary key on the left-hand side.

Since the BCNF is satisfied, there should be no modification anomalies.

3.2 Postgres Main Purpose and Functionality

Given that we have analyzed and ensured our relations will not cause any modification anomalies, we can begin discussing the process of implementing our physical database. PostgreSQL is a general and object-relational database management system that supports SQL querying. Psql is a terminal-based front-end to PostgreSQL. It allows users to issue queries to PostgreSQL which will then output the query result. The overall main purpose of a database management system like PostgreSQL is to give it's users the tools and power to effectively and efficiently handle the flow of data, giving them control of how data can be inserted, deleted, and modified.

PostgreSQL contains multiple types of functionality within its language that have come from the SQL standards such as: Data Types, Data Integrity, Concurrency, Performance, Reliability, Disaster Recovery, Security, Extensibility, Internationalisation, Text Search, and various others. Additionally, it contains many commands that allow for inserting, updating, and deleting data. With these sorts of tools at the disposal of its users, it lets users create and efficiently maintain databases while being able to query and search through tons of information to solve problems.

3.3 Schema Objects Allowed in Postgres

This section outlines the different objects that are in the Postgres Database Management System. Some of the topics covered in this section are tables, views, and drops and insertion of data.

Tables

Tables are the main construct for representing data in a database. They are similar to the relation from the relational model but do not have a perfect equivalence. A database table is similar to the relation, a database row is similar to a relation tuple, and a database column is similar to a relations attribute. Tables can also theoretically contain duplicate rows, which is not allowed in a relations tuple.

Tables are used to store data in a database and for establishing relationships between data sets. A relationship between tables is established through primary key and foreign key references. The syntax to create a table is as follows:

```
CREATE TABLE table_name (
   column_name TYPE column_constraint,
   table_constraint table_constraint
);
```

<u>Tables in this database:</u>

Contains

Customer

Delivery Address

Employee

Flower Product

Incoming Payment

Needs

Order Status

Package

Payment

Payment Type

Recipient

Refills

Requires

Supplier

Outgoing Payment

Supply Purchase Order

Work History

Work Shift

Insert

To add tuples to our table, we will use the INSERT command. For example, we will insert a new employee into our database. To insert into a specific table we must use the command “INSERT INTO” followed by the table name, followed by “VALUES(attribute1, attribute2, …);”

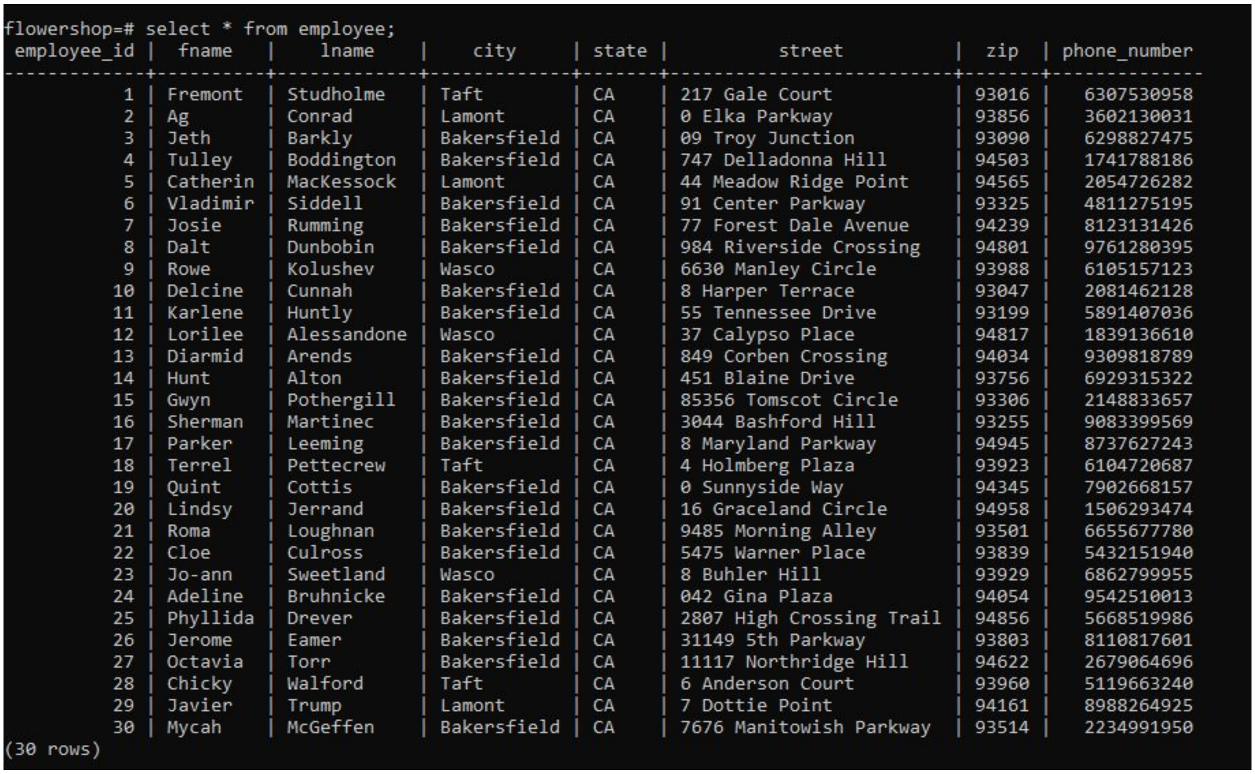
Ex:

```
INSERT INTO employee VALUES(
‘John’, ‘Doe’, ‘Bakersfield’, ‘CA’, ‘123 Main St.’, 93301, ‘JDoe1’, ‘rAnDoMpAsS’,
‘JDoe@mail.com’, to_date(‘04/12/2020’,’MM/DD/YYYY’), phone_number);
```

Select

Now if we want to see what tuples are contained within our table, we can use the “SELECT” command. To view all the tuples we can use “SELECT * FROM <tablename>”. This will select all the tuples from the specific table.

Ex:



```
flowershop=# select * from employee;
 employee_id |  fname   |    lname    |    city     | state |          street          |  zip  | phone_number
-------------+----------+-------------+-------------+-------+--------------------------+-------+--------------
           1 | Fremont  | Studholme   | Taft        | CA    | 217 Gale Court           | 93016 |   6307530958
           2 | Ag       | Conrad      | Lamont      | CA    | 0 Elka Parkway           | 93856 |   3602130031
           3 | Jeth     | Barkly      | Bakersfield | CA    | 09 Troy Junction         | 93090 |   6298827475
           4 | Tulley   | Boddington  | Bakersfield | CA    | 747 Delladonna Hill      | 94503 |   1741788186
           5 | Catherin | MacKessock  | Lamont      | CA    | 44 Meadow Ridge Point    | 94565 |   2054726282
           6 | Vladimir | Siddell     | Bakersfield | CA    | 91 Center Parkway        | 93325 |   4811275195
           7 | Josie    | Rumming     | Bakersfield | CA    | 77 Forest Dale Avenue    | 94239 |   8123131426
           8 | Dalt     | Dunbobin    | Bakersfield | CA    | 984 Riverside Crossing   | 94801 |   9761280395
           9 | Rowe     | Kolushev    | Wasco       | CA    | 6630 Manley Circle       | 93988 |   6105157123
          10 | Delcine  | Cunnah      | Bakersfield | CA    | 8 Harper Terrace         | 93047 |   2081462128
          11 | Karlene  | Huntly      | Bakersfield | CA    | 55 Tennessee Drive       | 93199 |   5891407036
          12 | Lorilee  | Alessandone | Wasco       | CA    | 37 Calypso Place         | 94817 |   1839136610
          13 | Diarmid  | Arends      | Bakersfield | CA    | 849 Corben Crossing      | 94034 |   9309818789
          14 | Hunt     | Alton       | Bakersfield | CA    | 451 Blaine Drive         | 93756 |   6929315322
          15 | Gwyn     | Pothergill  | Bakersfield | CA    | 85356 Tomscot Circle     | 93306 |   2148833657
          16 | Sherman  | Martinec    | Bakersfield | CA    | 3044 Bashford Hill       | 93255 |   9083399569
          17 | Parker   | Leeming     | Bakersfield | CA    | 8 Maryland Parkway       | 94945 |   8737627243
          18 | Terrel   | Pettecrew   | Taft        | CA    | 4 Holmberg Plaza         | 93923 |   6104720687
          19 | Quint    | Cottis      | Bakersfield | CA    | 0 Sunnyside Way          | 94345 |   7902668157
          20 | Lindsy   | Jerrand     | Bakersfield | CA    | 16 Graceland Circle      | 94958 |   1506293474
          21 | Roma     | Loughnan    | Bakersfield | CA    | 9485 Morning Alley       | 93501 |   6655677780
          22 | Cloe     | Culross     | Bakersfield | CA    | 5475 Warner Place        | 93839 |   5432151940
          23 | Jo-ann   | Sweetland   | Wasco       | CA    | 8 Buhler Hill            | 93929 |   6862799955
          24 | Adeline  | Bruhnicke   | Bakersfield | CA    | 042 Gina Plaza           | 94054 |   9542510013
          25 | Phyllida | Drever      | Bakersfield | CA    | 2807 High Crossing Trail | 94856 |   5668519986
          26 | Jerome   | Eamer       | Bakersfield | CA    | 31149 5th Parkway        | 93803 |   8110817601
          27 | Octavia  | Torr        | Bakersfield | CA    | 11117 Northridge Hill    | 94622 |   2679064696
          28 | Chicky   | Walford     | Taft        | CA    | 6 Anderson Court         | 93960 |   5119663240
          29 | Javier   | Trump       | Lamont      | CA    | 7 Dottie Point           | 94161 |   8988264925
          30 | Mycah    | McGeffen    | Bakersfield | CA    | 7676 Manitowish Parkway  | 93514 |   2234991950
(30 rows)
```

Views

In a database a view is the result of a stored query. Views act similar to tables in that they can be queried and information can be gathered from them. Except for the case of materialized view they do not store information but instead are a representation of data from underlying tables.

Views are useful for simplifying complex queries, adding security to databases, and providing faster access to data than directly querying the tables that built the view. The syntax for creating a view is as follows:

```
CREATE OR REPLACE VIEW name_of_view
AS
[selectStatement or query];
```

Stored Procedures

Stored procedures are subroutines for manipulating data and help in carrying out business logic in a database. Stored procedures and functions are similar, but stored procedures can perform their logic without returning any data.

Stored procedures allow an application to perform more efficiently if done correctly, as more can be accomplished in one call to the procedure than executing a series of queries from the application. The syntax to create a stored procedure is as follows:

```
CREATE [OR REPLACE] PROCEDURE procedure_name(parameter_list)
AS $varName$
    stored_procedure_body;
$varName$;
```

Functions

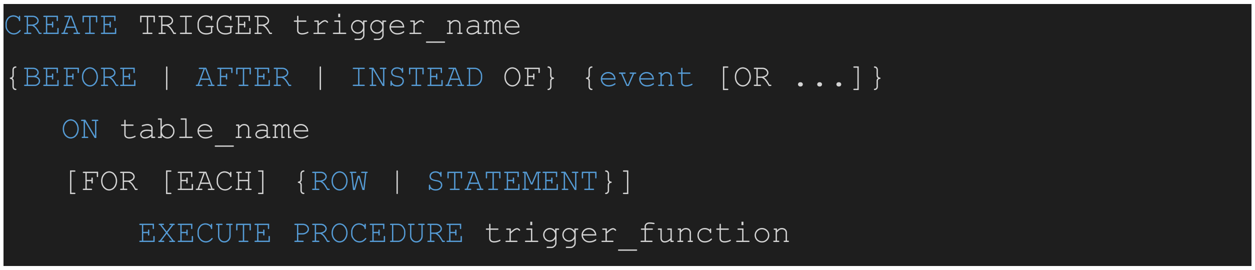
In postgreSQL there are user defined functions and built in functions. Built in functions include aggregate functions, date functions, string manipulation, and window functions. These are common functions between most database applications so they are included in postgreSQL.

User defined functions are similar to stored procedures, as they are subroutines in a database. The key difference between a stored procedure and a user defined function is a user defined function must return data. The syntax for a user defined function is as follows:

```
CREATE FUNCTION function_name(p1 type, p2 type)
 RETURNS type AS
BEGIN
 -- logic
END;
```

Trigger

A Trigger in postgreSQL is a user-defined function invoked automatically when an event occurs involving a table in a database. Triggers help maintain consistency between data that is related to each other. If a record in one table depends on the value in another table a trigger can automatically update the data involved in it’s defined routine. The syntax for a trigger is as follows:

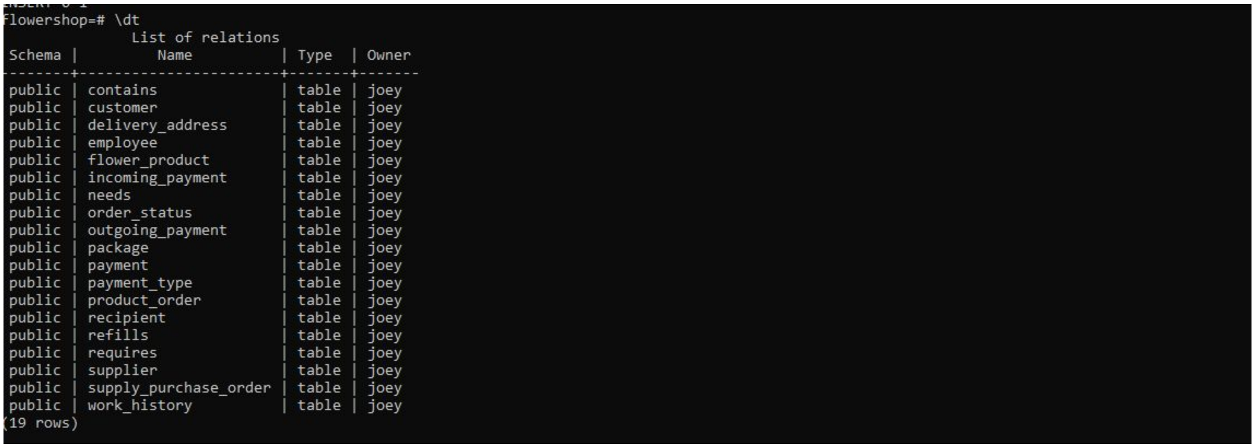


```
CREATE TRIGGER trigger_name
{BEFORE | AFTER | INSTEAD OF} {event [OR ...]}
   ON table_name
   [FOR [EACH] {ROW | STATEMENT}]
       EXECUTE PROCEDURE trigger_function
```

3.4 Displaying Relations with SQL Commands

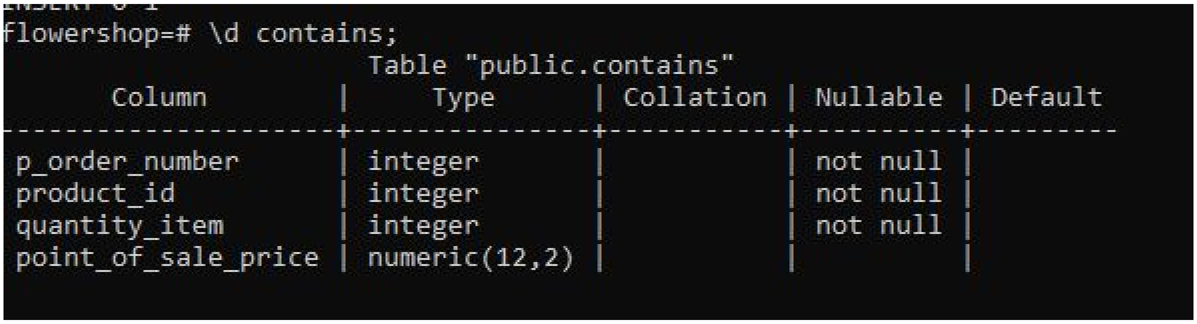
In this section, SQL commands will be used to display every relation and its data within the physical database. Command “\dt” will display a list of all tables within our database. Command “\d [table name]” will display every column, their types, tablespace, defaults, as well as any indexes, constraints, rules and triggers of that said table. Command “SELECT * FROM [table_name]” obtains rows from specified tables and retrieves all available fields in the table.

All Tables

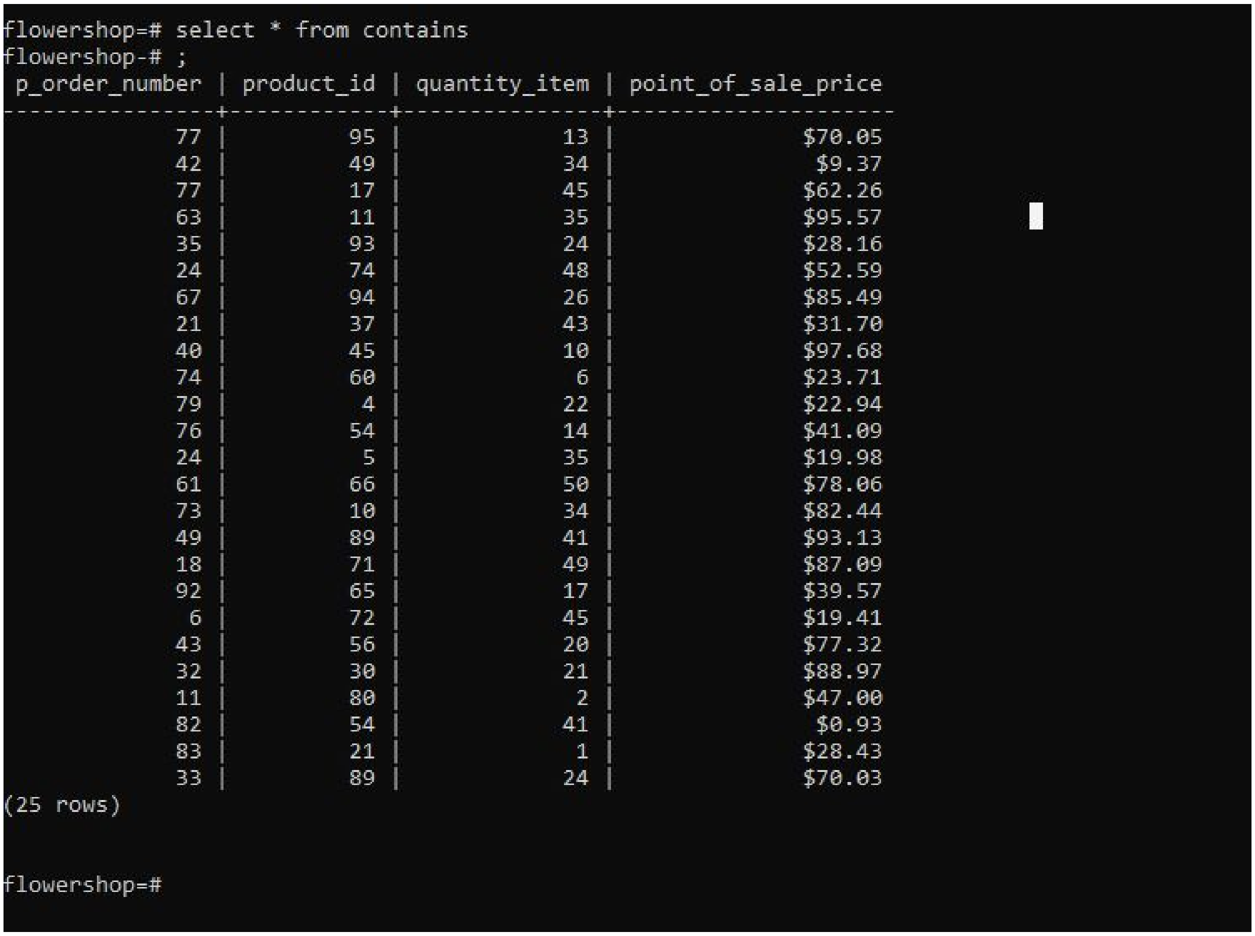


```
INSERT 0 1
flowershop=# \dt
               List of relations
 Schema |         Name          | Type  | Owner
--------+-----------------------+-------+-------
 public | contains              | table | joey
 public | customer              | table | joey
 public | delivery_address      | table | joey
 public | employee              | table | joey
 public | flower_product        | table | joey
 public | incoming_payment      | table | joey
 public | needs                 | table | joey
 public | order_status          | table | joey
 public | outgoing_payment      | table | joey
 public | package               | table | joey
 public | payment               | table | joey
 public | payment_type          | table | joey
 public | product_order         | table | joey
 public | recipient             | table | joey
 public | refills               | table | joey
 public | requires              | table | joey
 public | supplier              | table | joey
 public | supply_purchase_order | table | joey
 public | work_history          | table | joey
(19 rows)
```

Contains

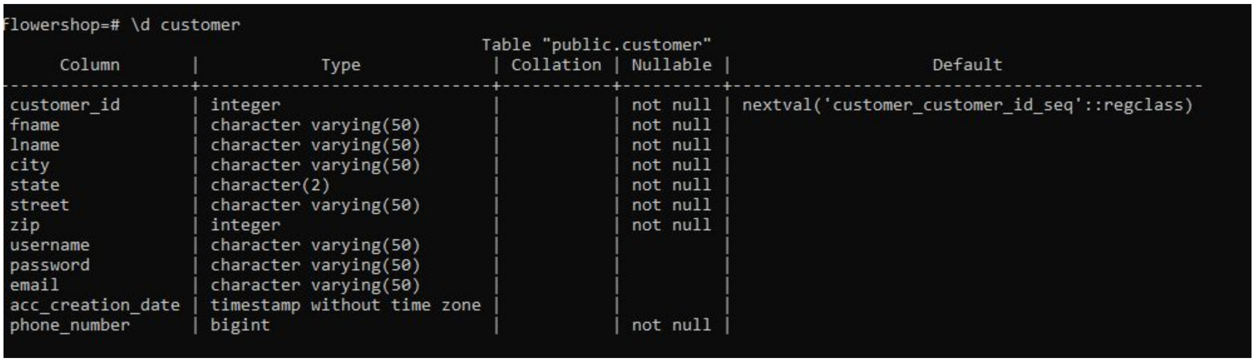


```
INSERT 0 1
flowershop=# \d contains;
                       Table "public.contains"
       Column        |     Type      | Collation | Nullable | Default
---------------------+---------------+-----------+----------+---------
 p_order_number      | integer       |           | not null |
 product_id          | integer       |           | not null |
 quantity_item       | integer       |           | not null |
 point_of_sale_price | numeric(12,2) |           |          |
```

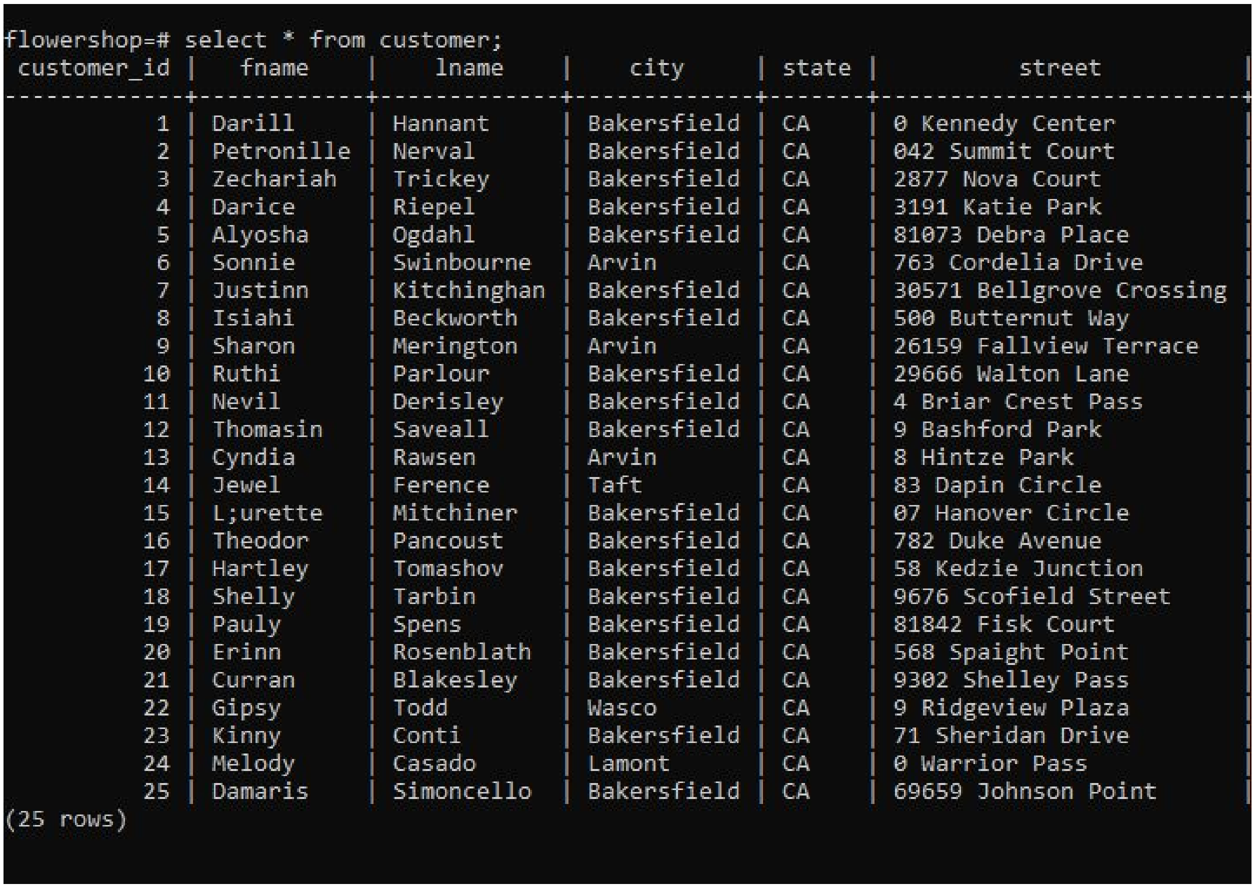


```
flowershop=# select * from contains
flowershop-# ;
 p_order_number | product_id | quantity_item | point_of_sale_price
----------------+------------+---------------+---------------------
             77 |         95 |            13 |              $70.05
             42 |         49 |            34 |               $9.37
             77 |         17 |            45 |              $62.26
             63 |         11 |            35 |              $95.57
             35 |         93 |            24 |              $28.16
             24 |         74 |            48 |              $52.59
             67 |         94 |            26 |              $85.49
             21 |         37 |            43 |              $31.70
             40 |         45 |            10 |              $97.68
             74 |         60 |             6 |              $23.71
             79 |          4 |            22 |              $22.94
             76 |         54 |            14 |              $41.09
             24 |          5 |            35 |              $19.98
             61 |         66 |            50 |              $78.06
             73 |         10 |            34 |              $82.44
             49 |         89 |            41 |              $93.13
             18 |         71 |            49 |              $87.09
             92 |         65 |            17 |              $39.57
              6 |         72 |            45 |              $19.41
             43 |         56 |            20 |              $77.32
             32 |         30 |            21 |              $88.97
             11 |         80 |             2 |              $47.00
             82 |         54 |            41 |               $0.93
             83 |         21 |             1 |              $28.43
             33 |         89 |            24 |              $70.03
(25 rows)


flowershop=#
```

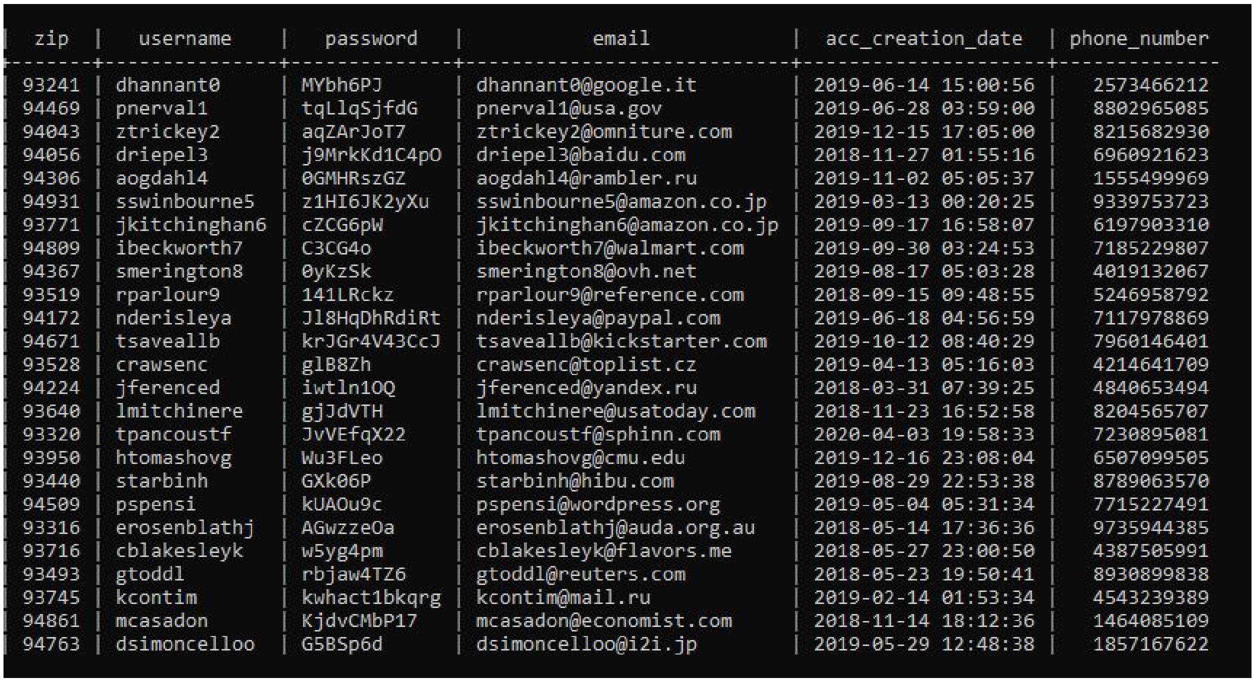
Customer



```
flowershop=# \d customer
                                       Table "public.customer"
      Column       |            Type             | Collation | Nullable |                    Default
-------------------+-----------------------------+-----------+----------+-----------------------------------------------
 customer_id       | integer                     |           | not null | nextval('customer_customer_id_seq'::regclass)
 fname             | character varying(50)       |           | not null |
 lname             | character varying(50)       |           | not null |
 city              | character varying(50)       |           | not null |
 state             | character(2)                |           | not null |
 street            | character varying(50)       |           | not null |
 zip               | integer                     |           | not null |
 username          | character varying(50)       |           |          |
 password          | character varying(50)       |           |          |
 email             | character varying(50)       |           |          |
 acc_creation_date | timestamp without time zone |           |          |
 phone_number      | bigint                      |           | not null |
```

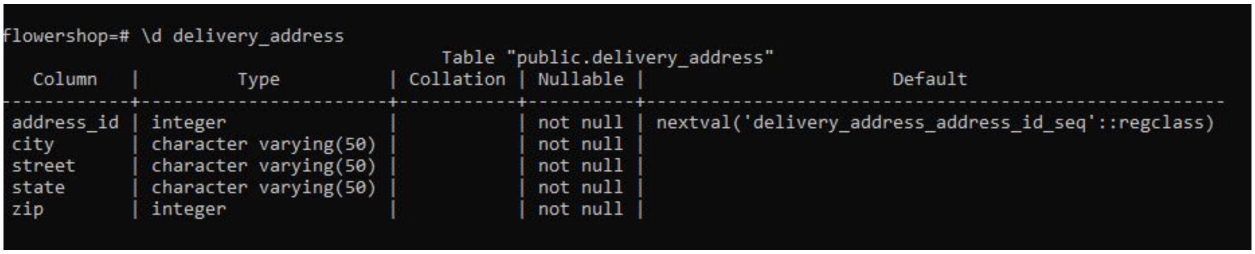


```
flowershop=# select * from customer;
 customer_id |   fname    |   lname    |    city     | state |         street          |
-------------+------------+------------+-------------+-------+-------------------------+
           1 | Darill     | Hannant    | Bakersfield | CA    | 0 Kennedy Center        |
           2 | Petronille | Nerval     | Bakersfield | CA    | 042 Summit Court        |
           3 | Zechariah  | Trickey    | Bakersfield | CA    | 2877 Nova Court         |
           4 | Darice     | Riepel     | Bakersfield | CA    | 3191 Katie Park         |
           5 | Alyosha    | Ogdahl     | Bakersfield | CA    | 81073 Debra Place       |
           6 | Sonnie     | Swinbourne | Arvin       | CA    | 763 Cordelia Drive      |
           7 | Justinn    | Kitchinghan| Bakersfield | CA    | 30571 Bellgrove Crossing|
           8 | Isiahi     | Beckworth  | Bakersfield | CA    | 500 Butternut Way       |
           9 | Sharon     | Merington  | Arvin       | CA    | 26159 Fallview Terrace  |
          10 | Ruthi      | Parlour    | Bakersfield | CA    | 29666 Walton Lane       |
          11 | Nevil      | Derisley   | Bakersfield | CA    | 4 Briar Crest Pass      |
          12 | Thomasin   | Saveall    | Bakersfield | CA    | 9 Bashford Park         |
          13 | Cyndia     | Rawsen     | Arvin       | CA    | 8 Hintze Park           |
          14 | Jewel      | Ference    | Taft        | CA    | 83 Dapin Circle         |
          15 | L;urette   | Mitchiner  | Bakersfield | CA    | 07 Hanover Circle       |
          16 | Theodor    | Pancoust   | Bakersfield | CA    | 782 Duke Avenue         |
          17 | Hartley    | Tomashov   | Bakersfield | CA    | 58 Kedzie Junction      |
          18 | Shelly     | Tarbin     | Bakersfield | CA    | 9676 Scofield Street    |
          19 | Pauly      | Spens      | Bakersfield | CA    | 81842 Fisk Court        |
          20 | Erinn      | Rosenblath | Bakersfield | CA    | 568 Spaight Point       |
          21 | Curran     | Blakesley  | Bakersfield | CA    | 9302 Shelley Pass       |
          22 | Gipsy      | Todd       | Wasco       | CA    | 9 Ridgeview Plaza       |
          23 | Kinny      | Conti      | Bakersfield | CA    | 71 Sheridan Drive       |
          24 | Melody     | Casado     | Lamont      | CA    | 0 Warrior Pass          |
          25 | Damaris    | Simoncello | Bakersfield | CA    | 69659 Johnson Point     |
(25 rows)
```

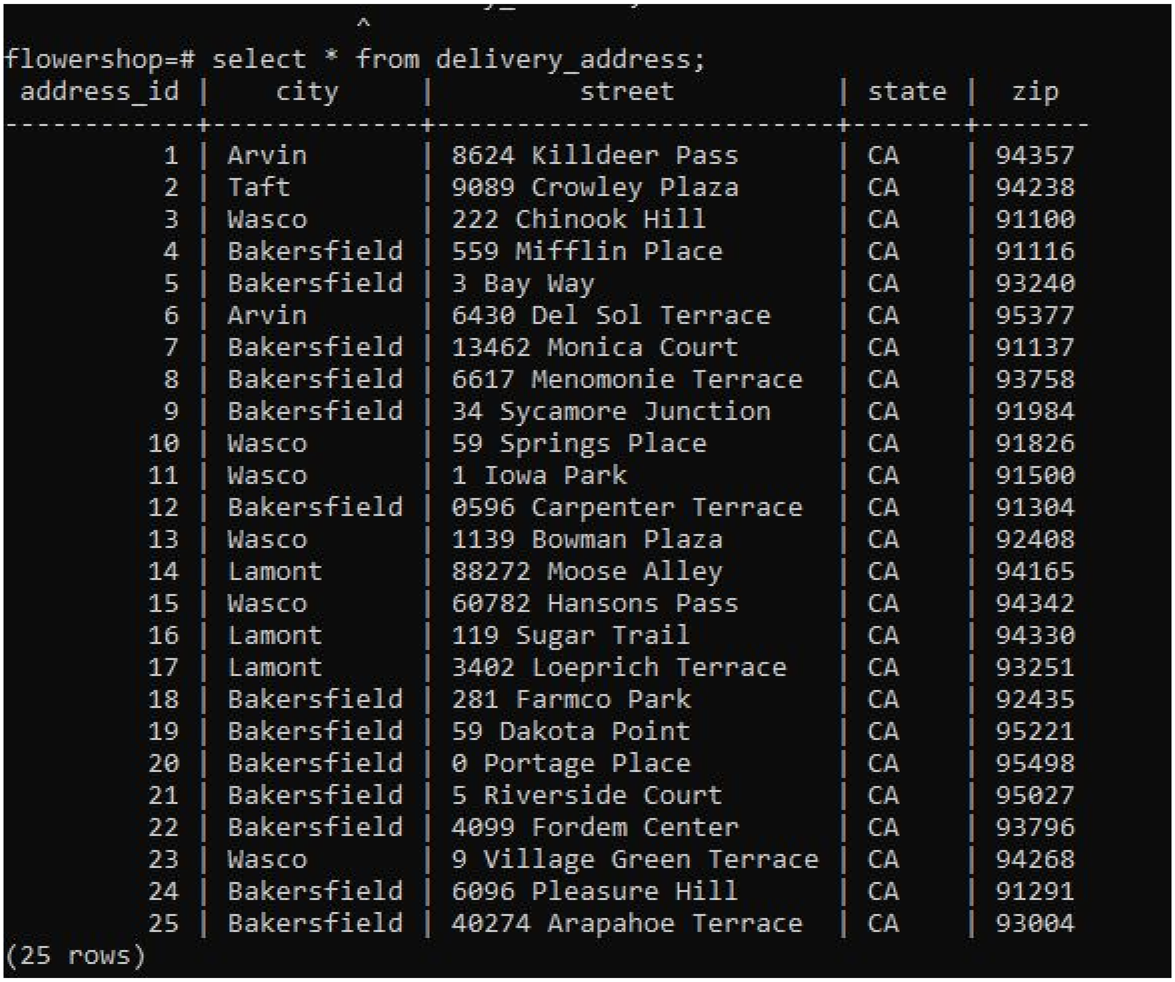


```
|  zip  |    username    |   password    |            email             |  acc_creation_date  | phone_number
+-------+----------------+---------------+------------------------------+---------------------+--------------
| 93241 | dhannant0      | MYbh6PJ       | dhannant0@google.it          | 2019-06-14 15:00:56 |   2573466212
| 94469 | pnerval1       | tqLlqSjfdG    | pnerval1@usa.gov             | 2019-06-28 03:59:00 |   8802965085
| 94043 | ztrickey2      | aqZArJoT7     | ztrickey2@omniture.com       | 2019-12-15 17:05:00 |   8215682930
| 94056 | driepel3       | j9MrkKd1C4pO  | driepel3@baidu.com           | 2018-11-27 01:55:16 |   6960921623
| 94306 | aogdahl4       | 0GMHRszGZ     | aogdahl4@rambler.ru          | 2019-11-02 05:05:37 |   1555499969
| 94931 | sswinbourne5   | z1HI6JK2yXu   | sswinbourne5@amazon.co.jp    | 2019-03-13 00:20:25 |   9339753723
| 93771 | jkitchinghan6  | cZCG6pW       | jkitchinghan6@amazon.co.jp   | 2019-09-17 16:58:07 |   6197903310
| 94809 | ibeckworth7    | C3CG4o        | ibeckworth7@walmart.com      | 2019-09-30 03:24:53 |   7185229807
| 94367 | smerington8    | 0yKzSk        | smerington8@ovh.net          | 2019-08-17 05:03:28 |   4019132067
| 93519 | rparlour9      | 141LRckz      | rparlour9@reference.com      | 2018-09-15 09:48:55 |   5246958792
| 94172 | nderisleya     | Jl8HqDhRdiRt  | nderisleya@paypal.com        | 2019-06-18 04:56:59 |   7117978869
| 94671 | tsaveallb      | krJGr4V43CcJ  | tsaveallb@kickstarter.com    | 2019-10-12 08:40:29 |   7960146401
| 93528 | crawsenc       | glB8Zh        | crawsenc@toplist.cz          | 2019-04-13 05:16:03 |   4214641709
| 94224 | jferenced      | iwtln1OQ      | jferenced@yandex.ru          | 2018-03-31 07:39:25 |   4840653494
| 93640 | lmitchinere    | gjJdVTH       | lmitchinere@usatoday.com     | 2018-11-23 16:52:58 |   8204565707
| 93320 | tpancoustf     | JvVEfqX22     | tpancoustf@sphinn.com        | 2020-04-03 19:58:33 |   7230895081
| 93950 | htomashovg     | Wu3FLeo       | htomashovg@cmu.edu           | 2019-12-16 23:08:04 |   6507099505
| 93440 | starbinh       | GXk06P        | starbinh@hibu.com            | 2019-08-29 22:53:38 |   8789063570
| 94509 | pspensi        | kUAOu9c       | pspensi@wordpress.org        | 2019-05-04 05:31:34 |   7715227491
| 93316 | erosenblathj   | AGwzzeOa      | erosenblathj@auda.org.au     | 2018-05-14 17:36:36 |   9735944385
| 93716 | cblakesleyk    | w5yg4pm       | cblakesleyk@flavors.me       | 2018-05-27 23:00:50 |   4387505991
| 93493 | gtoddl         | rbjaw4TZ6     | gtoddl@reuters.com           | 2018-05-23 19:50:41 |   8930899838
| 93745 | kcontim        | kwhact1bkqrg  | kcontim@mail.ru              | 2019-02-14 01:53:34 |   4543239389
| 94861 | mcasadon       | KjdvCMbP17    | mcasadon@economist.com       | 2018-11-14 18:12:36 |   1464085109
| 94763 | dsimoncelloo   | G5BSp6d       | dsimoncelloo@i2i.jp          | 2019-05-29 12:48:38 |   1857167622
```

Delivery Address

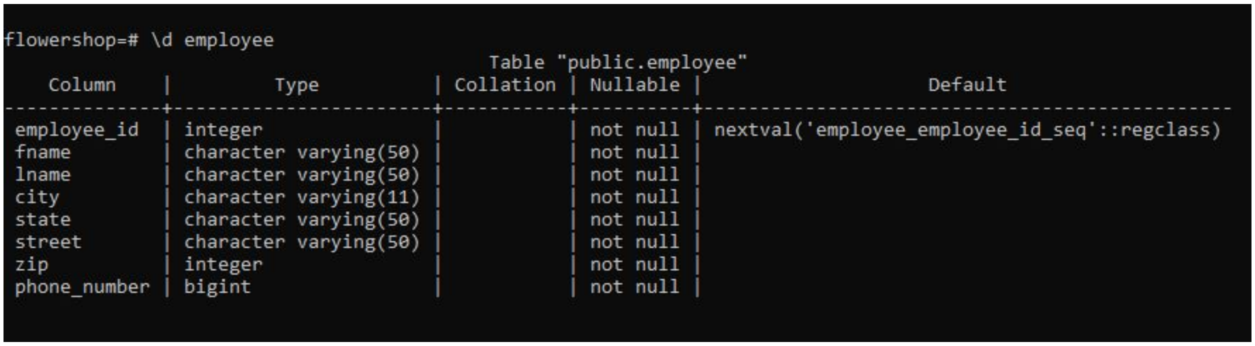


```
flowershop=# \d delivery_address
                                  Table "public.delivery_address"
  Column    |          Type          | Collation | Nullable |                         Default
------------+------------------------+-----------+----------+---------------------------------------------------------
 address_id | integer                |           | not null | nextval('delivery_address_address_id_seq'::regclass)
 city       | character varying(50)  |           | not null |
 street     | character varying(50)  |           | not null |
 state      | character varying(50)  |           | not null |
 zip        | integer                |           | not null |
```

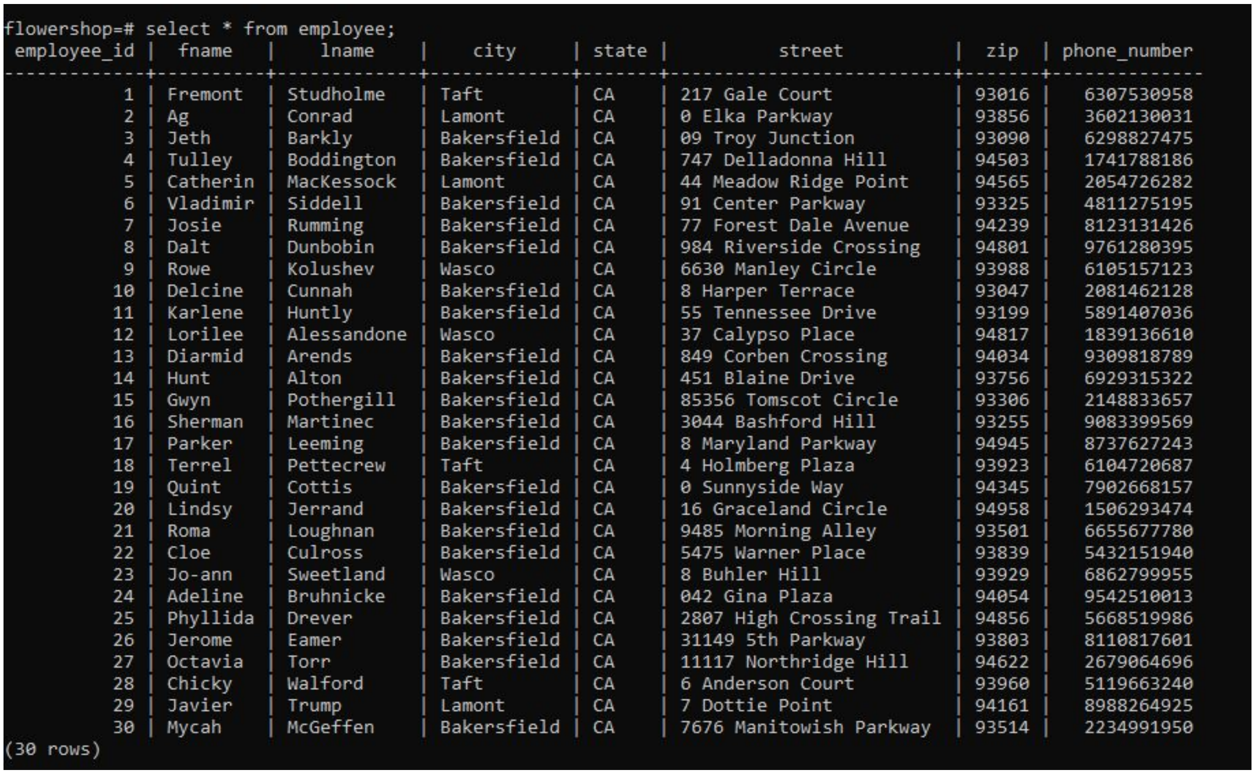


```
                      ^
flowershop=# select * from delivery_address;
 address_id |    city     |          street           | state |  zip
------------+-------------+---------------------------+-------+-------
          1 | Arvin       | 8624 Killdeer Pass        | CA    | 94357
          2 | Taft        | 9089 Crowley Plaza        | CA    | 94238
          3 | Wasco       | 222 Chinook Hill          | CA    | 91100
          4 | Bakersfield | 559 Mifflin Place         | CA    | 91116
          5 | Bakersfield | 3 Bay Way                 | CA    | 93240
          6 | Arvin       | 6430 Del Sol Terrace      | CA    | 95377
          7 | Bakersfield | 13462 Monica Court        | CA    | 91137
          8 | Bakersfield | 6617 Menomonie Terrace    | CA    | 93758
          9 | Bakersfield | 34 Sycamore Junction      | CA    | 91984
         10 | Wasco       | 59 Springs Place          | CA    | 91826
         11 | Wasco       | 1 Iowa Park               | CA    | 91500
         12 | Bakersfield | 0596 Carpenter Terrace    | CA    | 91304
         13 | Wasco       | 1139 Bowman Plaza         | CA    | 92408
         14 | Lamont      | 88272 Moose Alley         | CA    | 94165
         15 | Wasco       | 60782 Hansons Pass        | CA    | 94342
         16 | Lamont      | 119 Sugar Trail           | CA    | 94330
         17 | Lamont      | 3402 Loeprich Terrace     | CA    | 93251
         18 | Bakersfield | 281 Farmco Park           | CA    | 92435
         19 | Bakersfield | 59 Dakota Point           | CA    | 95221
         20 | Bakersfield | 0 Portage Place           | CA    | 95498
         21 | Bakersfield | 5 Riverside Court         | CA    | 95027
         22 | Bakersfield | 4099 Fordem Center        | CA    | 93796
         23 | Wasco       | 9 Village Green Terrace   | CA    | 94268
         24 | Bakersfield | 6096 Pleasure Hill        | CA    | 91291
         25 | Bakersfield | 40274 Arapahoe Terrace    | CA    | 93004
(25 rows)
```

Employee

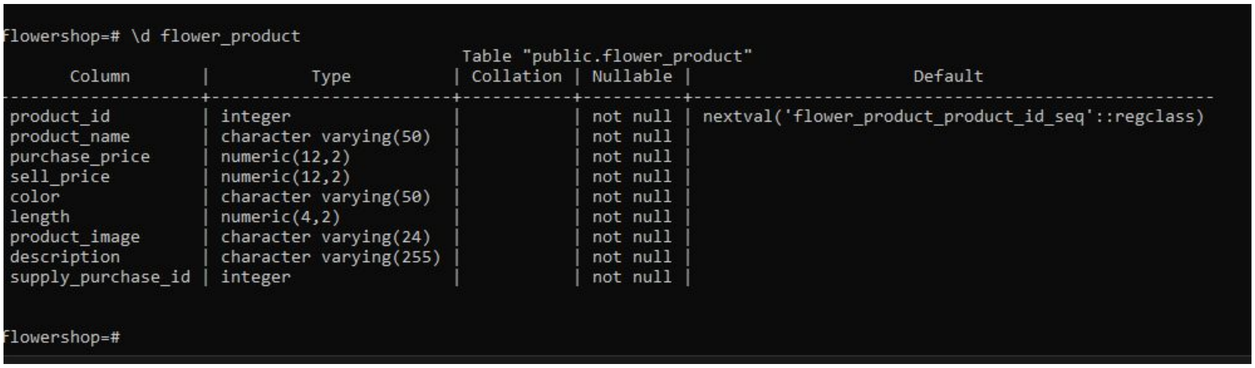


```
flowershop=# \d employee
                                      Table "public.employee"
    Column    |         Type          | Collation | Nullable |                    Default
--------------+-----------------------+-----------+----------+-----------------------------------------------
 employee_id  | integer               |           | not null | nextval('employee_employee_id_seq'::regclass)
 fname        | character varying(50) |           | not null |
 lname        | character varying(50) |           | not null |
 city         | character varying(11) |           | not null |
 state        | character varying(50) |           | not null |
 street       | character varying(50) |           | not null |
 zip          | integer               |           | not null |
 phone_number | bigint                |           | not null |
```

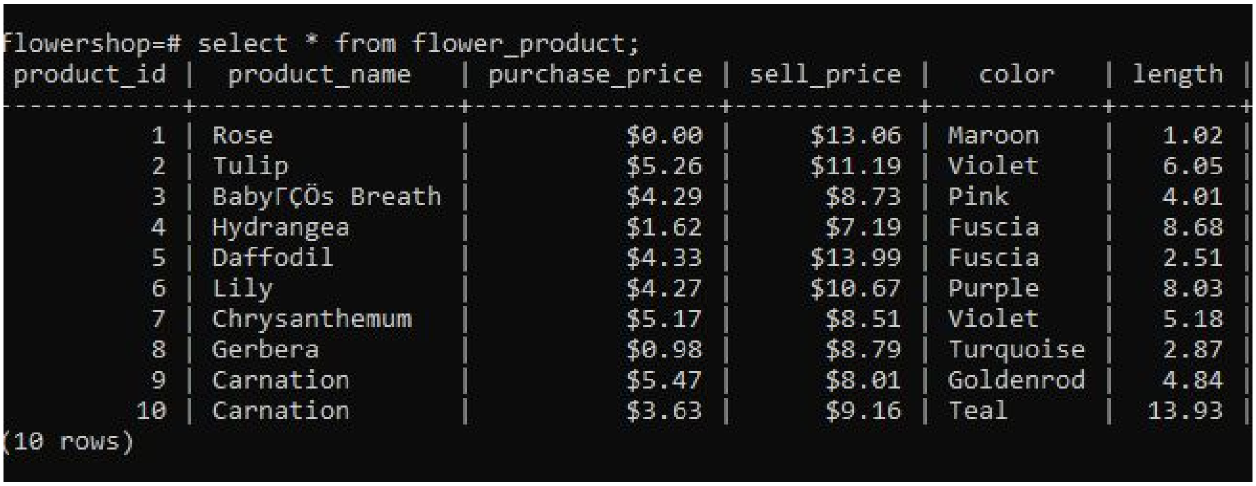


```
flowershop=# select * from employee;
 employee_id |  fname   |    lname    |    city     | state |          street          |  zip  | phone_number
-------------+----------+-------------+-------------+-------+--------------------------+-------+--------------
           1 | Fremont  | Studholme   | Taft        | CA    | 217 Gale Court           | 93016 |   6307530958
           2 | Ag       | Conrad      | Lamont      | CA    | 0 Elka Parkway           | 93856 |   3602130031
           3 | Jeth     | Barkly      | Bakersfield | CA    | 09 Troy Junction         | 93090 |   6298827475
           4 | Tulley   | Boddington  | Bakersfield | CA    | 747 Delladonna Hill      | 94503 |   1741788186
           5 | Catherin | MacKessock  | Lamont      | CA    | 44 Meadow Ridge Point    | 94565 |   2054726282
           6 | Vladimir | Siddell     | Bakersfield | CA    | 91 Center Parkway        | 93325 |   4811275195
           7 | Josie    | Rumming     | Bakersfield | CA    | 77 Forest Dale Avenue    | 94239 |   8123131426
           8 | Dalt     | Dunbobin    | Bakersfield | CA    | 984 Riverside Crossing   | 94801 |   9761280395
           9 | Rowe     | Kolushev    | Wasco       | CA    | 6630 Manley Circle       | 93988 |   6105157123
          10 | Delcine  | Cunnah      | Bakersfield | CA    | 8 Harper Terrace         | 93047 |   2081462128
          11 | Karlene  | Huntly      | Bakersfield | CA    | 55 Tennessee Drive       | 93199 |   5891407036
          12 | Lorilee  | Alessandone | Wasco       | CA    | 37 Calypso Place         | 94817 |   1839136610
          13 | Diarmid  | Arends      | Bakersfield | CA    | 849 Corben Crossing      | 94034 |   9309818789
          14 | Hunt     | Alton       | Bakersfield | CA    | 451 Blaine Drive         | 93756 |   6929315322
          15 | Gwyn     | Pothergill  | Bakersfield | CA    | 85356 Tomscot Circle     | 93306 |   2148833657
          16 | Sherman  | Martinec    | Bakersfield | CA    | 3044 Bashford Hill       | 93255 |   9083399569
          17 | Parker   | Leeming     | Bakersfield | CA    | 8 Maryland Parkway       | 94945 |   8737627243
          18 | Terrel   | Pettecrew   | Taft        | CA    | 4 Holmberg Plaza         | 93923 |   6104720687
          19 | Quint    | Cottis      | Bakersfield | CA    | 0 Sunnyside Way          | 94345 |   7902668157
          20 | Lindsy   | Jerrand     | Bakersfield | CA    | 16 Graceland Circle      | 94958 |   1506293474
          21 | Roma     | Loughnan    | Bakersfield | CA    | 9485 Morning Alley       | 93501 |   6655677780
          22 | Cloe     | Culross     | Bakersfield | CA    | 5475 Warner Place        | 93839 |   5432151940
          23 | Jo-ann   | Sweetland   | Wasco       | CA    | 8 Buhler Hill            | 93929 |   6862799955
          24 | Adeline  | Bruhnicke   | Bakersfield | CA    | 042 Gina Plaza           | 94054 |   9542510013
          25 | Phyllida | Drever      | Bakersfield | CA    | 2807 High Crossing Trail | 94856 |   5668519986
          26 | Jerome   | Eamer       | Bakersfield | CA    | 31149 5th Parkway        | 93803 |   8110817601
          27 | Octavia  | Torr        | Bakersfield | CA    | 11117 Northridge Hill    | 94622 |   2679064696
          28 | Chicky   | Walford     | Taft        | CA    | 6 Anderson Court         | 93960 |   5119663240
          29 | Javier   | Trump       | Lamont      | CA    | 7 Dottie Point           | 94161 |   8988264925
          30 | Mycah    | McGeffen    | Bakersfield | CA    | 7676 Manitowish Parkway  | 93514 |   2234991950
(30 rows)
```

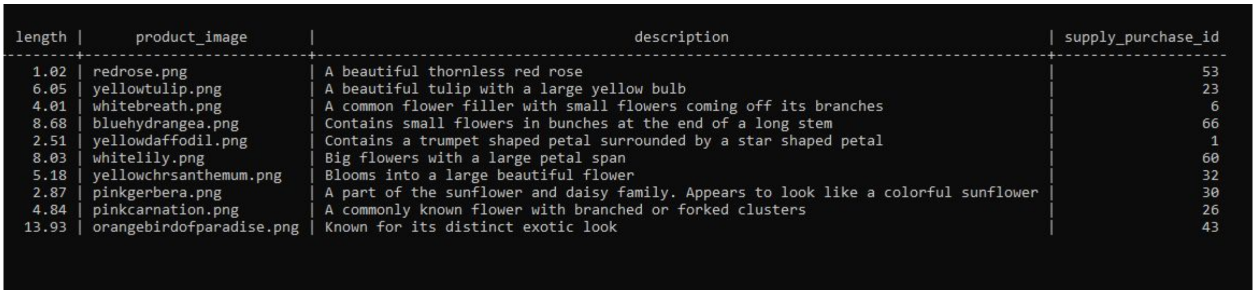
Flower Product



```
flowershop=# \d flower_product
                                        Table "public.flower_product"
       Column       |          Type          | Collation | Nullable |                      Default
--------------------+------------------------+-----------+----------+----------------------------------------------------
 product_id         | integer                |           | not null | nextval('flower_product_product_id_seq'::regclass)
 product_name       | character varying(50)  |           | not null |
 purchase_price     | numeric(12,2)          |           | not null |
 sell_price         | numeric(12,2)          |           | not null |
 color              | character varying(50)  |           | not null |
 length             | numeric(4,2)           |           | not null |
 product_image      | character varying(24)  |           | not null |
 description        | character varying(255) |           | not null |
 supply_purchase_id | integer                |           | not null |


flowershop=#
```

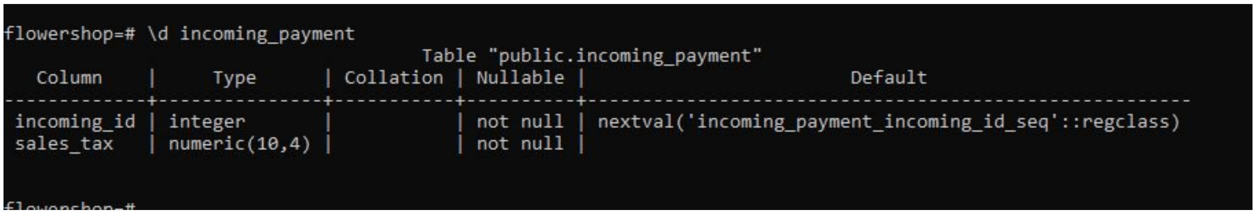


```
flowershop=# select * from flower_product;
 product_id |  product_name    | purchase_price | sell_price |   color    | length |
------------+------------------+----------------+------------+------------+--------+
          1 | Rose             |          $0.00 |     $13.06 | Maroon     |   1.02 |
          2 | Tulip            |          $5.26 |     $11.19 | Violet     |   6.05 |
          3 | BabyΓÇÖs Breath  |          $4.29 |      $8.73 | Pink       |   4.01 |
          4 | Hydrangea        |          $1.62 |      $7.19 | Fuscia     |   8.68 |
          5 | Daffodil         |          $4.33 |     $13.99 | Fuscia     |   2.51 |
          6 | Lily             |          $4.27 |     $10.67 | Purple     |   8.03 |
          7 | Chrysanthemum    |          $5.17 |      $8.51 | Violet     |   5.18 |
          8 | Gerbera          |          $0.98 |      $8.79 | Turquoise  |   2.87 |
          9 | Carnation        |          $5.47 |      $8.01 | Goldenrod  |   4.84 |
         10 | Carnation        |          $3.63 |      $9.16 | Teal       |  13.93 |
(10 rows)
```

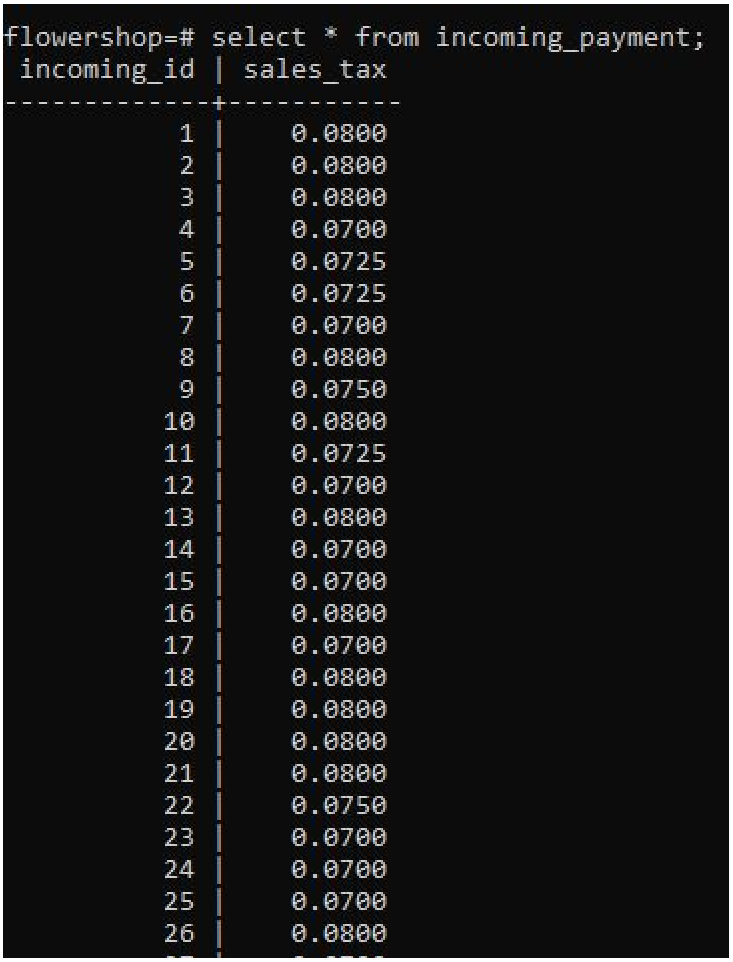


```
 length |      product_image       |                                    description                                     | supply_purchase_id
--------+--------------------------+------------------------------------------------------------------------------------+--------------------
   1.02 | redrose.png              | A beautiful thornless red rose                                                     |                 53
   6.05 | yellowtulip.png          | A beautiful tulip with a large yellow bulb                                         |                 23
   4.01 | whitebreath.png          | A common flower filler with small flowers coming off its branches                  |                  6
   8.68 | bluehydrangea.png        | Contains small flowers in bunches at the end of a long stem                        |                 66
   2.51 | yellowdaffodil.png       | Contains a trumpet shaped petal surrounded by a star shaped petal                  |                  1
   8.03 | whitelily.png            | Big flowers with a large petal span                                                |                 60
   5.18 | yellowchrsanthemum.png   | Blooms into a large beautiful flower                                               |                 32
   2.87 | pinkgerbera.png          | A part of the sunflower and daisy family. Appears to look like a colorful sunflower |                 30
   4.84 | pinkcarnation.png        | A commonly known flower with branched or forked clusters                           |                 26
  13.93 | orangebirdofparadise.png | Known for its distinct exotic look                                                 |                 43
```

Incoming Payment

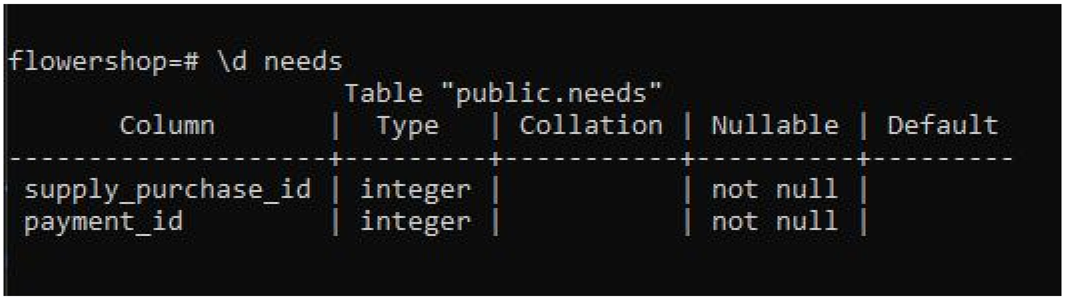


```
flowershop=# \d incoming_payment
                                    Table "public.incoming_payment"
   Column    |     Type      | Collation | Nullable |                        Default
-------------+---------------+-----------+----------+-------------------------------------------------------
 incoming_id | integer       |           | not null | nextval('incoming_payment_incoming_id_seq'::regclass)
 sales_tax   | numeric(10,4) |           | not null |

flowershop=#
```

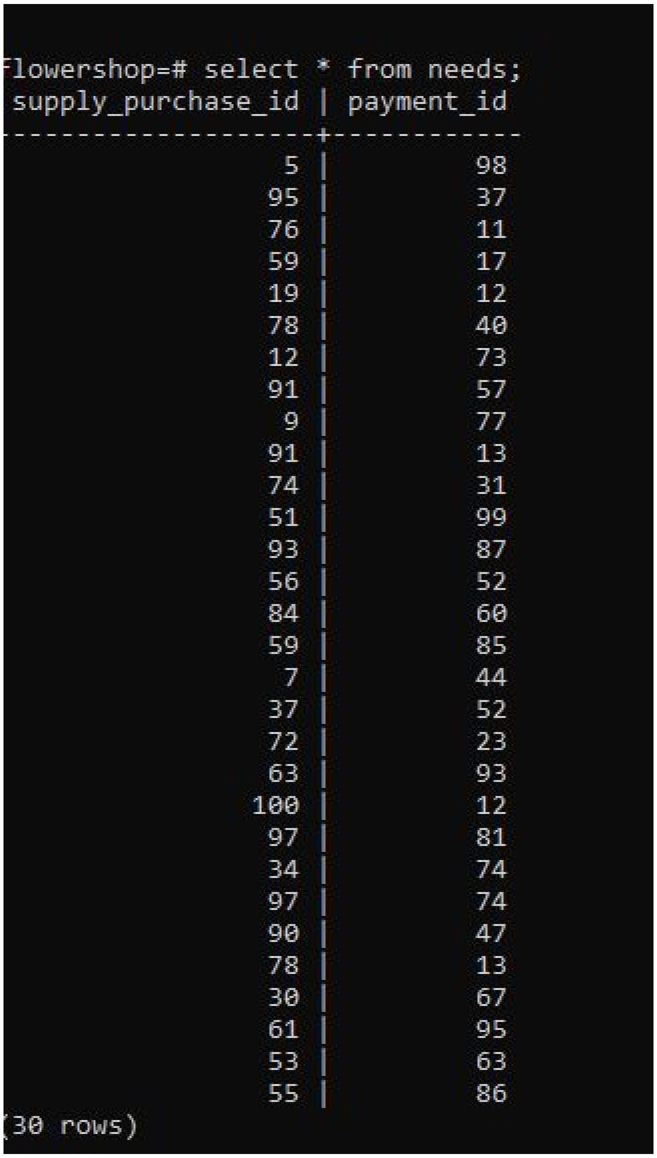


```
flowershop=# select * from incoming_payment;
 incoming_id | sales_tax
-------------+-----------
           1 |    0.0800
           2 |    0.0800
           3 |    0.0800
           4 |    0.0700
           5 |    0.0725
           6 |    0.0725
           7 |    0.0700
           8 |    0.0800
           9 |    0.0750
          10 |    0.0800
          11 |    0.0725
          12 |    0.0700
          13 |    0.0800
          14 |    0.0700
          15 |    0.0700
          16 |    0.0800
          17 |    0.0700
          18 |    0.0800
          19 |    0.0800
          20 |    0.0800
          21 |    0.0800
          22 |    0.0750
          23 |    0.0700
          24 |    0.0700
          25 |    0.0700
          26 |    0.0800
```

Needs

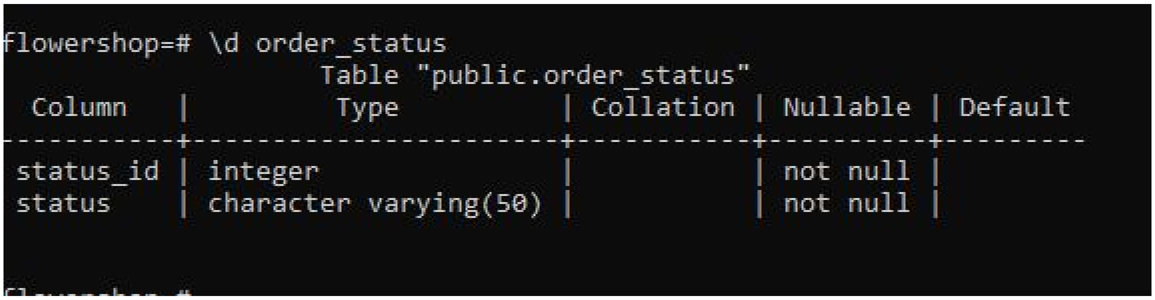


```
flowershop=# \d needs
                   Table "public.needs"
       Column       |  Type   | Collation | Nullable | Default
--------------------+---------+-----------+----------+---------
 supply_purchase_id | integer |           | not null |
 payment_id         | integer |           | not null |
```

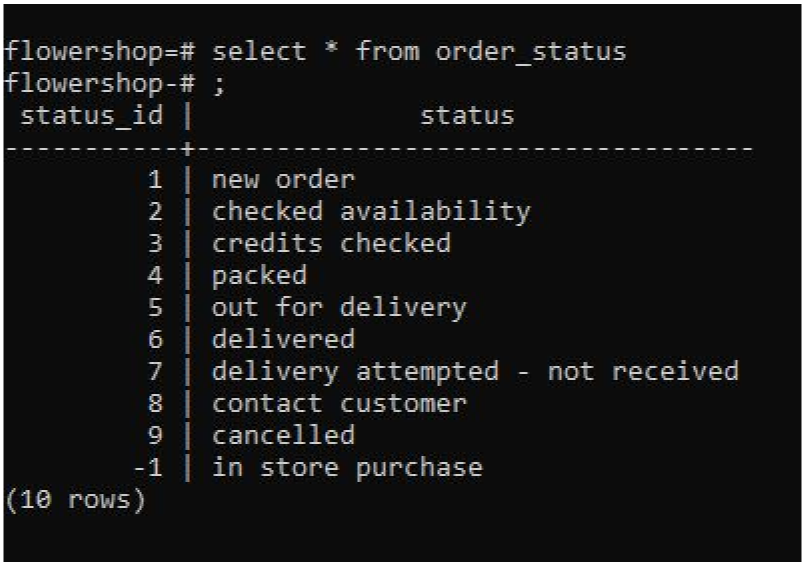


```
flowershop=# select * from needs;
 supply_purchase_id | payment_id
--------------------+------------
                  5 |         98
                 95 |         37
                 76 |         11
                 59 |         17
                 19 |         12
                 78 |         40
                 12 |         73
                 91 |         57
                  9 |         77
                 91 |         13
                 74 |         31
                 51 |         99
                 93 |         87
                 56 |         52
                 84 |         60
                 59 |         85
                  7 |         44
                 37 |         52
                 72 |         23
                 63 |         93
                100 |         12
                 97 |         81
                 34 |         74
                 97 |         74
                 90 |         47
                 78 |         13
                 30 |         67
                 61 |         95
                 53 |         63
                 55 |         86
(30 rows)
```

Order Status

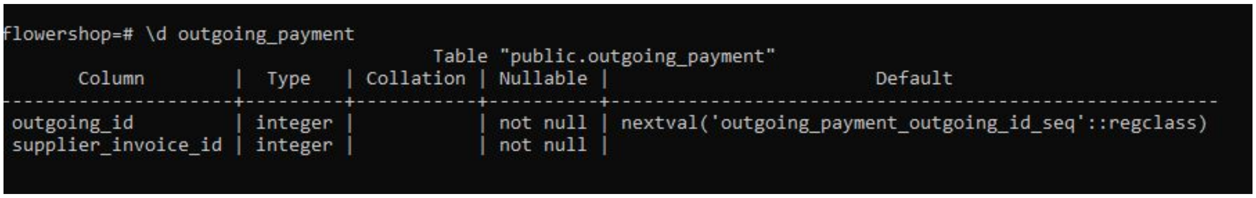


```
flowershop=# \d order_status
                    Table "public.order_status"
  Column   |         Type          | Collation | Nullable | Default
-----------+-----------------------+-----------+----------+---------
 status_id | integer               |           | not null |
 status    | character varying(50) |           | not null |


flowershop=#
```

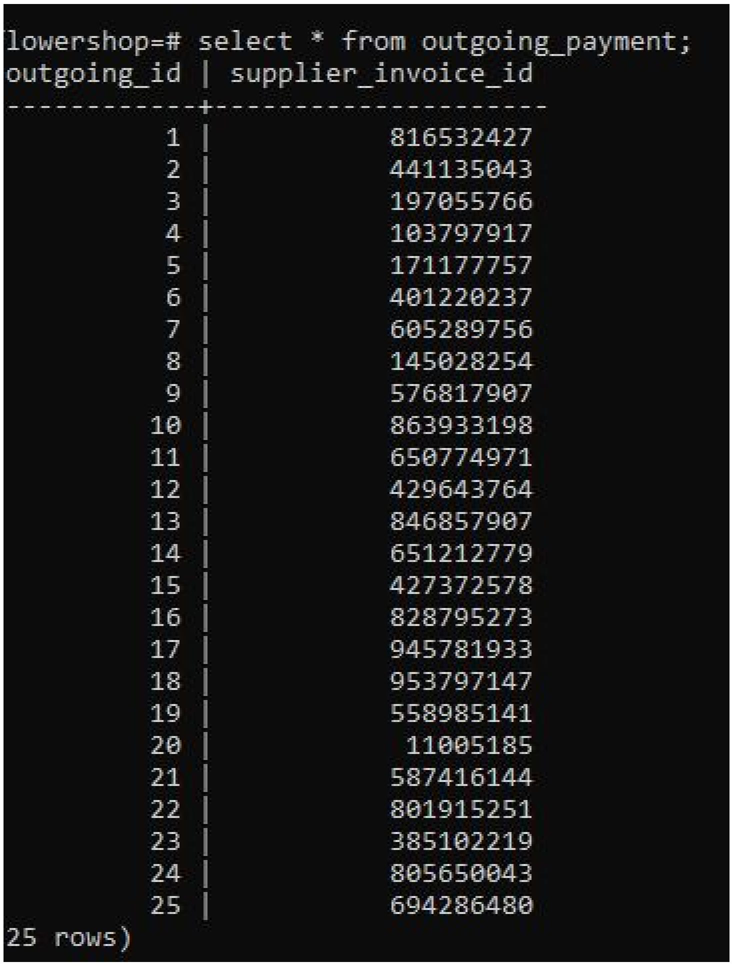


```
flowershop=# select * from order_status
flowershop-# ;
 status_id |              status
-----------+-----------------------------------
         1 | new order
         2 | checked availability
         3 | credits checked
         4 | packed
         5 | out for delivery
         6 | delivered
         7 | delivery attempted - not received
         8 | contact customer
         9 | cancelled
        -1 | in store purchase
(10 rows)
```

Outgoing Payment

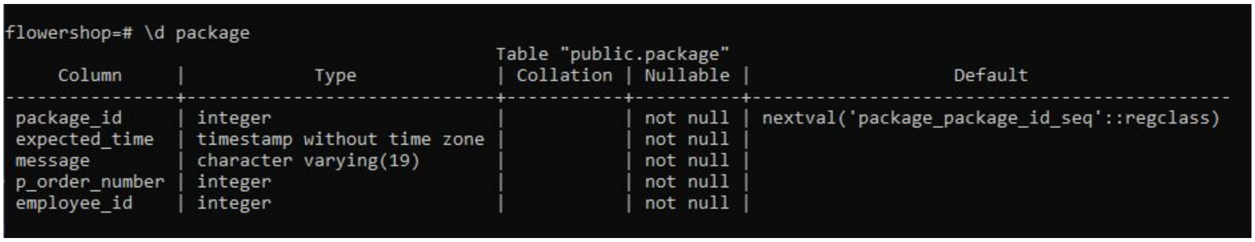


```
flowershop=# \d outgoing_payment
                                       Table "public.outgoing_payment"
       Column        |  Type   | Collation | Nullable |                      Default
---------------------+---------+-----------+----------+------------------------------------------------------
 outgoing_id         | integer |           | not null | nextval('outgoing_payment_outgoing_id_seq'::regclass)
 supplier_invoice_id | integer |           | not null |
```

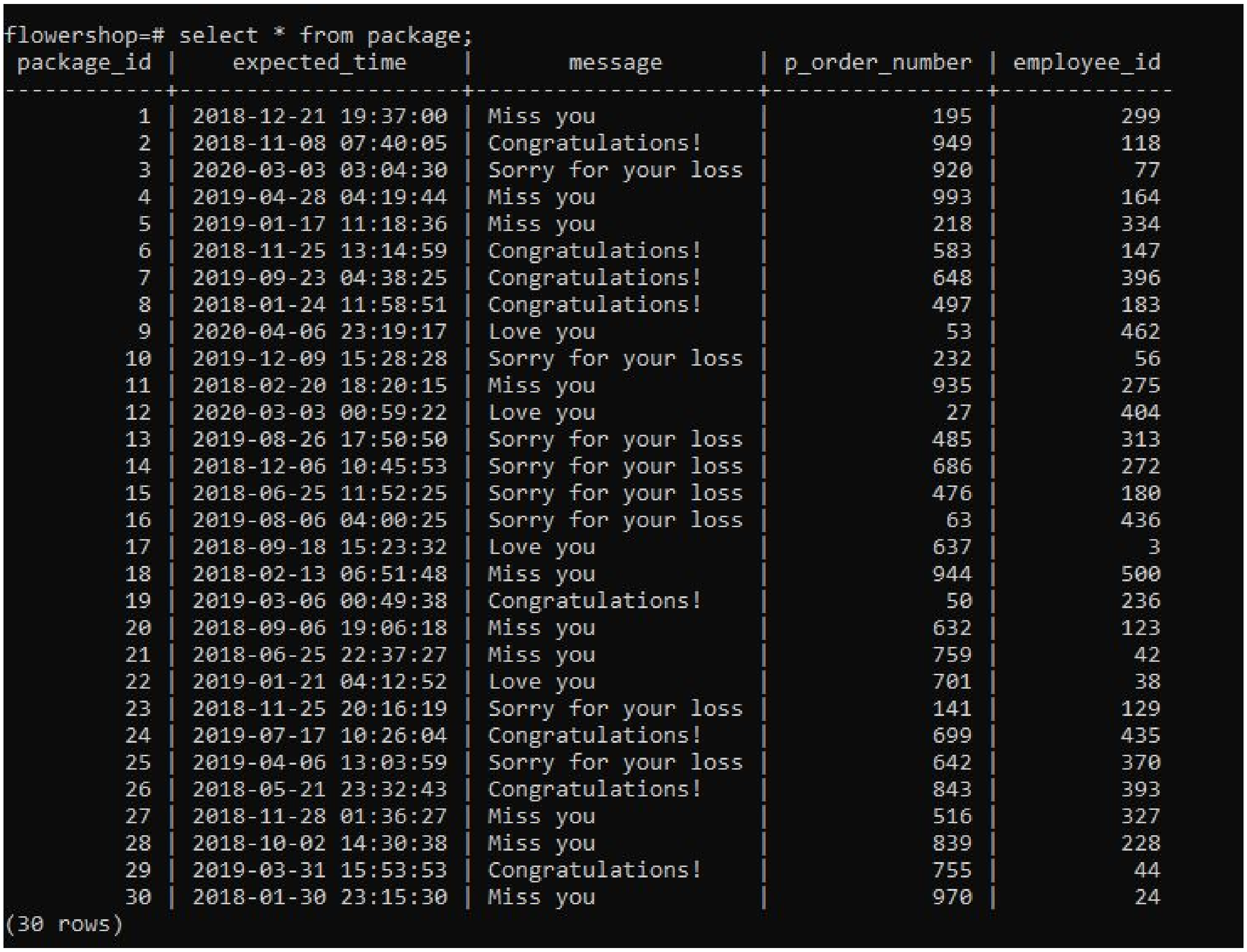


```
lowershop=# select * from outgoing_payment;
outgoing_id | supplier_invoice_id
------------+---------------------
          1 |           816532427
          2 |           441135043
          3 |           197055766
          4 |           103797917
          5 |           171177757
          6 |           401220237
          7 |           605289756
          8 |           145028254
          9 |           576817907
         10 |           863933198
         11 |           650774971
         12 |           429643764
         13 |           846857907
         14 |           651212779
         15 |           427372578
         16 |           828795273
         17 |           945781933
         18 |           953797147
         19 |           558985141
         20 |            11005185
         21 |           587416144
         22 |           801915251
         23 |           385102219
         24 |           805650043
         25 |           694286480
25 rows)
```

Package

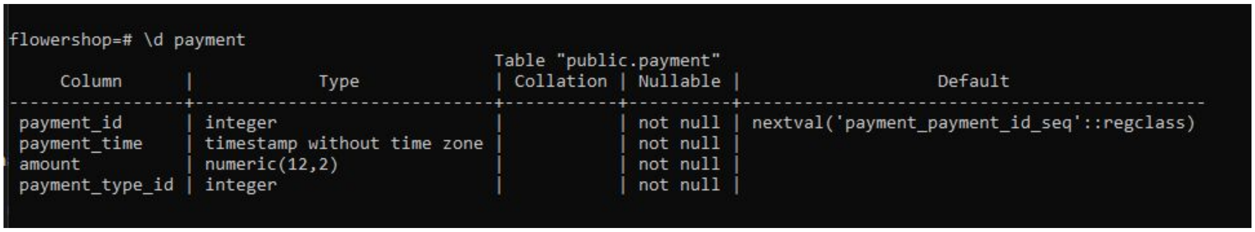


```
flowershop=# \d package
                                        Table "public.package"
    Column     |            Type             | Collation | Nullable |                 Default
---------------+-----------------------------+-----------+----------+------------------------------------------
 package_id    | integer                     |           | not null | nextval('package_package_id_seq'::regclass)
 expected_time | timestamp without time zone |           | not null |
 message       | character varying(19)       |           | not null |
 p_order_number | integer                    |           | not null |
 employee_id   | integer                     |           | not null |
```

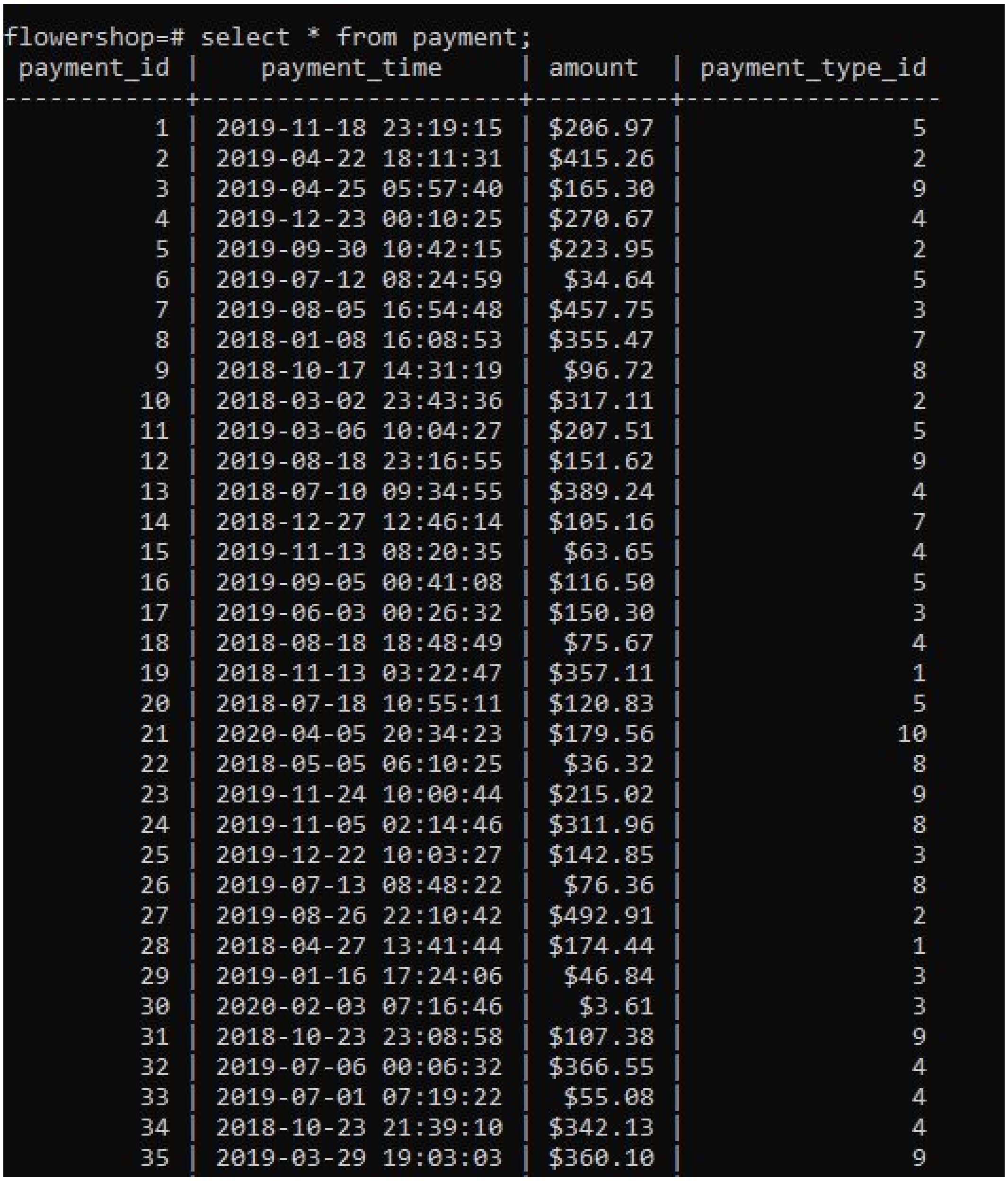


```
flowershop=# select * from package;
 package_id |    expected_time    |        message        | p_order_number | employee_id
------------+---------------------+-----------------------+----------------+-------------
          1 | 2018-12-21 19:37:00 | Miss you              |            195 |         299
          2 | 2018-11-08 07:40:05 | Congratulations!      |            949 |         118
          3 | 2020-03-03 03:04:30 | Sorry for your loss   |            920 |          77
          4 | 2019-04-28 04:19:44 | Miss you              |            993 |         164
          5 | 2019-01-17 11:18:36 | Miss you              |            218 |         334
          6 | 2018-11-25 13:14:59 | Congratulations!      |            583 |         147
          7 | 2019-09-23 04:38:25 | Congratulations!      |            648 |         396
          8 | 2018-01-24 11:58:51 | Congratulations!      |            497 |         183
          9 | 2020-04-06 23:19:17 | Love you              |             53 |         462
         10 | 2019-12-09 15:28:28 | Sorry for your loss   |            232 |          56
         11 | 2018-02-20 18:20:15 | Miss you              |            935 |         275
         12 | 2020-03-03 00:59:22 | Love you              |             27 |         404
         13 | 2019-08-26 17:50:50 | Sorry for your loss   |            485 |         313
         14 | 2018-12-06 10:45:53 | Sorry for your loss   |            686 |         272
         15 | 2018-06-25 11:52:25 | Sorry for your loss   |            476 |         180
         16 | 2019-08-06 04:00:25 | Sorry for your loss   |             63 |         436
         17 | 2018-09-18 15:23:32 | Love you              |            637 |           3
         18 | 2018-02-13 06:51:48 | Miss you              |            944 |         500
         19 | 2019-03-06 00:49:38 | Congratulations!      |             50 |         236
         20 | 2018-09-06 19:06:18 | Miss you              |            632 |         123
         21 | 2018-06-25 22:37:27 | Miss you              |            759 |          42
         22 | 2019-01-21 04:12:52 | Love you              |            701 |          38
         23 | 2018-11-25 20:16:19 | Sorry for your loss   |            141 |         129
         24 | 2019-07-17 10:26:04 | Congratulations!      |            699 |         435
         25 | 2019-04-06 13:03:59 | Sorry for your loss   |            642 |         370
         26 | 2018-05-21 23:32:43 | Congratulations!      |            843 |         393
         27 | 2018-11-28 01:36:27 | Miss you              |            516 |         327
         28 | 2018-10-02 14:30:38 | Miss you              |            839 |         228
         29 | 2019-03-31 15:53:53 | Congratulations!      |            755 |          44
         30 | 2018-01-30 23:15:30 | Miss you              |            970 |          24
(30 rows)
```

Payment

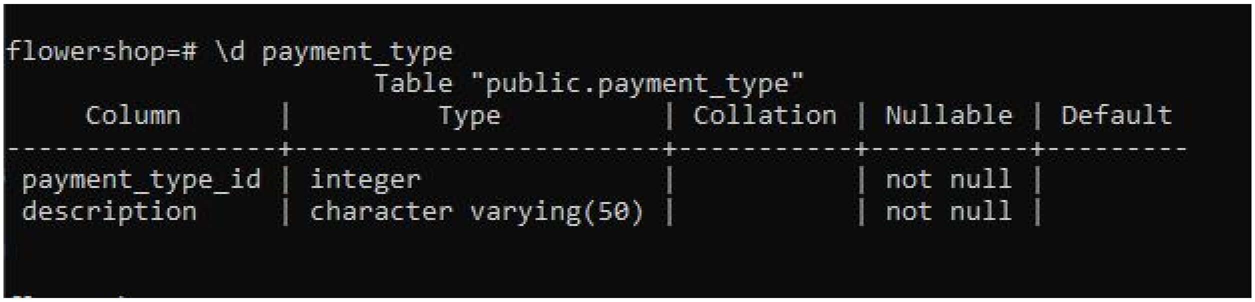


```
flowershop=# \d payment
                        Table "public.payment"
     Column      |            Type             | Collation | Nullable |                   Default
-----------------+-----------------------------+-----------+----------+---------------------------------------------
 payment_id      | integer                     |           | not null | nextval('payment_payment_id_seq'::regclass)
 payment_time    | timestamp without time zone |           | not null |
 amount          | numeric(12,2)               |           | not null |
 payment_type_id | integer                     |           | not null |
```

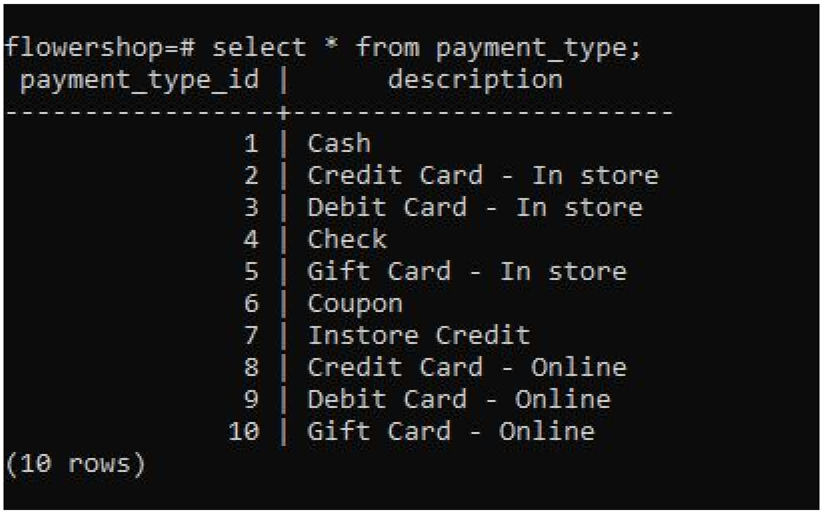


```
flowershop=# select * from payment;
 payment_id |    payment_time     | amount  | payment_type_id
------------+---------------------+---------+-----------------
          1 | 2019-11-18 23:19:15 | $206.97 |               5
          2 | 2019-04-22 18:11:31 | $415.26 |               2
          3 | 2019-04-25 05:57:40 | $165.30 |               9
          4 | 2019-12-23 00:10:25 | $270.67 |               4
          5 | 2019-09-30 10:42:15 | $223.95 |               2
          6 | 2019-07-12 08:24:59 |  $34.64 |               5
          7 | 2019-08-05 16:54:48 | $457.75 |               3
          8 | 2018-01-08 16:08:53 | $355.47 |               7
          9 | 2018-10-17 14:31:19 |  $96.72 |               8
         10 | 2018-03-02 23:43:36 | $317.11 |               2
         11 | 2019-03-06 10:04:27 | $207.51 |               5
         12 | 2019-08-18 23:16:55 | $151.62 |               9
         13 | 2018-07-10 09:34:55 | $389.24 |               4
         14 | 2018-12-27 12:46:14 | $105.16 |               7
         15 | 2019-11-13 08:20:35 |  $63.65 |               4
         16 | 2019-09-05 00:41:08 | $116.50 |               5
         17 | 2019-06-03 00:26:32 | $150.30 |               3
         18 | 2018-08-18 18:48:49 |  $75.67 |               4
         19 | 2018-11-13 03:22:47 | $357.11 |               1
         20 | 2018-07-18 10:55:11 | $120.83 |               5
         21 | 2020-04-05 20:34:23 | $179.56 |              10
         22 | 2018-05-05 06:10:25 |  $36.32 |               8
         23 | 2019-11-24 10:00:44 | $215.02 |               9
         24 | 2019-11-05 02:14:46 | $311.96 |               8
         25 | 2019-12-22 10:03:27 | $142.85 |               3
         26 | 2019-07-13 08:48:22 |  $76.36 |               8
         27 | 2019-08-26 22:10:42 | $492.91 |               2
         28 | 2018-04-27 13:41:44 | $174.44 |               1
         29 | 2019-01-16 17:24:06 |  $46.84 |               3
         30 | 2020-02-03 07:16:46 |   $3.61 |               3
         31 | 2018-10-23 23:08:58 | $107.38 |               9
         32 | 2019-07-06 00:06:32 | $366.55 |               4
         33 | 2019-07-01 07:19:22 |  $55.08 |               4
         34 | 2018-10-23 21:39:10 | $342.13 |               4
         35 | 2019-03-29 19:03:03 | $360.10 |               9
```

Payment Type

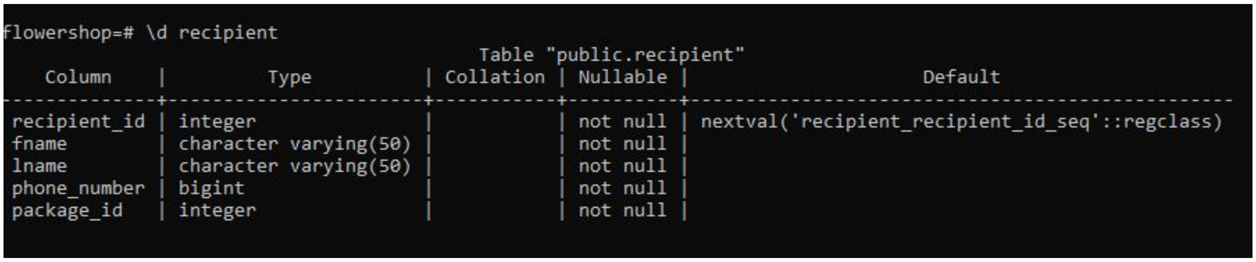


```
flowershop=# \d payment_type
                      Table "public.payment_type"
     Column      |         Type          | Collation | Nullable | Default
-----------------+-----------------------+-----------+----------+---------
 payment_type_id | integer               |           | not null |
 description     | character varying(50) |           | not null |
```

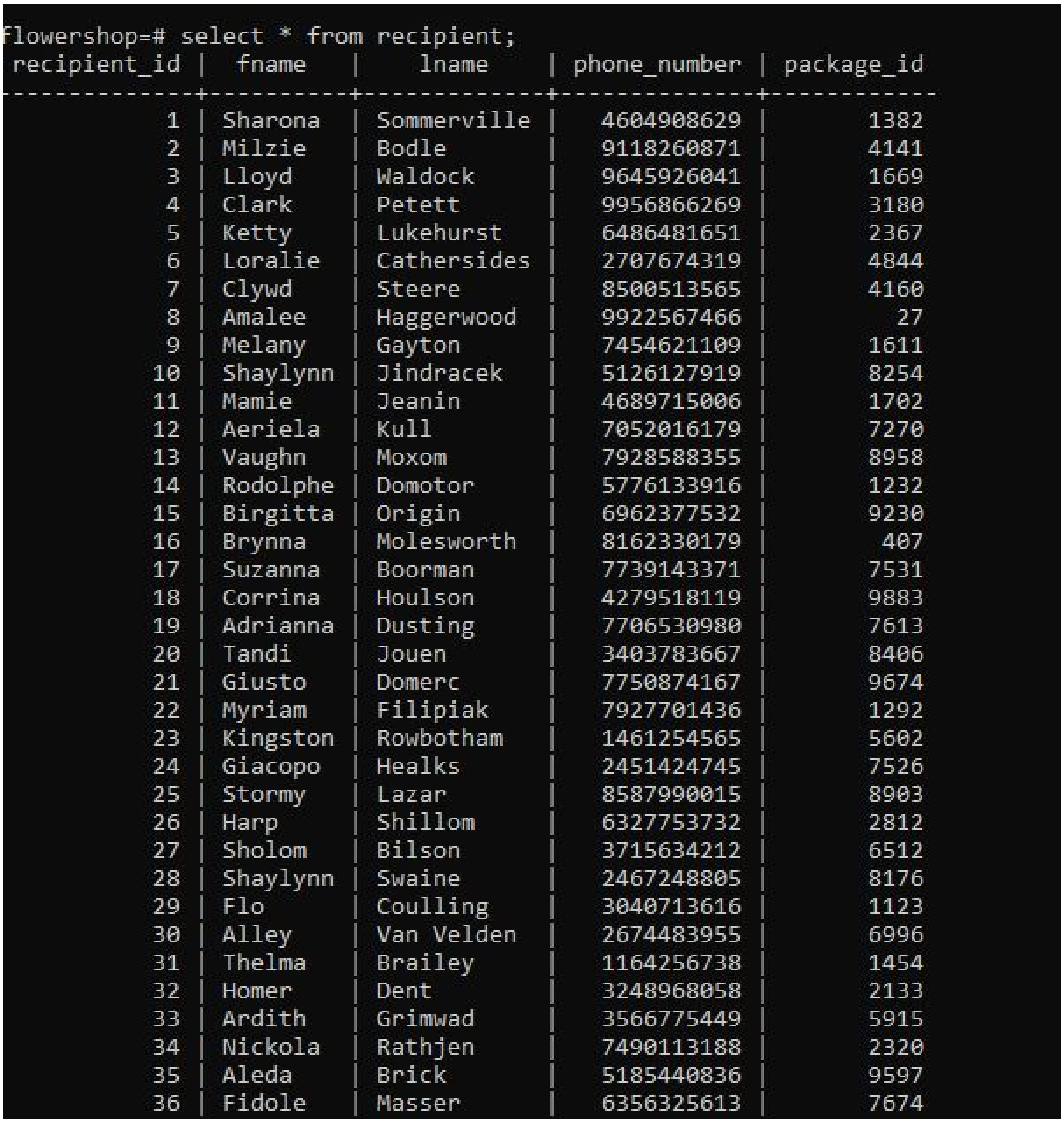


```
flowershop=# select * from payment_type;
 payment_type_id |      description
-----------------+------------------------
               1 | Cash
               2 | Credit Card - In store
               3 | Debit Card - In store
               4 | Check
               5 | Gift Card - In store
               6 | Coupon
               7 | Instore Credit
               8 | Credit Card - Online
               9 | Debit Card - Online
              10 | Gift Card - Online
(10 rows)
```

Recipient

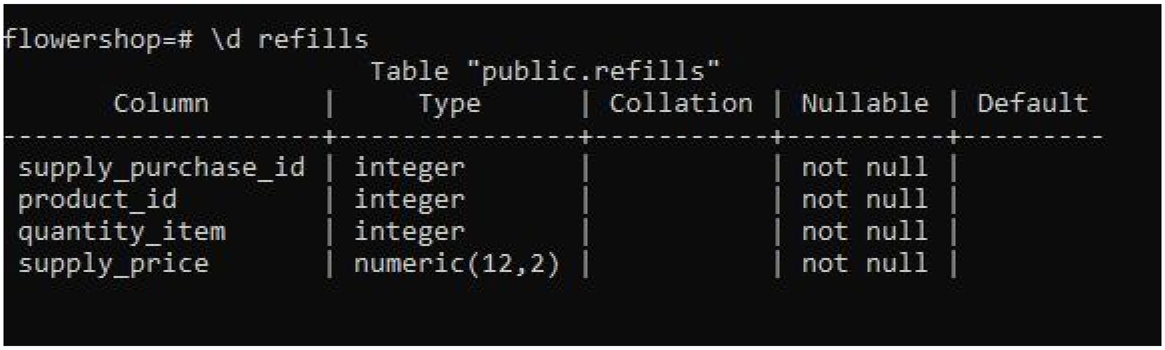


```
flowershop=# \d recipient
                                     Table "public.recipient"
    Column    |         Type          | Collation | Nullable |                     Default
--------------+-----------------------+-----------+----------+-------------------------------------------------
 recipient_id | integer               |           | not null | nextval('recipient_recipient_id_seq'::regclass)
 fname        | character varying(50) |           | not null |
 lname        | character varying(50) |           | not null |
 phone_number | bigint                |           | not null |
 package_id   | integer               |           | not null |
```

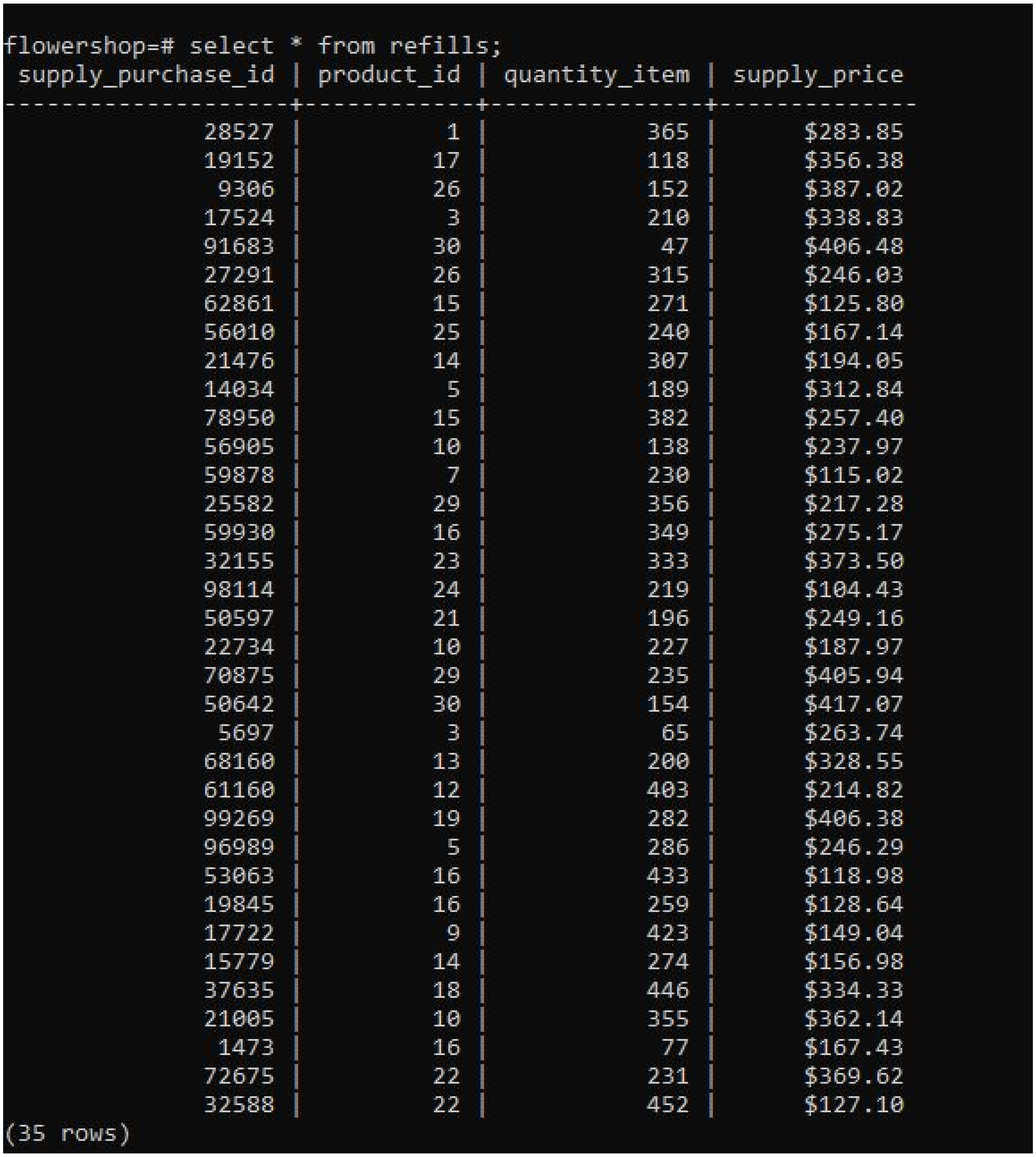


```
flowershop=# select * from recipient;
 recipient_id |  fname   |    lname    | phone_number | package_id
--------------+----------+-------------+--------------+------------
            1 | Sharona  | Sommerville |   4604908629 |       1382
            2 | Milzie   | Bodle       |   9118260871 |       4141
            3 | Lloyd    | Waldock     |   9645926041 |       1669
            4 | Clark    | Petett      |   9956866269 |       3180
            5 | Ketty    | Lukehurst   |   6486481651 |       2367
            6 | Loralie  | Cathersides |   2707674319 |       4844
            7 | Clywd    | Steere      |   8500513565 |       4160
            8 | Amalee   | Haggerwood  |   9922567466 |         27
            9 | Melany   | Gayton      |   7454621109 |       1611
           10 | Shaylynn | Jindracek   |   5126127919 |       8254
           11 | Mamie    | Jeanin      |   4689715006 |       1702
           12 | Aeriela  | Kull        |   7052016179 |       7270
           13 | Vaughn   | Moxom       |   7928588355 |       8958
           14 | Rodolphe | Domotor     |   5776133916 |       1232
           15 | Birgitta | Origin      |   6962377532 |       9230
           16 | Brynna   | Molesworth  |   8162330179 |        407
           17 | Suzanna  | Boorman     |   7739143371 |       7531
           18 | Corrina  | Houlson     |   4279518119 |       9883
           19 | Adrianna | Dusting     |   7706530980 |       7613
           20 | Tandi    | Jouen       |   3403783667 |       8406
           21 | Giusto   | Domerc      |   7750874167 |       9674
           22 | Myriam   | Filipiak    |   7927701436 |       1292
           23 | Kingston | Rowbotham   |   1461254565 |       5602
           24 | Giacopo  | Healks      |   2451424745 |       7526
           25 | Stormy   | Lazar       |   8587990015 |       8903
           26 | Harp     | Shillom     |   6327753732 |       2812
           27 | Sholom   | Bilson      |   3715634212 |       6512
           28 | Shaylynn | Swaine      |   2467248805 |       8176
           29 | Flo      | Coulling    |   3040713616 |       1123
           30 | Alley    | Van Velden  |   2674483955 |       6996
           31 | Thelma   | Brailey     |   1164256738 |       1454
           32 | Homer    | Dent        |   3248968058 |       2133
           33 | Ardith   | Grimwad     |   3566775449 |       5915
           34 | Nickola  | Rathjen     |   7490113188 |       2320
           35 | Aleda    | Brick       |   5185440836 |       9597
           36 | Fidole   | Masser      |   6356325613 |       7674
```

Refills

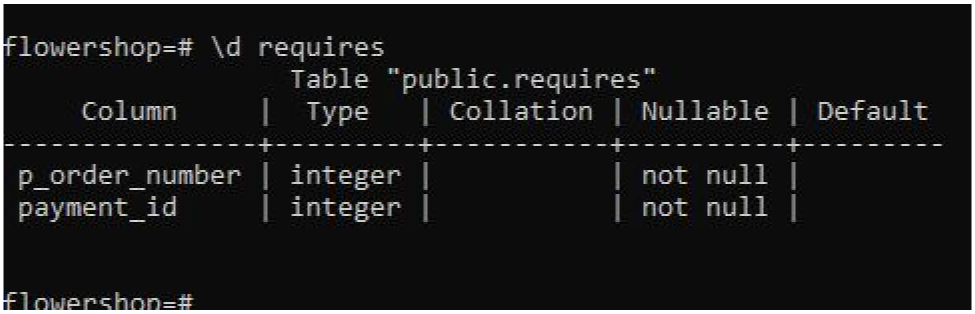


```
flowershop=# \d refills
                       Table "public.refills"
       Column        |     Type      | Collation | Nullable | Default
---------------------+---------------+-----------+----------+---------
 supply_purchase_id  | integer       |           | not null |
 product_id          | integer       |           | not null |
 quantity_item       | integer       |           | not null |
 supply_price        | numeric(12,2) |           | not null |
```

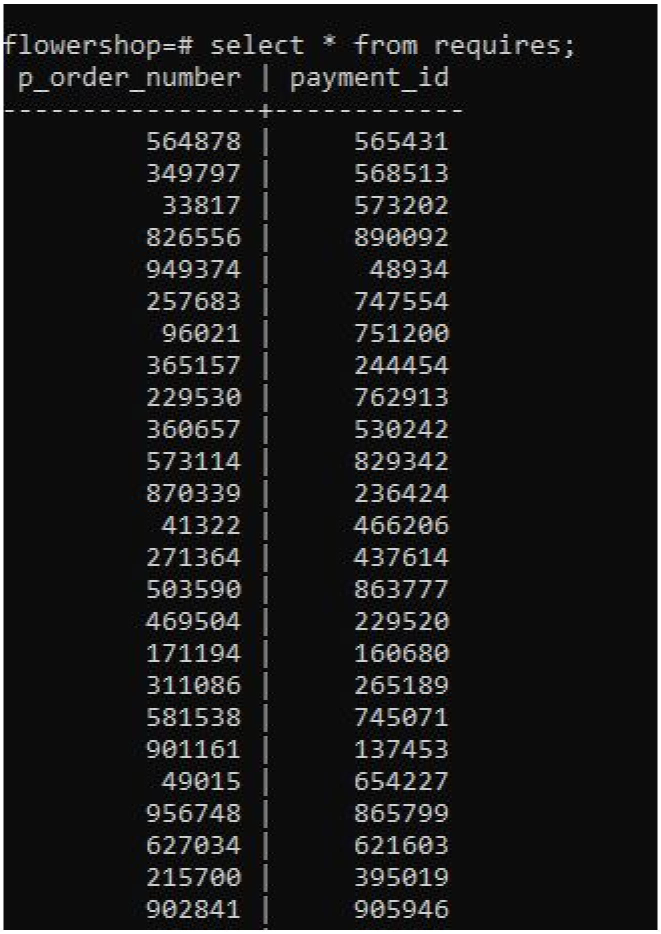


```
flowershop=# select * from refills;
 supply_purchase_id | product_id | quantity_item | supply_price
--------------------+------------+---------------+--------------
              28527 |          1 |           365 |      $283.85
              19152 |         17 |           118 |      $356.38
               9306 |         26 |           152 |      $387.02
              17524 |          3 |           210 |      $338.83
              91683 |         30 |            47 |      $406.48
              27291 |         26 |           315 |      $246.03
              62861 |         15 |           271 |      $125.80
              56010 |         25 |           240 |      $167.14
              21476 |         14 |           307 |      $194.05
              14034 |          5 |           189 |      $312.84
              78950 |         15 |           382 |      $257.40
              56905 |         10 |           138 |      $237.97
              59878 |          7 |           230 |      $115.02
              25582 |         29 |           356 |      $217.28
              59930 |         16 |           349 |      $275.17
              32155 |         23 |           333 |      $373.50
              98114 |         24 |           219 |      $104.43
              50597 |         21 |           196 |      $249.16
              22734 |         10 |           227 |      $187.97
              70875 |         29 |           235 |      $405.94
              50642 |         30 |           154 |      $417.07
               5697 |          3 |            65 |      $263.74
              68160 |         13 |           200 |      $328.55
              61160 |         12 |           403 |      $214.82
              99269 |         19 |           282 |      $406.38
              96989 |          5 |           286 |      $246.29
              53063 |         16 |           433 |      $118.98
              19845 |         16 |           259 |      $128.64
              17722 |          9 |           423 |      $149.04
              15779 |         14 |           274 |      $156.98
              37635 |         18 |           446 |      $334.33
              21005 |         10 |           355 |      $362.14
               1473 |         16 |            77 |      $167.43
              72675 |         22 |           231 |      $369.62
              32588 |         22 |           452 |      $127.10
(35 rows)
```

Requires

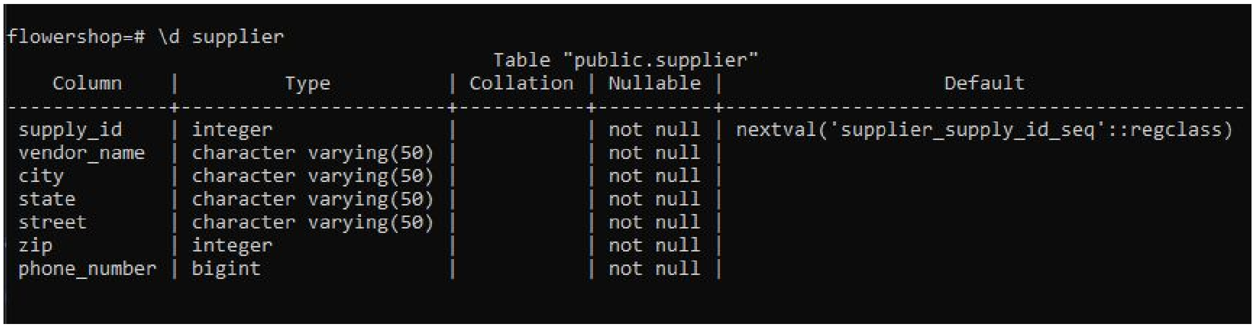


```
flowershop=# \d requires
                    Table "public.requires"
     Column      |  Type   | Collation | Nullable | Default
-----------------+---------+-----------+----------+---------
 p_order_number | integer |           | not null |
 payment_id     | integer |           | not null |


flowershop=#
```

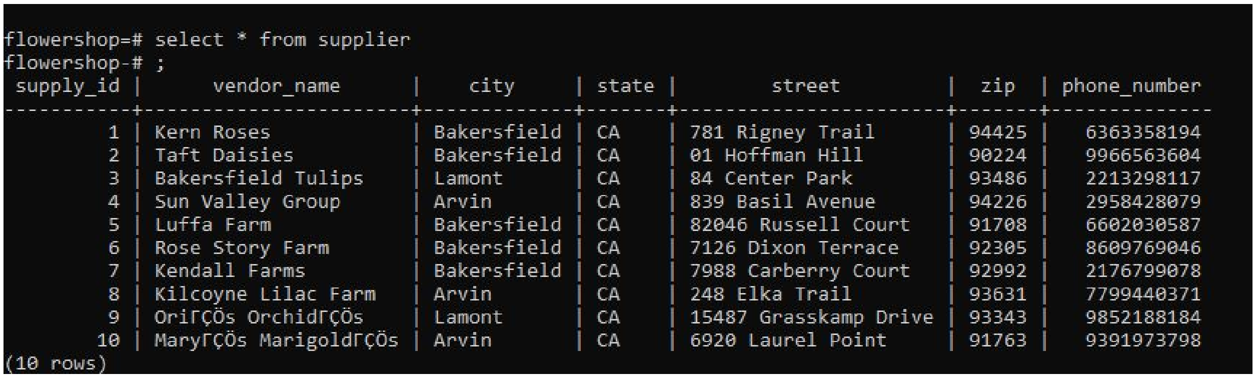


```
flowershop=# select * from requires;
 p_order_number | payment_id
----------------+------------
         564878 |     565431
         349797 |     568513
          33817 |     573202
         826556 |     890092
         949374 |      48934
         257683 |     747554
          96021 |     751200
         365157 |     244454
         229530 |     762913
         360657 |     530242
         573114 |     829342
         870339 |     236424
          41322 |     466206
         271364 |     437614
         503590 |     863777
         469504 |     229520
         171194 |     160680
         311086 |     265189
         581538 |     745071
         901161 |     137453
          49015 |     654227
         956748 |     865799
         627034 |     621603
         215700 |     395019
         902841 |     905946
```

Supplier

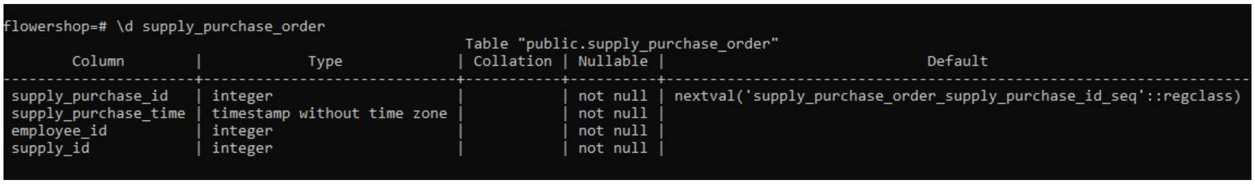


```
flowershop=# \d supplier
                                     Table "public.supplier"
    Column    |         Type          | Collation | Nullable |                  Default
--------------+-----------------------+-----------+----------+-------------------------------------------
 supply_id    | integer               |           | not null | nextval('supplier_supply_id_seq'::regclass)
 vendor_name  | character varying(50) |           | not null |
 city         | character varying(50) |           | not null |
 state        | character varying(50) |           | not null |
 street       | character varying(50) |           | not null |
 zip          | integer               |           | not null |
 phone_number | bigint                |           | not null |
```

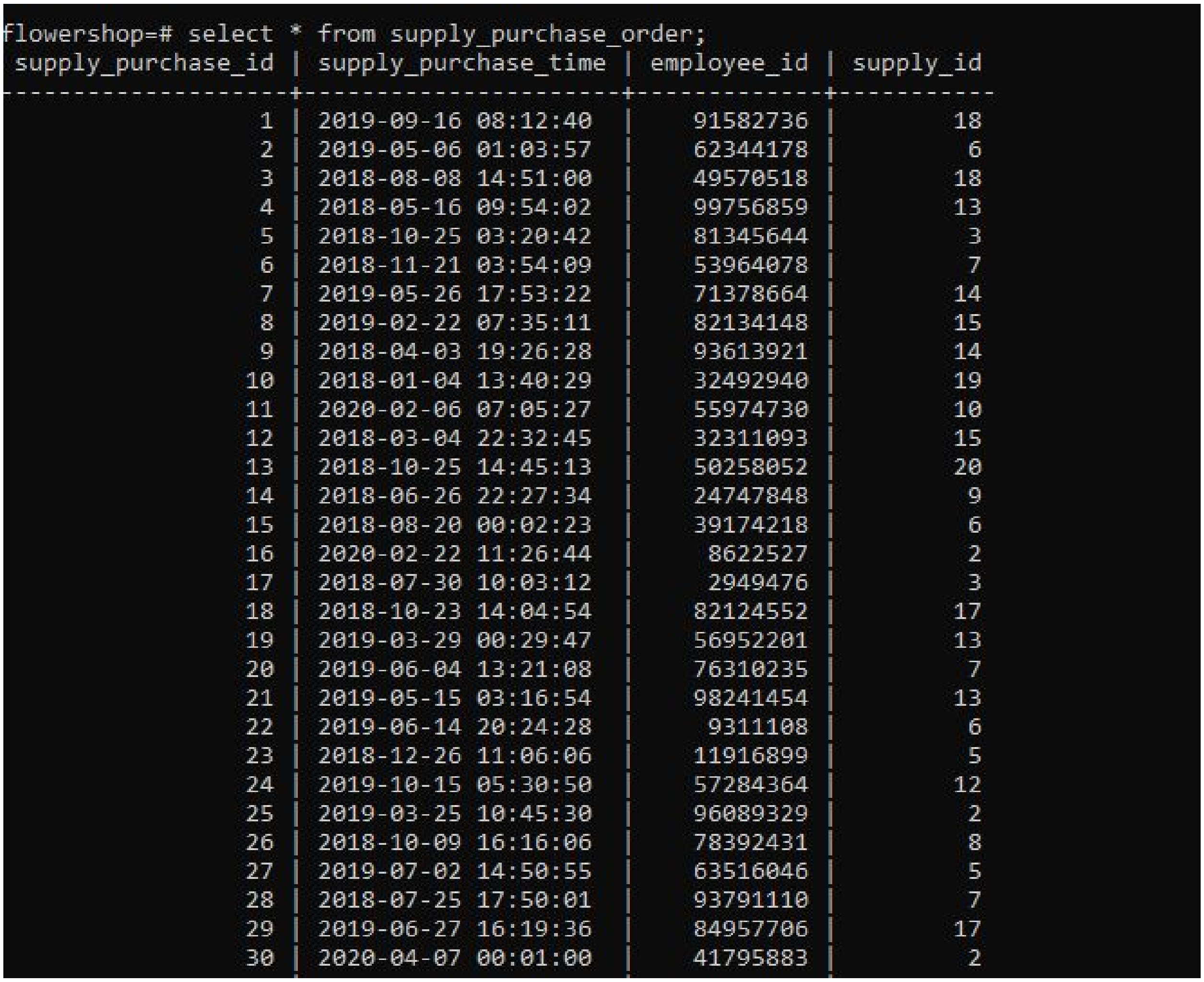


```
flowershop=# select * from supplier
flowershop-# ;
 supply_id |      vendor_name      |    city     | state |         street         |  zip  | phone_number
-----------+-----------------------+-------------+-------+------------------------+-------+--------------
         1 | Kern Roses            | Bakersfield | CA    | 781 Rigney Trail       | 94425 |   6363358194
         2 | Taft Daisies          | Bakersfield | CA    | 01 Hoffman Hill        | 90224 |   9966563604
         3 | Bakersfield Tulips    | Lamont      | CA    | 84 Center Park         | 93486 |   2213298117
         4 | Sun Valley Group      | Arvin       | CA    | 839 Basil Avenue       | 94226 |   2958428079
         5 | Luffa Farm            | Bakersfield | CA    | 82046 Russell Court    | 91708 |   6602030587
         6 | Rose Story Farm       | Bakersfield | CA    | 7126 Dixon Terrace     | 92305 |   8609769046
         7 | Kendall Farms         | Bakersfield | CA    | 7988 Carberry Court    | 92992 |   2176799078
         8 | Kilcoyne Lilac Farm   | Arvin       | CA    | 248 Elka Trail         | 93631 |   7799440371
         9 | OriΓÇÖs OrchidΓÇÖs    | Lamont      | CA    | 15487 Grasskamp Drive  | 93343 |   9852188184
        10 | MaryΓÇÖs MarigoldΓÇÖs | Arvin       | CA    | 6920 Laurel Point      | 91763 |   9391973798
(10 rows)
```

Supply Purchase Order

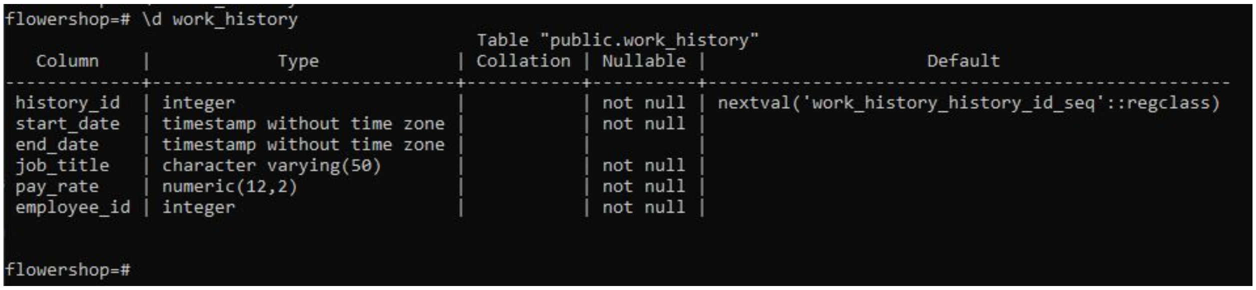


```
flowershop=# \d supply_purchase_order
                                           Table "public.supply_purchase_order"
        Column        |            Type             | Collation | Nullable |                            Default
----------------------+-----------------------------+-----------+----------+----------------------------------------------------------------
 supply_purchase_id   | integer                     |           | not null | nextval('supply_purchase_order_supply_purchase_id_seq'::regclass)
 supply_purchase_time | timestamp without time zone |           | not null |
 employee_id          | integer                     |           | not null |
 supply_id            | integer                     |           | not null |
```

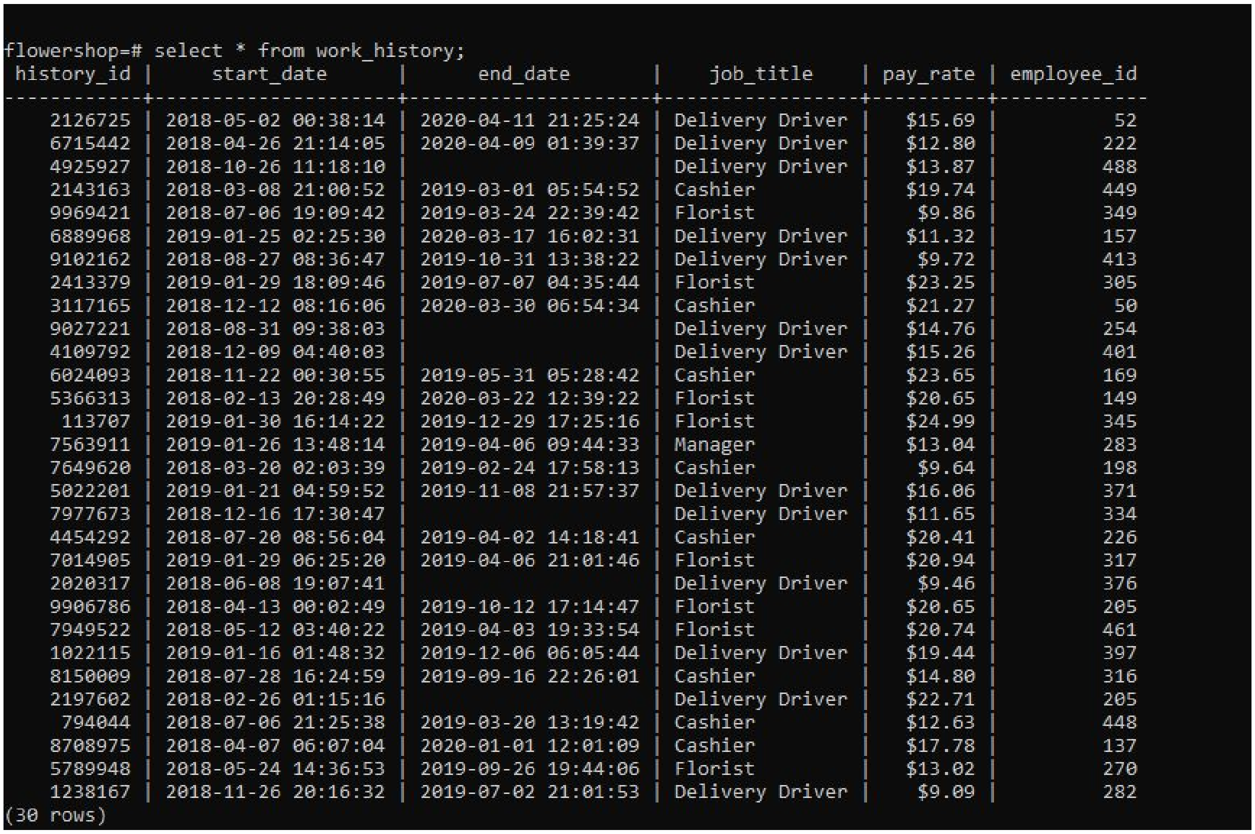


```
flowershop=# select * from supply_purchase_order;
 supply_purchase_id | supply_purchase_time | employee_id | supply_id
--------------------+----------------------+-------------+-----------
                  1 | 2019-09-16 08:12:40  |    91582736 |        18
                  2 | 2019-05-06 01:03:57  |    62344178 |         6
                  3 | 2018-08-08 14:51:00  |    49570518 |        18
                  4 | 2018-05-16 09:54:02  |    99756859 |        13
                  5 | 2018-10-25 03:20:42  |    81345644 |         3
                  6 | 2018-11-21 03:54:09  |    53964078 |         7
                  7 | 2019-05-26 17:53:22  |    71378664 |        14
                  8 | 2019-02-22 07:35:11  |    82134148 |        15
                  9 | 2018-04-03 19:26:28  |    93613921 |        14
                 10 | 2018-01-04 13:40:29  |    32492940 |        19
                 11 | 2020-02-06 07:05:27  |    55974730 |        10
                 12 | 2018-03-04 22:32:45  |    32311093 |        15
                 13 | 2018-10-25 14:45:13  |    50258052 |        20
                 14 | 2018-06-26 22:27:34  |    24747848 |         9
                 15 | 2018-08-20 00:02:23  |    39174218 |         6
                 16 | 2020-02-22 11:26:44  |     8622527 |         2
                 17 | 2018-07-30 10:03:12  |     2949476 |         3
                 18 | 2018-10-23 14:04:54  |    82124552 |        17
                 19 | 2019-03-29 00:29:47  |    56952201 |        13
                 20 | 2019-06-04 13:21:08  |    76310235 |         7
                 21 | 2019-05-15 03:16:54  |    98241454 |        13
                 22 | 2019-06-14 20:24:28  |     9311108 |         6
                 23 | 2018-12-26 11:06:06  |    11916899 |         5
                 24 | 2019-10-15 05:30:50  |    57284364 |        12
                 25 | 2019-03-25 10:45:30  |    96089329 |         2
                 26 | 2018-10-09 16:16:06  |    78392431 |         8
                 27 | 2019-07-02 14:50:55  |    63516046 |         5
                 28 | 2018-07-25 17:50:01  |    93791110 |         7
                 29 | 2019-06-27 16:19:36  |    84957706 |        17
                 30 | 2020-04-07 00:01:00  |    41795883 |         2
```

Work History

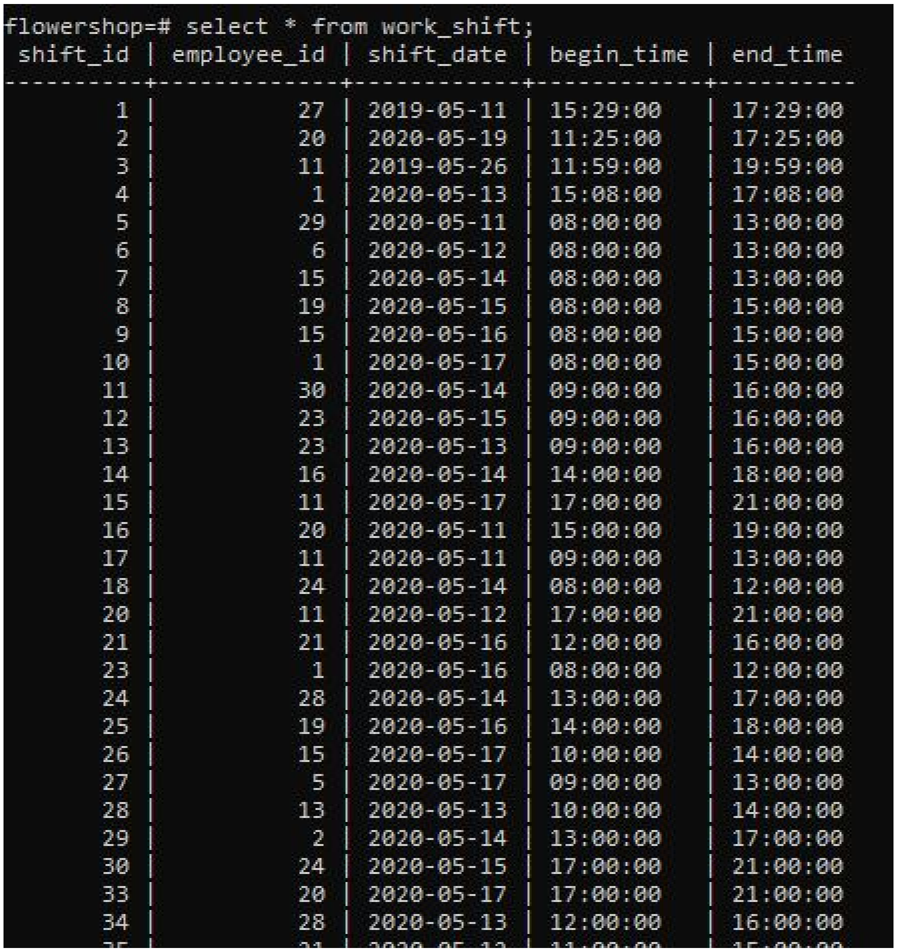


```
flowershop=# \d work_history
                                     Table "public.work_history"
   Column    |            Type             | Collation | Nullable |                    Default
-------------+-----------------------------+-----------+----------+-----------------------------------------------
 history_id  | integer                     |           | not null | nextval('work_history_history_id_seq'::regclass)
 start_date  | timestamp without time zone |           | not null |
 end_date    | timestamp without time zone |           |          |
 job_title   | character varying(50)       |           | not null |
 pay_rate    | numeric(12,2)               |           | not null |
 employee_id | integer                     |           | not null |


flowershop=#
```



```
flowershop=# select * from work_history;
 history_id |      start_date      |       end_date       |    job_title    | pay_rate | employee_id
------------+---------------------+---------------------+-----------------+----------+-------------
    2126725 | 2018-05-02 00:38:14 | 2020-04-11 21:25:24 | Delivery Driver |   $15.69 |          52
    6715442 | 2018-04-26 21:14:05 | 2020-04-09 01:39:37 | Delivery Driver |   $12.80 |         222
    4925927 | 2018-10-26 11:18:10 |                     | Delivery Driver |   $13.87 |         488
    2143163 | 2018-03-08 21:00:52 | 2019-03-01 05:54:52 | Cashier         |   $19.74 |         449
    9969421 | 2018-07-06 19:09:42 | 2019-03-24 22:39:42 | Florist         |    $9.86 |         349
    6889968 | 2019-01-25 02:25:30 | 2020-03-17 16:02:31 | Delivery Driver |   $11.32 |         157
    9102162 | 2018-08-27 08:36:47 | 2019-10-31 13:38:22 | Delivery Driver |    $9.72 |         413
    2413379 | 2019-01-29 18:09:46 | 2019-07-07 04:35:44 | Florist         |   $23.25 |         305
    3117165 | 2018-12-12 08:16:06 | 2020-03-30 06:54:34 | Cashier         |   $21.27 |          50
    9027221 | 2018-08-31 09:38:03 |                     | Delivery Driver |   $14.76 |         254
    4109792 | 2018-12-09 04:40:03 |                     | Delivery Driver |   $15.26 |         401
    6024093 | 2018-11-22 00:30:55 | 2019-05-31 05:28:42 | Cashier         |   $23.65 |         169
    5366313 | 2018-02-13 20:28:49 | 2020-03-22 12:39:22 | Florist         |   $20.65 |         149
     113707 | 2019-01-30 16:14:22 | 2019-12-29 17:25:16 | Florist         |   $24.99 |         345
    7563911 | 2019-01-26 13:48:14 | 2019-04-06 09:44:33 | Manager         |   $13.04 |         283
    7649620 | 2018-03-20 02:03:39 | 2019-02-24 17:58:13 | Cashier         |    $9.64 |         198
    5022201 | 2019-01-21 04:59:52 | 2019-11-08 21:57:37 | Delivery Driver |   $16.06 |         371
    7977673 | 2018-12-16 17:30:47 |                     | Delivery Driver |   $11.65 |         334
    4454292 | 2018-07-20 08:56:04 | 2019-04-02 14:18:41 | Cashier         |   $20.41 |         226
    7014905 | 2019-01-29 06:25:20 | 2019-04-06 21:01:46 | Florist         |   $20.94 |         317
    2020317 | 2018-06-08 19:07:41 |                     | Delivery Driver |    $9.46 |         376
    9906786 | 2018-04-13 00:02:49 | 2019-10-12 17:14:47 | Florist         |   $20.65 |         205
    7949522 | 2018-05-12 03:40:22 | 2019-04-03 19:33:54 | Florist         |   $20.74 |         461
    1022115 | 2019-01-16 01:48:32 | 2019-12-06 06:05:44 | Delivery Driver |   $19.44 |         397
    8150009 | 2018-07-28 16:24:59 | 2019-09-16 22:26:01 | Cashier         |   $14.80 |         316
    2197602 | 2018-02-26 01:15:16 |                     | Delivery Driver |   $22.71 |         205
     794044 | 2018-07-06 21:25:38 | 2019-03-20 13:19:42 | Cashier         |   $12.63 |         448
    8708975 | 2018-04-07 06:07:04 | 2020-01-01 12:01:09 | Cashier         |   $17.78 |         137
    5789948 | 2018-05-24 14:36:53 | 2019-09-26 19:44:06 | Florist         |   $13.02 |         270
    1238167 | 2018-11-26 20:16:32 | 2019-07-02 21:01:53 | Delivery Driver |    $9.09 |         282
(30 rows)
```

Work Shift

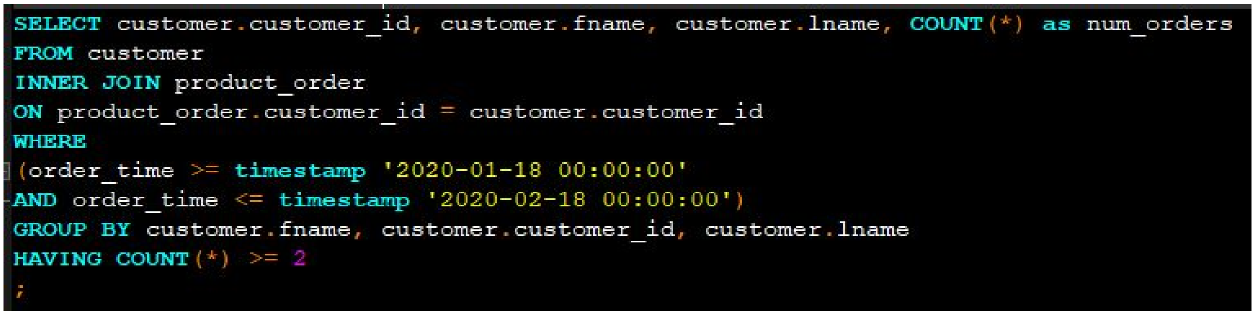


```
flowershop=# select * from work_shift;
 shift_id | employee_id | shift_date | begin_time | end_time
----------+-------------+------------+------------+----------
        1 |          27 | 2019-05-11 | 15:29:00   | 17:29:00
        2 |          20 | 2020-05-19 | 11:25:00   | 17:25:00
        3 |          11 | 2019-05-26 | 11:59:00   | 19:59:00
        4 |           1 | 2020-05-13 | 15:08:00   | 17:08:00
        5 |          29 | 2020-05-11 | 08:00:00   | 13:00:00
        6 |           6 | 2020-05-12 | 08:00:00   | 13:00:00
        7 |          15 | 2020-05-14 | 08:00:00   | 13:00:00
        8 |          19 | 2020-05-15 | 08:00:00   | 15:00:00
        9 |          15 | 2020-05-16 | 08:00:00   | 15:00:00
       10 |           1 | 2020-05-17 | 08:00:00   | 15:00:00
       11 |          30 | 2020-05-14 | 09:00:00   | 16:00:00
       12 |          23 | 2020-05-15 | 09:00:00   | 16:00:00
       13 |          23 | 2020-05-13 | 09:00:00   | 16:00:00
       14 |          16 | 2020-05-14 | 14:00:00   | 18:00:00
       15 |          11 | 2020-05-17 | 17:00:00   | 21:00:00
       16 |          20 | 2020-05-11 | 15:00:00   | 19:00:00
       17 |          11 | 2020-05-11 | 09:00:00   | 13:00:00
       18 |          24 | 2020-05-14 | 08:00:00   | 12:00:00
       20 |          11 | 2020-05-12 | 17:00:00   | 21:00:00
       21 |          21 | 2020-05-16 | 12:00:00   | 16:00:00
       23 |           1 | 2020-05-16 | 08:00:00   | 12:00:00
       24 |          28 | 2020-05-14 | 13:00:00   | 17:00:00
       25 |          19 | 2020-05-16 | 14:00:00   | 18:00:00
       26 |          15 | 2020-05-17 | 10:00:00   | 14:00:00
       27 |           5 | 2020-05-17 | 09:00:00   | 13:00:00
       28 |          13 | 2020-05-13 | 10:00:00   | 14:00:00
       29 |           2 | 2020-05-14 | 13:00:00   | 17:00:00
       30 |          24 | 2020-05-15 | 17:00:00   | 21:00:00
       33 |          20 | 2020-05-17 | 17:00:00   | 21:00:00
       34 |          28 | 2020-05-13 | 12:00:00   | 16:00:00
       35 |          21 | 2020-05-13 | 11:00:00   | 15:00:00
```

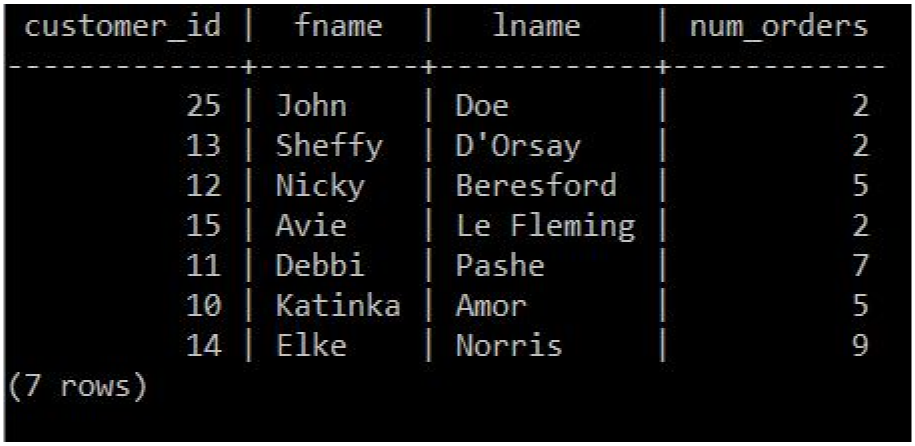
3.5 Queries in SQL

We will now present the SQL implementation for queries from section 2.4.

1. List customers who have made at least 2 product orders between 1/18/20 and 2/18/20.

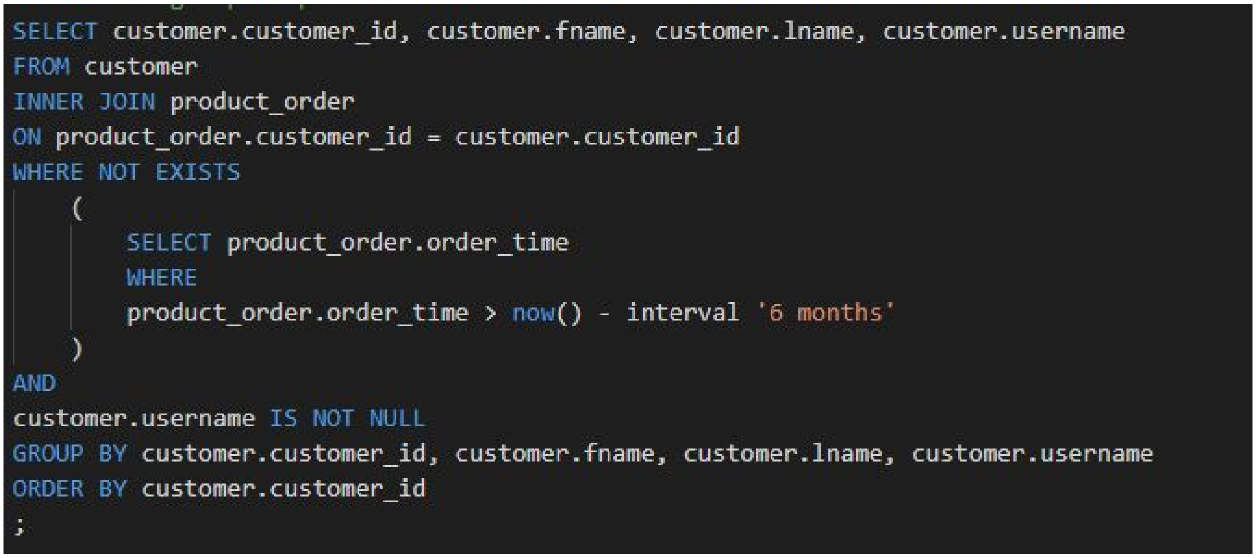


```
SELECT customer.customer_id, customer.fname, customer.lname, COUNT(*) as num_orders
FROM customer
INNER JOIN product_order
ON product_order.customer_id = customer.customer_id
WHERE
(order_time >= timestamp '2020-01-18 00:00:00'
AND order_time <= timestamp '2020-02-18 00:00:00')
GROUP BY customer.fname, customer.customer_id, customer.lname
HAVING COUNT(*) >= 2
;
```

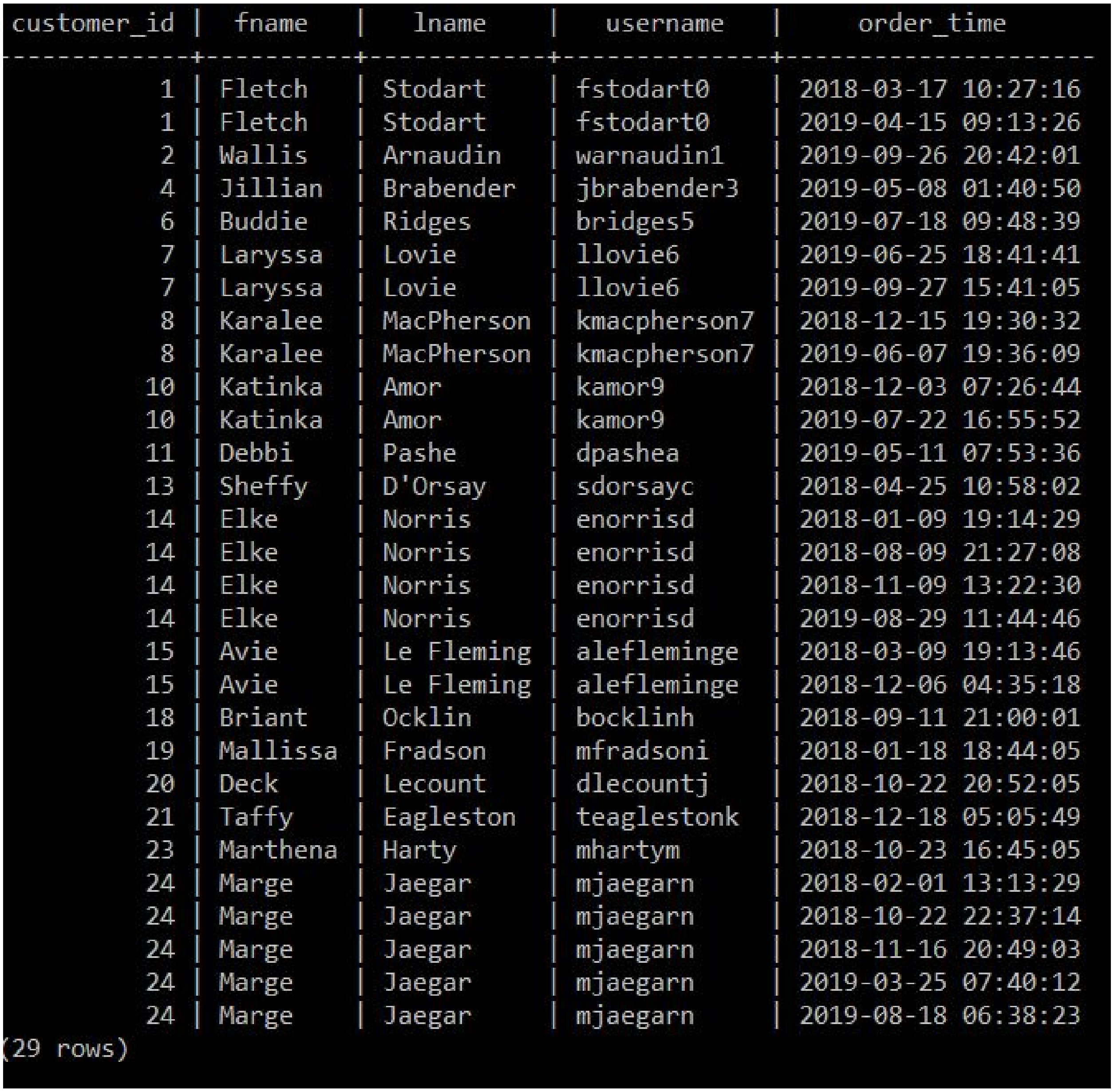


```
 customer_id |  fname   |    lname    | num_orders
-------------+----------+-------------+------------
          25 | John     | Doe         |          2
          13 | Sheffy   | D'Orsay     |          2
          12 | Nicky    | Beresford   |          5
          15 | Avie     | Le Fleming  |          2
          11 | Debbi    | Pashe       |          7
          10 | Katinka  | Amor        |          5
          14 | Elke     | Norris      |          9
(7 rows)
```

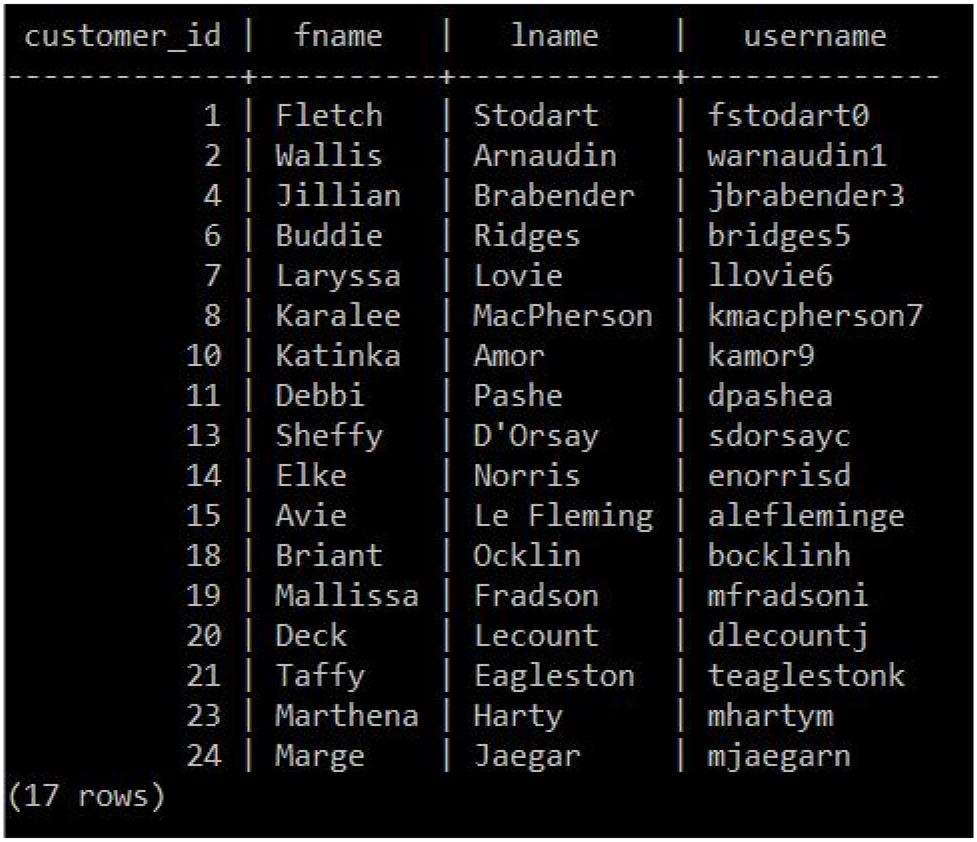
2. List customers with accounts on our website that have not made a product order in the past 6 months.



```
SELECT customer.customer_id, customer.fname, customer.lname, customer.username
FROM customer
INNER JOIN product_order
ON product_order.customer_id = customer.customer_id
WHERE NOT EXISTS
    (
        SELECT product_order.order_time
        WHERE
        product_order.order_time > now() - interval '6 months'
    )
AND
customer.username IS NOT NULL
GROUP BY customer.customer_id, customer.fname, customer.lname, customer.username
ORDER BY customer.customer_id
;
```

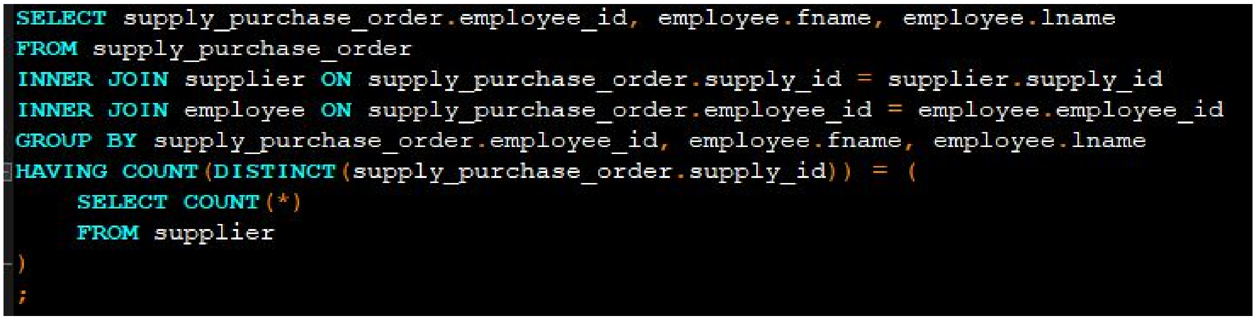


```
 customer_id |  fname   |   lname    |   username   |     order_time
-------------+----------+------------+--------------+---------------------
           1 | Fletch   | Stodart    | fstodart0    | 2018-03-17 10:27:16
           1 | Fletch   | Stodart    | fstodart0    | 2019-04-15 09:13:26
           2 | Wallis   | Arnaudin   | warnaudin1   | 2019-09-26 20:42:01
           4 | Jillian  | Brabender  | jbrabender3  | 2019-05-08 01:40:50
           6 | Buddie   | Ridges     | bridges5     | 2019-07-18 09:48:39
           7 | Laryssa  | Lovie      | llovie6      | 2019-06-25 18:41:41
           7 | Laryssa  | Lovie      | llovie6      | 2019-09-27 15:41:05
           8 | Karalee  | MacPherson | kmacpherson7 | 2018-12-15 19:30:32
           8 | Karalee  | MacPherson | kmacpherson7 | 2019-06-07 19:36:09
          10 | Katinka  | Amor       | kamor9       | 2018-12-03 07:26:44
          10 | Katinka  | Amor       | kamor9       | 2019-07-22 16:55:52
          11 | Debbi    | Pashe      | dpashea      | 2019-05-11 07:53:36
          13 | Sheffy   | D'Orsay    | sdorsayc     | 2018-04-25 10:58:02
          14 | Elke     | Norris     | enorrisd     | 2018-01-09 19:14:29
          14 | Elke     | Norris     | enorrisd     | 2018-08-09 21:27:08
          14 | Elke     | Norris     | enorrisd     | 2018-11-09 13:22:30
          14 | Elke     | Norris     | enorrisd     | 2019-08-29 11:44:46
          15 | Avie     | Le Fleming | alefleminge  | 2018-03-09 19:13:46
          15 | Avie     | Le Fleming | alefleminge  | 2018-12-06 04:35:18
          18 | Briant   | Ocklin     | bocklinh     | 2018-09-11 21:00:01
          19 | Mallissa | Fradson    | mfradsoni    | 2018-01-18 18:44:05
          20 | Deck     | Lecount    | dlecountj    | 2018-10-22 20:52:05
          21 | Taffy    | Eagleston  | teaglestonk  | 2018-12-18 05:05:49
          23 | Marthena | Harty      | mhartym      | 2018-10-23 16:45:05
          24 | Marge    | Jaegar     | mjaegarn     | 2018-02-01 13:13:29
          24 | Marge    | Jaegar     | mjaegarn     | 2018-10-22 22:37:14
          24 | Marge    | Jaegar     | mjaegarn     | 2018-11-16 20:49:03
          24 | Marge    | Jaegar     | mjaegarn     | 2019-03-25 07:40:12
          24 | Marge    | Jaegar     | mjaegarn     | 2019-08-18 06:38:23
(29 rows)
```

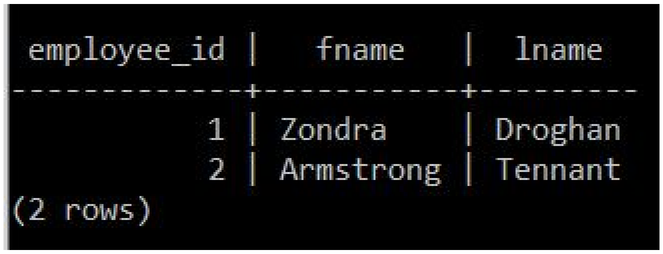


```
 customer_id |  fname    |   lname     |   username
-------------+-----------+-------------+--------------
           1 | Fletch    | Stodart     | fstodart0
           2 | Wallis    | Arnaudin    | warnaudin1
           4 | Jillian   | Brabender   | jbrabender3
           6 | Buddie    | Ridges      | bridges5
           7 | Laryssa   | Lovie       | llovie6
           8 | Karalee   | MacPherson  | kmacpherson7
          10 | Katinka   | Amor        | kamor9
          11 | Debbi     | Pashe       | dpashea
          13 | Sheffy    | D'Orsay     | sdorsayc
          14 | Elke      | Norris      | enorrisd
          15 | Avie      | Le Fleming  | alefleminge
          18 | Briant    | Ocklin      | bocklinh
          19 | Mallissa  | Fradson     | mfradsoni
          20 | Deck      | Lecount     | dlecountj
          21 | Taffy     | Eagleston   | teaglestonk
          23 | Marthena  | Harty       | mhartym
          24 | Marge     | Jaegar      | mjaegarn
(17 rows)
```

3. List employees who purchased flower products from every supplier.

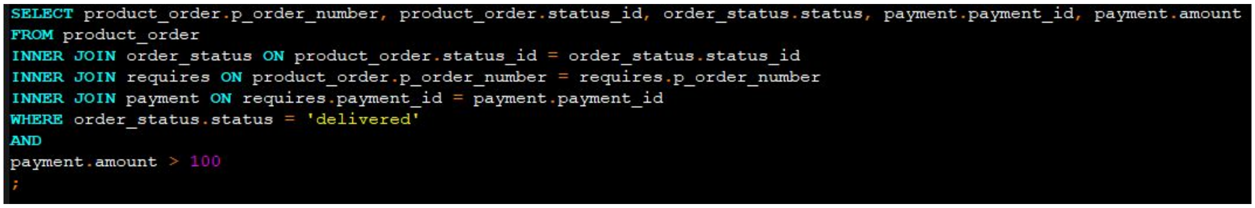


```
SELECT supply_purchase_order.employee_id, employee.fname, employee.lname
FROM supply_purchase_order
INNER JOIN supplier ON supply_purchase_order.supply_id = supplier.supply_id
INNER JOIN employee ON supply_purchase_order.employee_id = employee.employee_id
GROUP BY supply_purchase_order.employee_id, employee.fname, employee.lname
HAVING COUNT(DISTINCT(supply_purchase_order.supply_id)) = (
    SELECT COUNT(*)
    FROM supplier
)
;
```

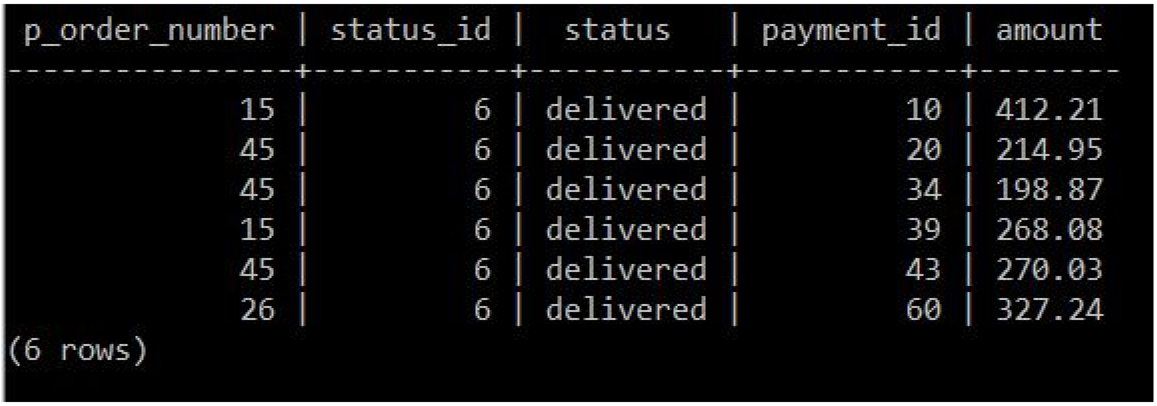


```
 employee_id |   fname    |  lname
-------------+------------+---------
           1 | Zondra     | Droghan
           2 | Armstrong  | Tennant
(2 rows)
```

4. List product orders with a payment greater than $100 that have been delivered.

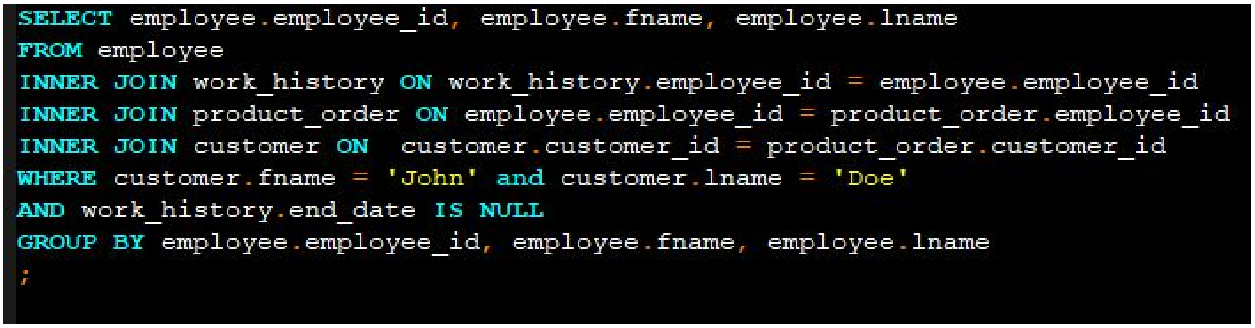


```
SELECT product_order.p_order_number, product_order.status_id, order_status.status, payment.payment_id, payment.amount
FROM product_order
INNER JOIN order_status ON product_order.status_id = order_status.status_id
INNER JOIN requires ON product_order.p_order_number = requires.p_order_number
INNER JOIN payment ON requires.payment_id = payment.payment_id
WHERE order_status.status = 'delivered'
AND
payment.amount > 100
;
```

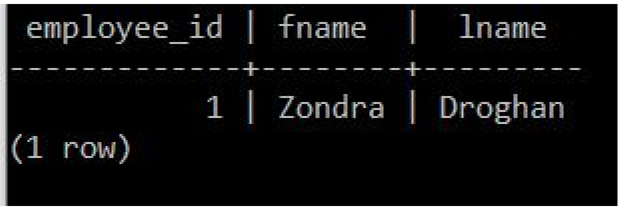


```
 p_order_number | status_id |   status    | payment_id | amount
----------------+-----------+-------------+------------+--------
             15 |         6 | delivered   |         10 | 412.21
             45 |         6 | delivered   |         20 | 214.95
             45 |         6 | delivered   |         34 | 198.87
             15 |         6 | delivered   |         39 | 268.08
             45 |         6 | delivered   |         43 | 270.03
             26 |         6 | delivered   |         60 | 327.24
(6 rows)
```

5. List current employees who have processed all John Doe’s purchases.

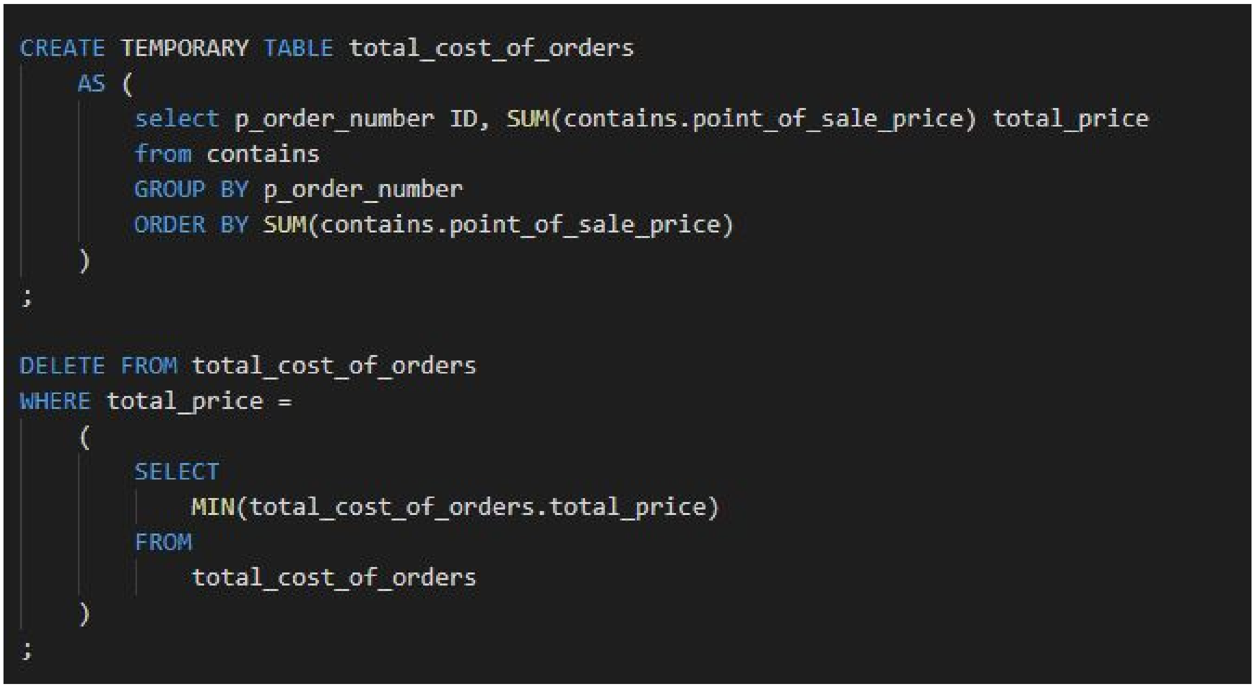


```
SELECT employee.employee_id, employee.fname, employee.lname
FROM employee
INNER JOIN work_history ON work_history.employee_id = employee.employee_id
INNER JOIN product_order ON employee.employee_id = product_order.employee_id
INNER JOIN customer ON  customer.customer_id = product_order.customer_id
WHERE customer.fname = 'John' and customer.lname = 'Doe'
AND work_history.end_date IS NULL
GROUP BY employee.employee_id, employee.fname, employee.lname
;
```

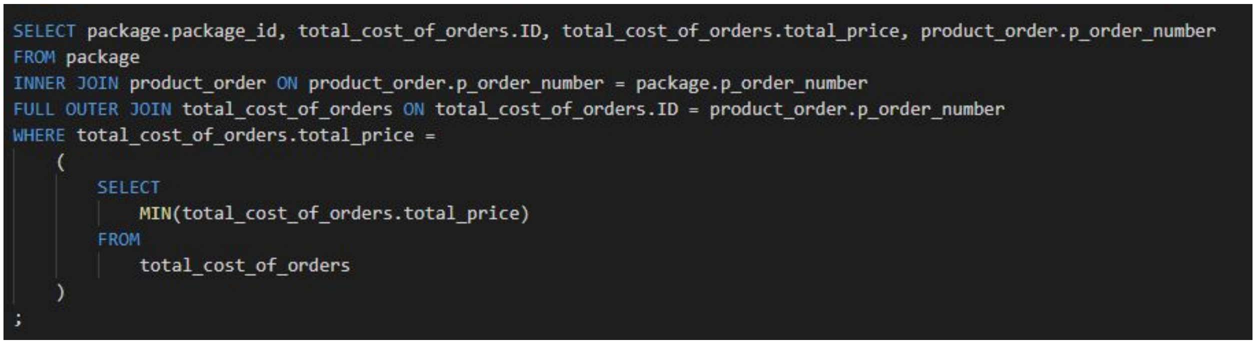


```
 employee_id | fname  |  lname
-------------+--------+---------
           1 | Zondra | Droghan
(1 row)
```

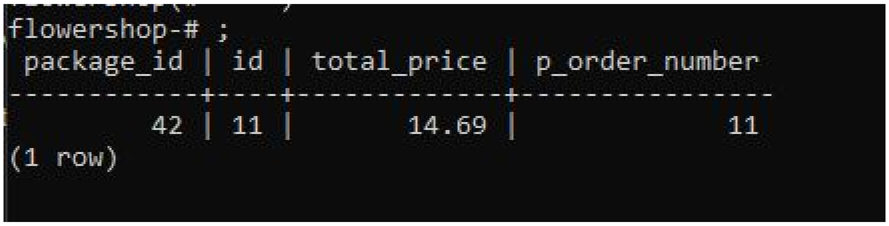
6. List the package(s) that has the second least expensive product order.



```
CREATE TEMPORARY TABLE total_cost_of_orders
    AS (
        select p_order_number ID, SUM(contains.point_of_sale_price) total_price
        from contains
        GROUP BY p_order_number
        ORDER BY SUM(contains.point_of_sale_price)
    )
;

DELETE FROM total_cost_of_orders
WHERE total_price =
    (
        SELECT
            MIN(total_cost_of_orders.total_price)
        FROM
            total_cost_of_orders
    )
;
```

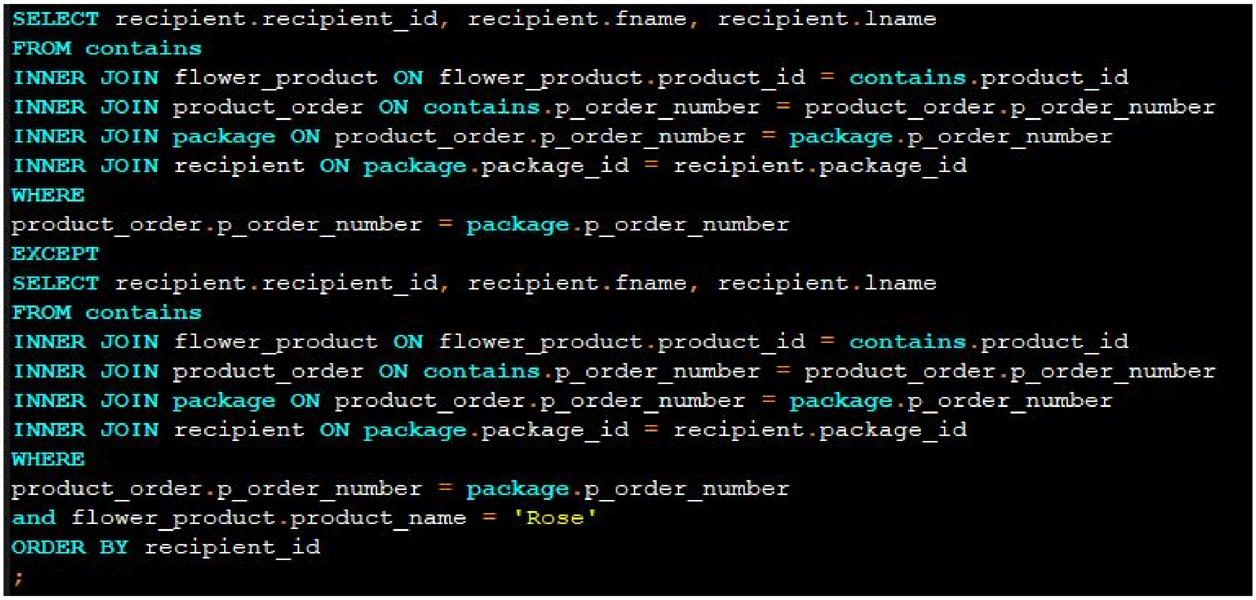


```
SELECT package.package_id, total_cost_of_orders.ID, total_cost_of_orders.total_price, product_order.p_order_number
FROM package
INNER JOIN product_order ON product_order.p_order_number = package.p_order_number
FULL OUTER JOIN total_cost_of_orders ON total_cost_of_orders.ID = product_order.p_order_number
WHERE total_cost_of_orders.total_price =
    (
        SELECT
            MIN(total_cost_of_orders.total_price)
        FROM
            total_cost_of_orders
    )
;
```

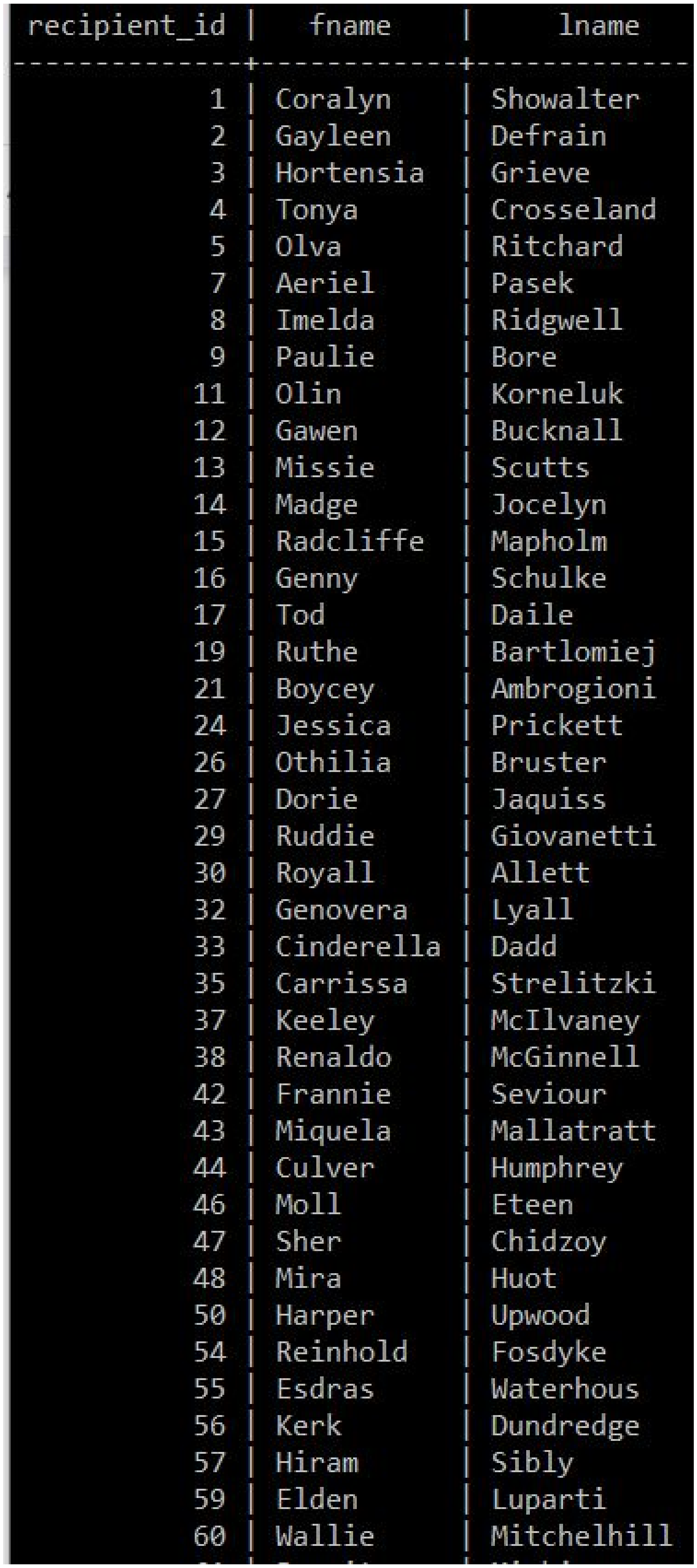


```
flowershop-# ;
 package_id | id | total_price | p_order_number
------------+----+-------------+----------------
         42 | 11 |       14.69 |             11
(1 row)
```

7. List recipients who have never received red roses.

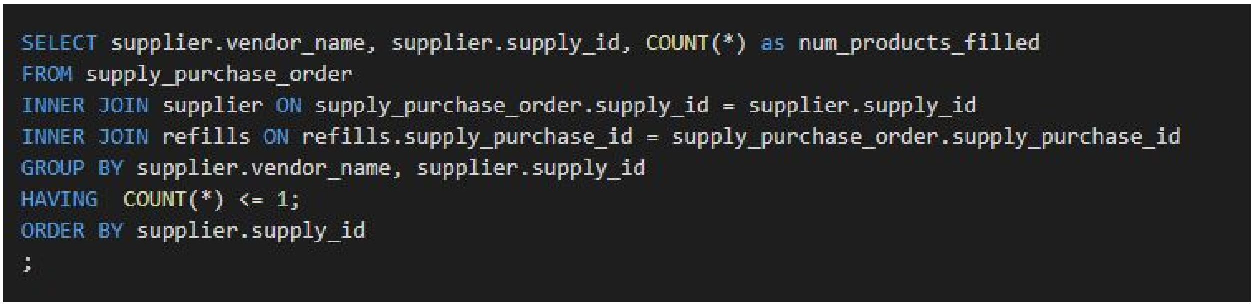


```
SELECT recipient.recipient_id, recipient.fname, recipient.lname
FROM contains
INNER JOIN flower_product ON flower_product.product_id = contains.product_id
INNER JOIN product_order ON contains.p_order_number = product_order.p_order_number
INNER JOIN package ON product_order.p_order_number = package.p_order_number
INNER JOIN recipient ON package.package_id = recipient.package_id
WHERE
product_order.p_order_number = package.p_order_number
EXCEPT
SELECT recipient.recipient_id, recipient.fname, recipient.lname
FROM contains
INNER JOIN flower_product ON flower_product.product_id = contains.product_id
INNER JOIN product_order ON contains.p_order_number = product_order.p_order_number
INNER JOIN package ON product_order.p_order_number = package.p_order_number
INNER JOIN recipient ON package.package_id = recipient.package_id
WHERE
product_order.p_order_number = package.p_order_number
and flower_product.product_name = 'Rose'
ORDER BY recipient_id
;
```

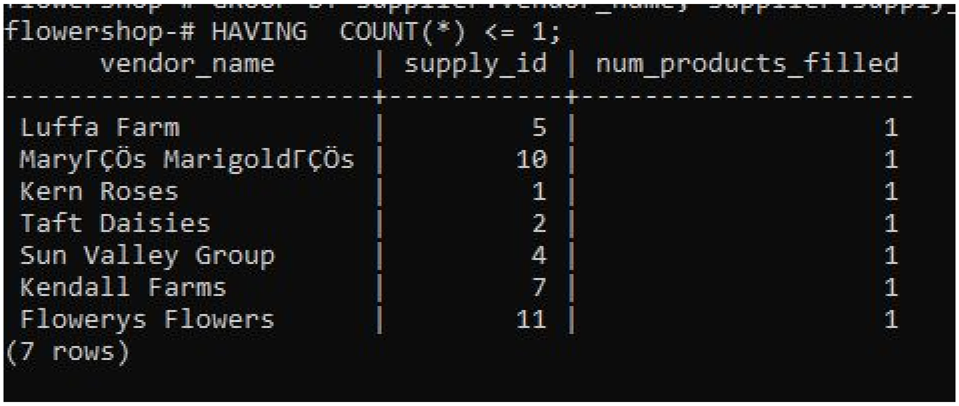


```
 recipient_id |   fname    |    lname
--------------+------------+-------------
            1 | Coralyn    | Showalter
            2 | Gayleen    | Defrain
            3 | Hortensia  | Grieve
            4 | Tonya      | Crosseland
            5 | Olva       | Ritchard
            7 | Aeriel     | Pasek
            8 | Imelda     | Ridgwell
            9 | Paulie     | Bore
           11 | Olin       | Korneluk
           12 | Gawen      | Bucknall
           13 | Missie     | Scutts
           14 | Madge      | Jocelyn
           15 | Radcliffe  | Mapholm
           16 | Genny      | Schulke
           17 | Tod        | Daile
           19 | Ruthe      | Bartlomiej
           21 | Boycey     | Ambrogioni
           24 | Jessica    | Prickett
           26 | Othilia    | Bruster
           27 | Dorie      | Jaquiss
           29 | Ruddie     | Giovanetti
           30 | Royall     | Allett
           32 | Genovera   | Lyall
           33 | Cinderella | Dadd
           35 | Carrissa   | Strelitzki
           37 | Keeley     | McIlvaney
           38 | Renaldo    | McGinnell
           42 | Frannie    | Seviour
           43 | Miquela    | Mallatratt
           44 | Culver     | Humphrey
           46 | Moll       | Eteen
           47 | Sher       | Chidzoy
           48 | Mira       | Huot
           50 | Harper     | Upwood
           54 | Reinhold   | Fosdyke
           55 | Esdras     | Waterhous
           56 | Kerk       | Dundredge
           57 | Hiram      | Sibly
           59 | Elden      | Luparti
           60 | Wallie     | Mitchelhill
```

8. List the suppliers that have no supply purchase order with more than 1 flower product.

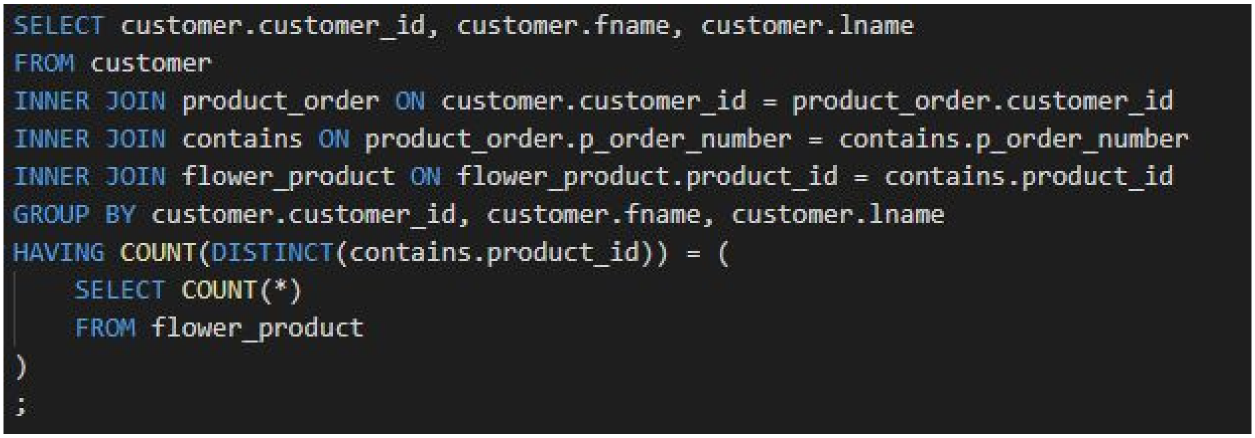


```
SELECT supplier.vendor_name, supplier.supply_id, COUNT(*) as num_products_filled
FROM supply_purchase_order
INNER JOIN supplier ON supply_purchase_order.supply_id = supplier.supply_id
INNER JOIN refills ON refills.supply_purchase_id = supply_purchase_order.supply_purchase_id
GROUP BY supplier.vendor_name, supplier.supply_id
HAVING  COUNT(*) <= 1;
ORDER BY supplier.supply_id
;
```

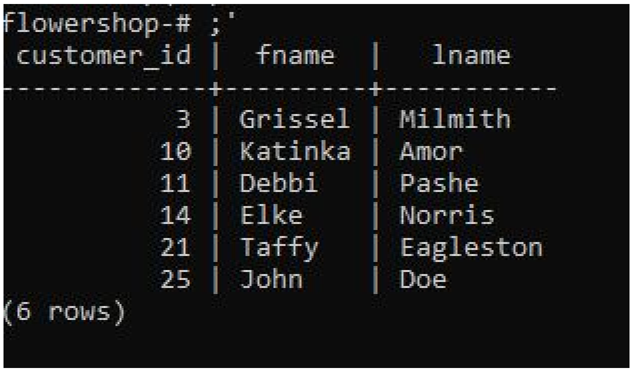


```
flowershop-# HAVING  COUNT(*) <= 1;
      vendor_name       | supply_id | num_products_filled
------------------------+-----------+---------------------
 Luffa Farm             |         5 |                   1
 MaryΓÇÖs MarigoldΓÇÖs |        10 |                   1
 Kern Roses             |         1 |                   1
 Taft Daisies           |         2 |                   1
 Sun Valley Group       |         4 |                   1
 Kendall Farms          |         7 |                   1
 Flowerys Flowers       |        11 |                   1
(7 rows)
```

9. List customers who have purchased all flower products.

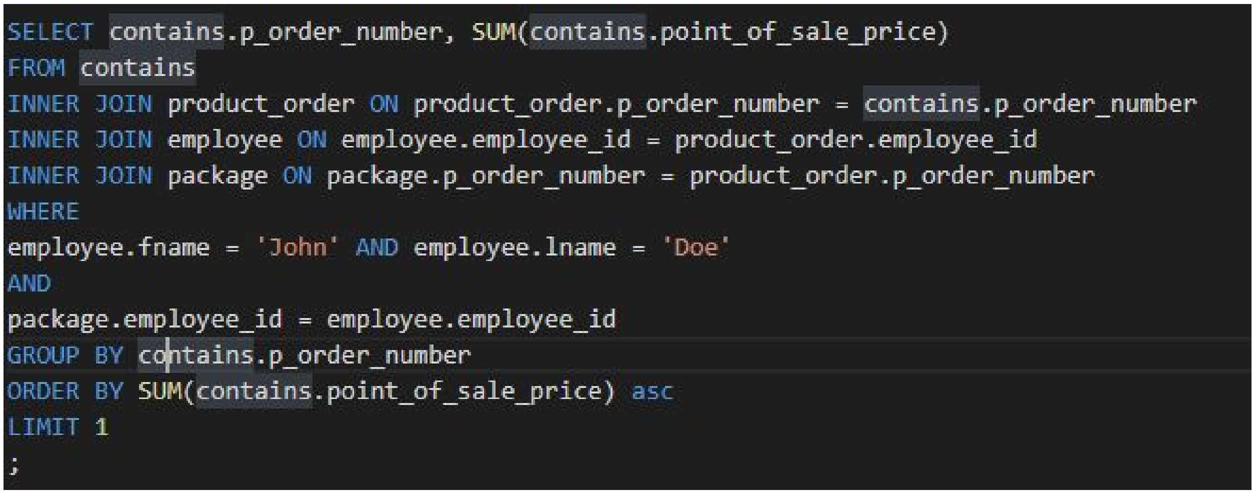


```
SELECT customer.customer_id, customer.fname, customer.lname
FROM customer
INNER JOIN product_order ON customer.customer_id = product_order.customer_id
INNER JOIN contains ON product_order.p_order_number = contains.p_order_number
INNER JOIN flower_product ON flower_product.product_id = contains.product_id
GROUP BY customer.customer_id, customer.fname, customer.lname
HAVING COUNT(DISTINCT(contains.product_id)) = (
    SELECT COUNT(*)
    FROM flower_product
)
;
```

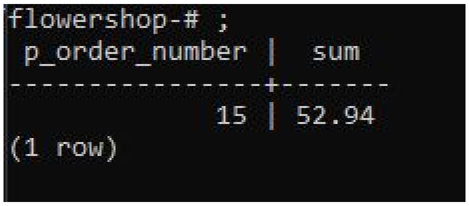


```
flowershop-# ;'
 customer_id |  fname   |   lname
-------------+----------+-----------
           3 | Grissel  | Milmith
          10 | Katinka  | Amor
          11 | Debbi    | Pashe
          14 | Elke     | Norris
          21 | Taffy    | Eagleston
          25 | John     | Doe
(6 rows)
```

10. List the cheapest package delivered by John Doe.



```
SELECT contains.p_order_number, SUM(contains.point_of_sale_price)
FROM contains
INNER JOIN product_order ON product_order.p_order_number = contains.p_order_number
INNER JOIN employee ON employee.employee_id = product_order.employee_id
INNER JOIN package ON package.p_order_number = product_order.p_order_number
WHERE
employee.fname = 'John' AND employee.lname = 'Doe'
AND
package.employee_id = employee.employee_id
GROUP BY contains.p_order_number
ORDER BY SUM(contains.point_of_sale_price) asc
LIMIT 1
;
```



```
flowershop-# ;
 p_order_number |  sum
----------------+-------
             15 | 52.94
(1 row)
```

Phase 4: DBMS Procedural Language & Stored Procedures and Triggers

Database Management System or DBMS is a system that manages a collection of databases. There exist different types of DBMS in the world. Some DBMS examples include PostgreSQL, MySQL, Oracle, and others. Every DBMS has similar yet different data structures and query language. Because there exists so many DBMS available, it is essential that there is a way that they can communicate with one another.

For various DBMS to communicate with each other, they have allowed one database to integrate with other databases meaning common SQL statements are translated from one program's syntax into a syntax that other databases can understand. In this phase, we will discuss the syntax of stored procedures and triggers for different DBMS including PostgreSQL. Similarities and differences among different DBMS procedural languages will be explained.

4.1 Postgres PL/pgsql

PL/pgSQL is known as Procedural Language/PostgreSQL and is a procedural programming language supported by the PostgreSQL DBMS that allows for much more procedural control than a standard SQL language. It is very similar to Oracle's own language, Procedural Language/SQL (PL/SQL). The overall goal of this language was to allow PostgreSQL users to be able to perform more complex operations and computations than SQL, while being easy to use and not cumbersome to the user.

4.1.1 Introduction to PL/pgsql

For PL/pgsql, its design goals are to construct a loadable procedural language that can be used to create functions and trigger procedures which run within Postgres. These functions are also known as stored procedures in other databases. PL/pgsql is a block-structured language for PostgreSQL. That would mean that a PL/pgSQL function is organized into blocks. With more procedural control than SQL, PL/pgSQL has the ability to use loops and other control structures.

4.1.2 Advantages of PL/pgsql

One of the benefits of using PL/pgSQL is that it has unique features built into the language to aid in managing the database. These features are used in the form of stored procedures/functions. The purpose of stored procedures is to perform actions without returning any result, this can include operations where data is inserted or updated. A few purposes of functions are to return one or more scalar values as OUT parameters or to return one or more results sets. The purpose of user-defined functions in PL/pgSQL is to process input parameters while returning new values.

There are many benefits of using stored procedures over sending SQL statements from front-end/client software to DBMS. These benefits include maintainability, testing can be independent of the application, stored procedures are already compiled on the server allowing the database to have increased speed, utilization of set-based processing and better security over sending SQL statements.

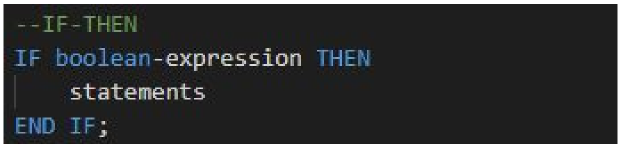
4.1.3 Control Statements and their Syntax

Control structure statements allow us to manipulate PostgresQL data in a very flexible way. These control statements include if-else, case, and loops. If and case are two conditional statements that can execute alternative commands under certain conditions.

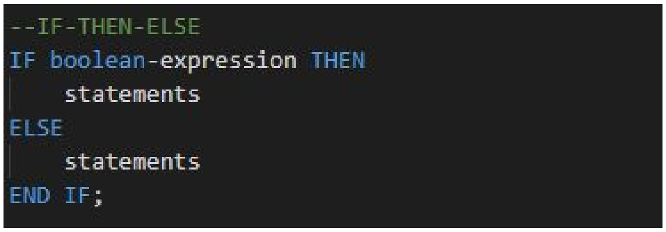
There are three forms of IF: if-then, if-then-else, and if-then-elsif. There are two forms of CASE: simple and searched.

Loops can arrange a PL/pgSQL function to repeat a series of commands. They can be used to repeat a number of different ways to achieve a certain task through repetition. One of those tasks can be executing a block of statements repeatedly until a condition becomes true. Loops can call conditional statements thus controlling the function's flow. PL/pgSQL provides us three forms of LOOPS: basic loop, while loop, and for loop.

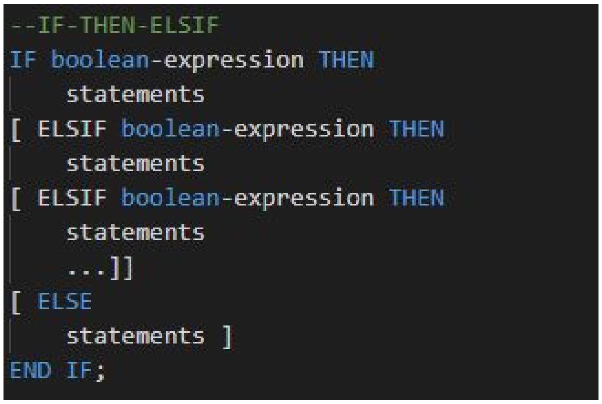
Syntax of Conditional Statements



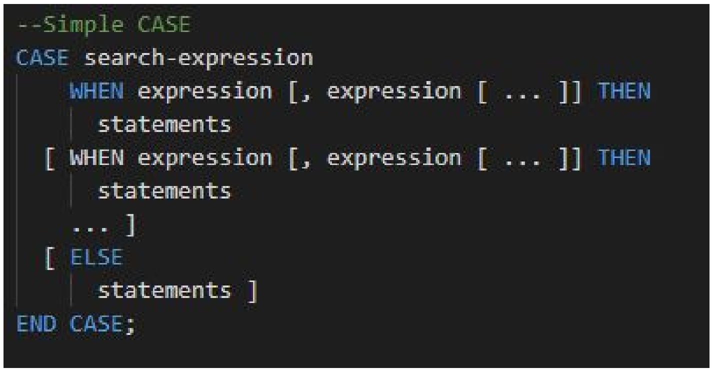
```
--IF-THEN
IF boolean-expression THEN
    statements
END IF;
```



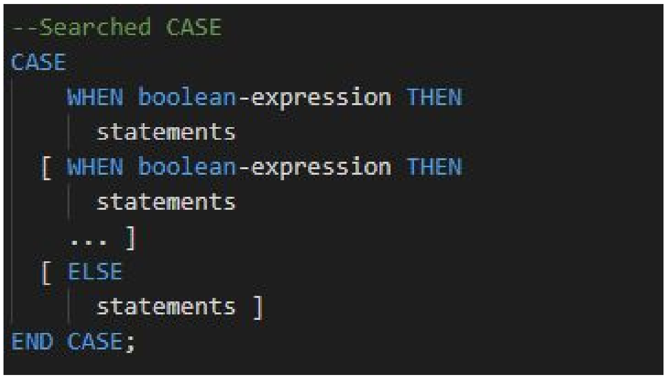
```
--IF-THEN-ELSE
IF boolean-expression THEN
    statements
ELSE
    statements
END IF;
```



```
--IF-THEN-ELSIF
IF boolean-expression THEN
    statements
[ ELSIF boolean-expression THEN
    statements
[ ELSIF boolean-expression THEN
    statements
    ...]]
[ ELSE
    statements ]
END IF;
```

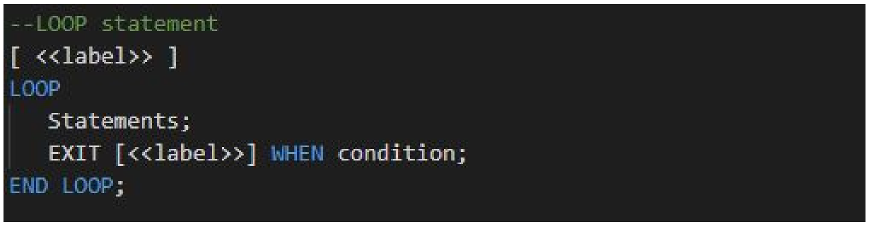


```
--Simple CASE
CASE search-expression
    WHEN expression [, expression [ ... ]] THEN
      statements
  [ WHEN expression [, expression [ ... ]] THEN
      statements
    ... ]
  [ ELSE
      statements ]
END CASE;
```

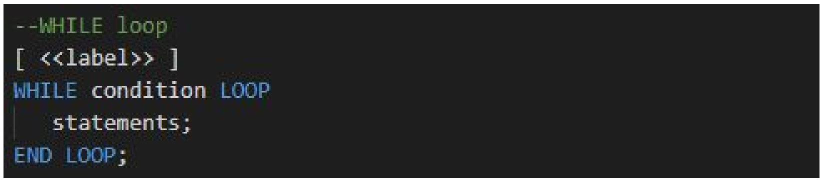


```
--Searched CASE
CASE
    WHEN boolean-expression THEN
      statements
  [ WHEN boolean-expression THEN
      statements
    ... ]
  [ ELSE
      statements ]
END CASE;
```

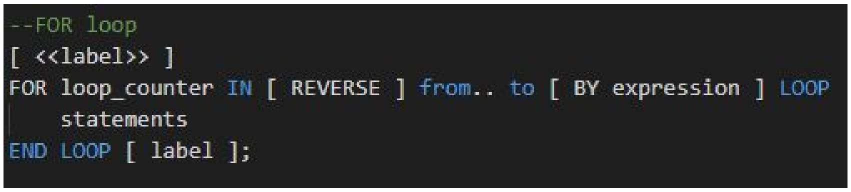
Syntax of Loops



```
--LOOP statement
[ <<label>> ]
LOOP
   Statements;
   EXIT [<<label>>] WHEN condition;
END LOOP;
```



```
--WHILE loop
[ <<label>> ]
WHILE condition LOOP
   statements;
END LOOP;
```

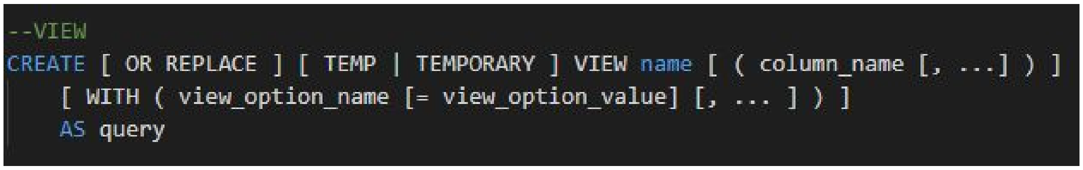


```
--FOR loop
[ <<label>> ]
FOR loop_counter IN [ REVERSE ] from.. to [ BY expression ] LOOP
    statements
END LOOP [ label ];
```

4.1.4 PL/pgSQL Syntax of VIEW, FUNCTION, PROCEDURE, TRIGGER

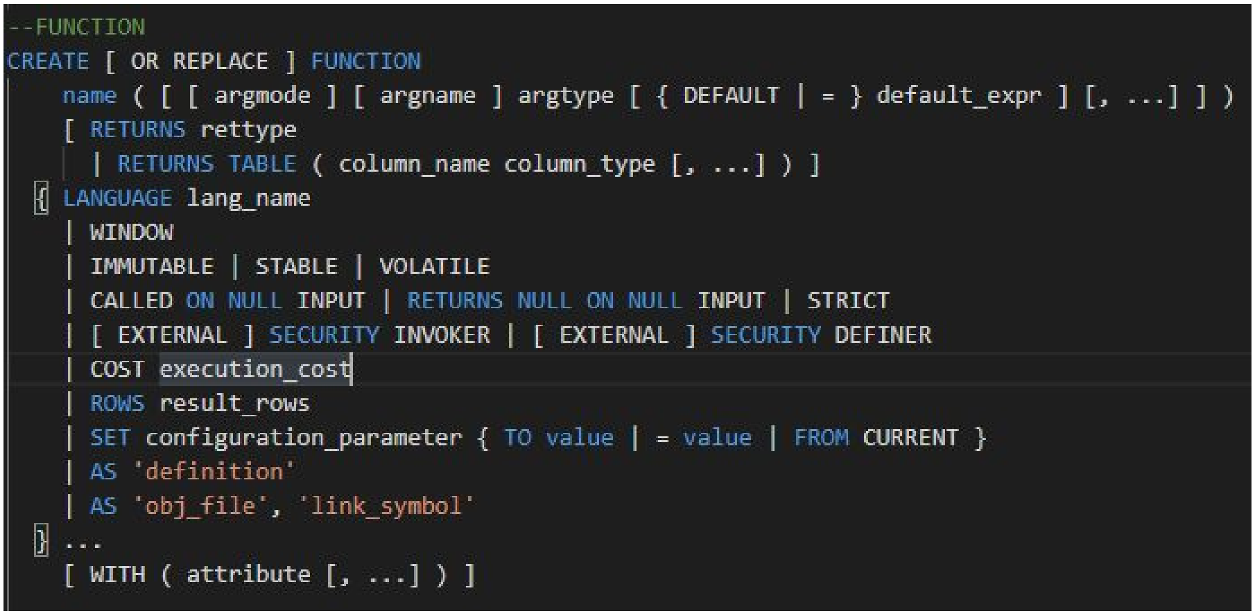
This section will go over the generalized syntax of views, functions, procedures, and trigger operations of PL/pgSQL. The following sections will provide examples of implementations of how we used these constructs in Bakersfield Flowershops database.

Syntax of View



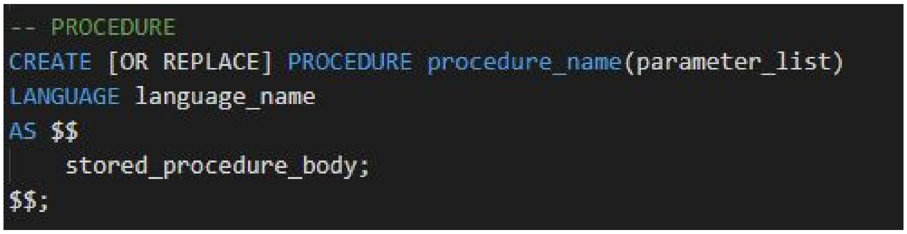
```
--VIEW
CREATE [ OR REPLACE ] [ TEMP | TEMPORARY ] VIEW name [ ( column_name [, ...] ) ]
    [ WITH ( view_option_name [= view_option_value] [, ... ] ) ]
    AS query
```

Syntax of Function



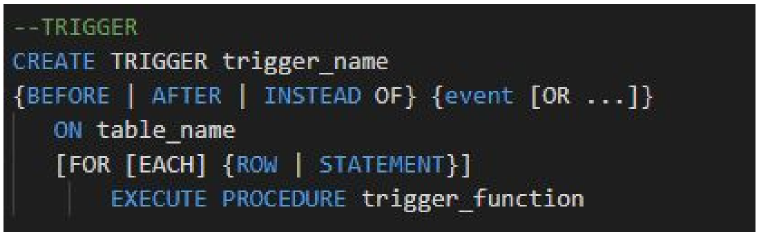
```
--FUNCTION
CREATE [ OR REPLACE ] FUNCTION
    name ( [ [ argmode ] [ argname ] argtype [ { DEFAULT | = } default_expr ] [, ...] ] )
    [ RETURNS rettype
      | RETURNS TABLE ( column_name column_type [, ...] ) ]
  { LANGUAGE lang_name
    | WINDOW
    | IMMUTABLE | STABLE | VOLATILE
    | CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT | STRICT
    | [ EXTERNAL ] SECURITY INVOKER | [ EXTERNAL ] SECURITY DEFINER
    | COST execution_cost
    | ROWS result_rows
    | SET configuration_parameter { TO value | = value | FROM CURRENT }
    | AS 'definition'
    | AS 'obj_file', 'link_symbol'
  } ...
    [ WITH ( attribute [, ...] ) ]
```

Syntax of Procedure



```
-- PROCEDURE
CREATE [OR REPLACE] PROCEDURE procedure_name(parameter_list)
LANGUAGE language_name
AS $$
    stored_procedure_body;
$$;
```

Syntax of Trigger



```
--TRIGGER
CREATE TRIGGER trigger_name
{BEFORE | AFTER | INSTEAD OF} {event [OR ...]}
   ON table_name
   [FOR [EACH] {ROW | STATEMENT}]
       EXECUTE PROCEDURE trigger_function
```

4.2 Views and Stored Subprograms of our Database

Views and stored subprograms allow a programmer to define more specific actions they would like the data in their database to interact. Views allow a programmer to create virtual tables that can simplify interacting with data. Stored subprograms can allow the programmer to define actions that need to take place in contexts of data entering the world the database represents.

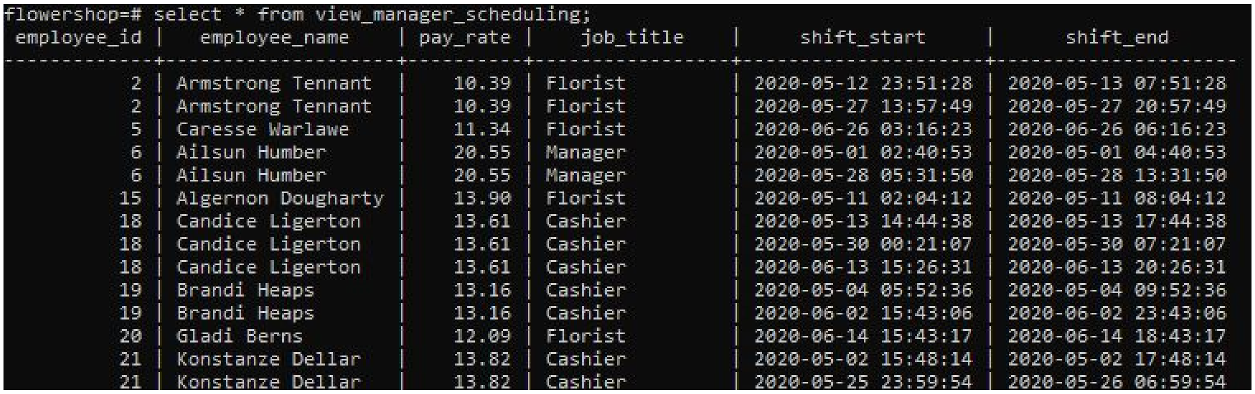
The first subsection will show the views we are using in Bakersfield Flowershop’s Database. The second subsection will show the procedures, functions, and triggers that we have built into our database. The last subsection will display the results of our procedures, functions, and triggers from our tables and views in our database.

4.2.1 Views

Views allow programmers to abstract away some of the complexity of their database. They are built from underlying tables into a virtual table. They are useful for simplifying complex queries and adding security to a database. A programmer can build views for user groups that only allow the user groups to interact with the data they need to.

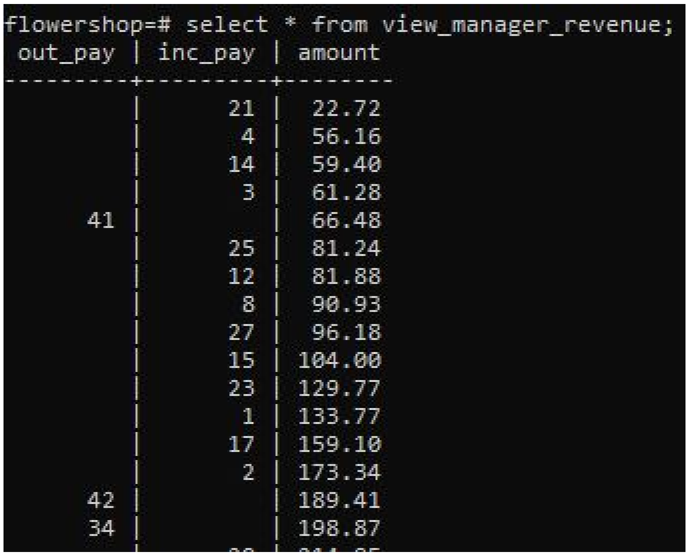
The user groups of our database are customers visiting our online store, an in store employee, and a manager for the database. This section will display the contents of the views in Bakersfield Flower Shops Database.

View_manager_scheduling



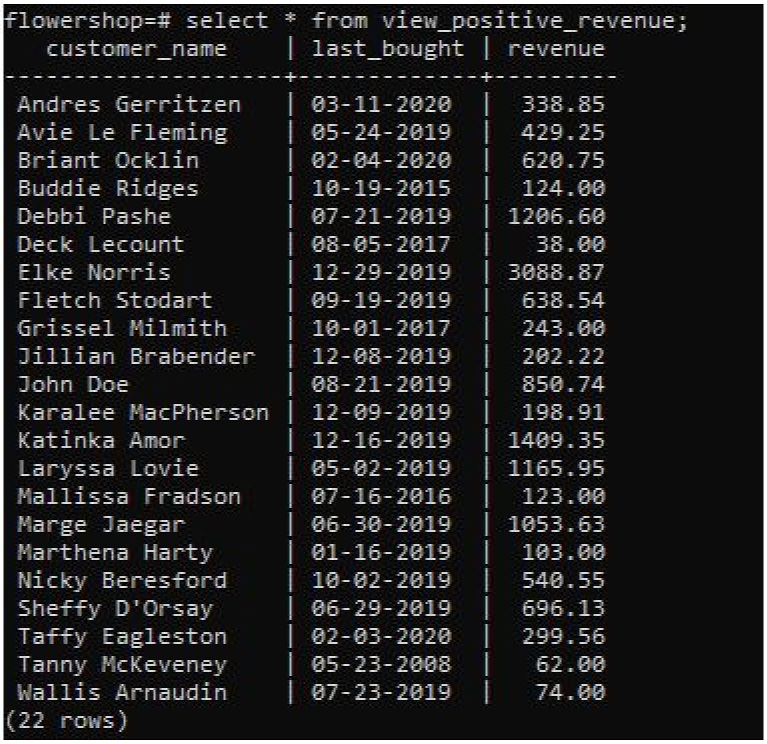
```
flowershop=# select * from view_manager_scheduling;
 employee_id |   employee_name    | pay_rate |    job_title     |      shift_start       |       shift_end
-------------+--------------------+----------+------------------+------------------------+----------------------
           2 | Armstrong Tennant  |    10.39 | Florist          | 2020-05-12 23:51:28 | 2020-05-13 07:51:28
           2 | Armstrong Tennant  |    10.39 | Florist          | 2020-05-27 13:57:49 | 2020-05-27 20:57:49
           5 | Caresse Warlawe    |    11.34 | Florist          | 2020-06-26 03:16:23 | 2020-06-26 06:16:23
           6 | Ailsun Humber      |    20.55 | Manager          | 2020-05-01 02:40:53 | 2020-05-01 04:40:53
           6 | Ailsun Humber      |    20.55 | Manager          | 2020-05-28 05:31:50 | 2020-05-28 13:31:50
          15 | Algernon Dougharty |    13.90 | Florist          | 2020-05-11 02:04:12 | 2020-05-11 08:04:12
          18 | Candice Ligerton   |    13.61 | Cashier          | 2020-05-13 14:44:38 | 2020-05-13 17:44:38
          18 | Candice Ligerton   |    13.61 | Cashier          | 2020-05-30 00:21:07 | 2020-05-30 07:21:07
          18 | Candice Ligerton   |    13.61 | Cashier          | 2020-06-13 15:26:31 | 2020-06-13 20:26:31
          19 | Brandi Heaps       |    13.16 | Cashier          | 2020-05-04 05:52:36 | 2020-05-04 09:52:36
          19 | Brandi Heaps       |    13.16 | Cashier          | 2020-06-02 15:43:06 | 2020-06-02 23:43:06
          20 | Gladi Berns        |    12.09 | Florist          | 2020-06-14 15:43:17 | 2020-06-14 18:43:17
          21 | Konstanze Dellar   |    13.82 | Cashier          | 2020-05-02 15:48:14 | 2020-05-02 17:48:14
          21 | Konstanze Dellar   |    13.82 | Cashier          | 2020-05-25 23:59:54 | 2020-05-26 06:59:54
```

View_manager_revenue



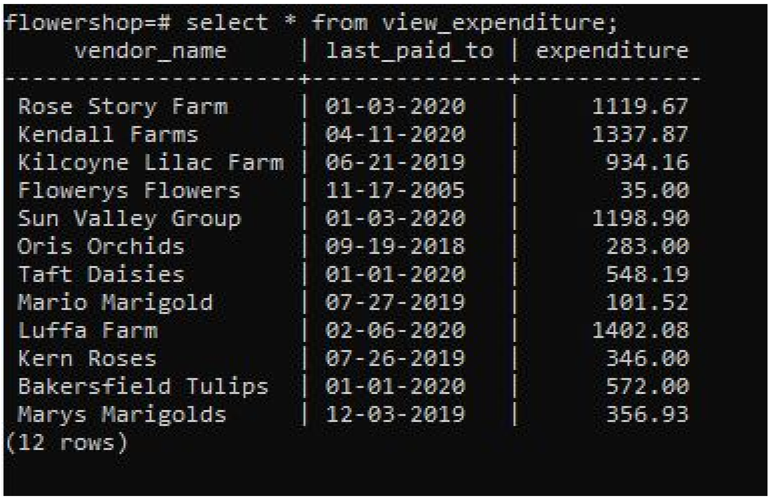
```
flowershop=# select * from view_manager_revenue;
 out_pay | inc_pay | amount
---------+---------+--------
         |      21 |  22.72
         |       4 |  56.16
         |      14 |  59.40
         |       3 |  61.28
      41 |         |  66.48
         |      25 |  81.24
         |      12 |  81.88
         |       8 |  90.93
         |      27 |  96.18
         |      15 | 104.00
         |      23 | 129.77
         |       1 | 133.77
         |      17 | 159.10
         |       2 | 173.34
      42 |         | 189.41
      34 |         | 198.87
```

View_positive_revenue



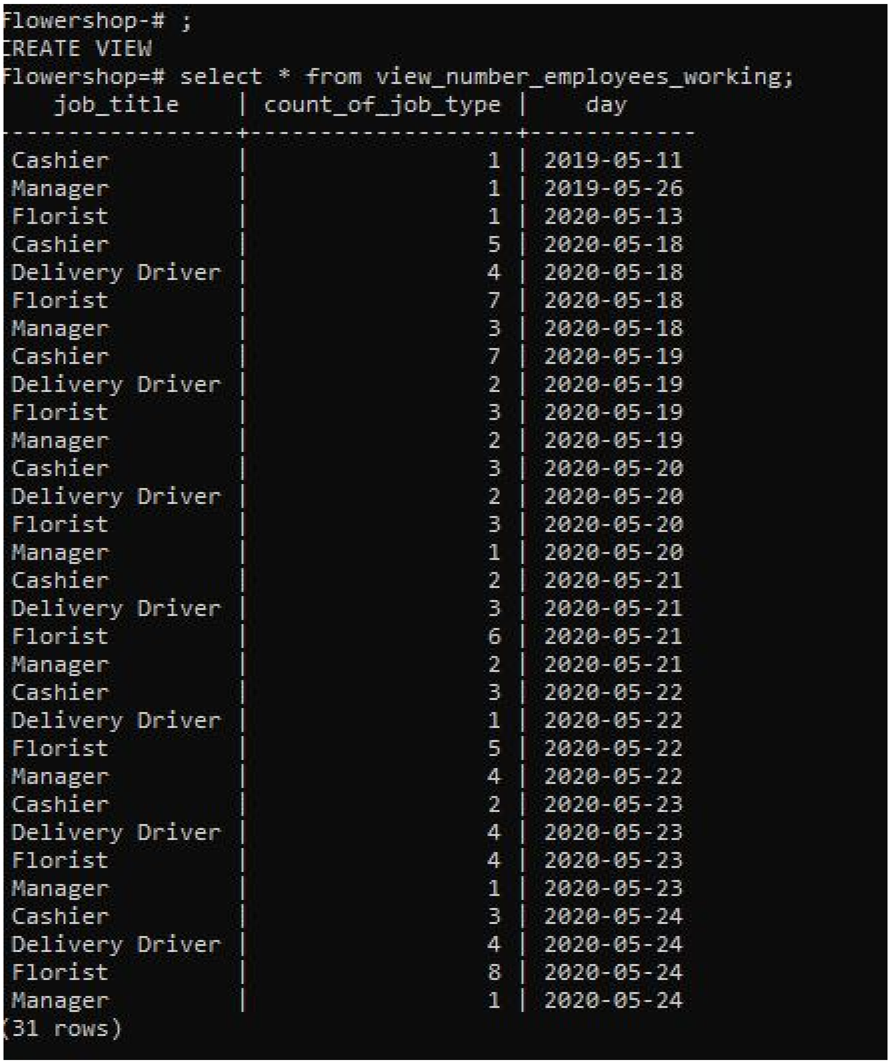
```
flowershop=# select * from view_positive_revenue;
   customer_name    | last_bought | revenue
--------------------+-------------+---------
 Andres Gerritzen   | 03-11-2020  |  338.85
 Avie Le Fleming    | 05-24-2019  |  429.25
 Briant Ocklin      | 02-04-2020  |  620.75
 Buddie Ridges      | 10-19-2015  |  124.00
 Debbi Pashe        | 07-21-2019  | 1206.60
 Deck Lecount       | 08-05-2017  |   38.00
 Elke Norris        | 12-29-2019  | 3088.87
 Fletch Stodart     | 09-19-2019  |  638.54
 Grissel Milmith    | 10-01-2017  |  243.00
 Jillian Brabender  | 12-08-2019  |  202.22
 John Doe           | 08-21-2019  |  850.74
 Karalee MacPherson | 12-09-2019  |  198.91
 Katinka Amor       | 12-16-2019  | 1409.35
 Laryssa Lovie      | 05-02-2019  | 1165.95
 Mallissa Fradson   | 07-16-2016  |  123.00
 Marge Jaegar       | 06-30-2019  | 1053.63
 Marthena Harty     | 01-16-2019  |  103.00
 Nicky Beresford    | 10-02-2019  |  540.55
 Sheffy D'Orsay     | 06-29-2019  |  696.13
 Taffy Eagleston    | 02-03-2020  |  299.56
 Tanny McKeveney    | 05-23-2008  |   62.00
 Wallis Arnaudin    | 07-23-2019  |   74.00
(22 rows)
```

View_expenditure



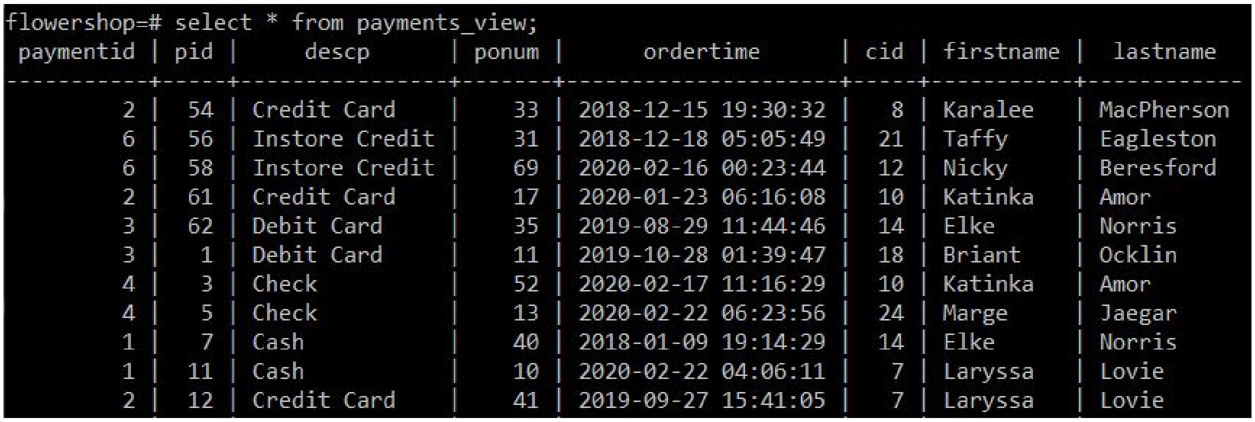
```
flowershop=# select * from view_expenditure;
     vendor_name     | last_paid_to | expenditure
---------------------+--------------+-------------
 Rose Story Farm     | 01-03-2020   |     1119.67
 Kendall Farms       | 04-11-2020   |     1337.87
 Kilcoyne Lilac Farm | 06-21-2019   |      934.16
 Flowerys Flowers    | 11-17-2005   |       35.00
 Sun Valley Group    | 01-03-2020   |     1198.90
 Oris Orchids        | 09-19-2018   |      283.00
 Taft Daisies        | 01-01-2020   |      548.19
 Mario Marigold      | 07-27-2019   |      101.52
 Luffa Farm          | 02-06-2020   |     1402.08
 Kern Roses          | 07-26-2019   |      346.00
 Bakersfield Tulips  | 01-01-2020   |      572.00
 Marys Marigolds     | 12-03-2019   |      356.93
(12 rows)
```

View_number_employees_working



```
flowershop-# ;
CREATE VIEW
flowershop=# select * from view_number_employees_working;
    job_title     | count_of_job_type |    day
------------------+-------------------+------------
 Cashier          |                 1 | 2019-05-11
 Manager          |                 1 | 2019-05-26
 Florist          |                 1 | 2020-05-13
 Cashier          |                 5 | 2020-05-18
 Delivery Driver  |                 4 | 2020-05-18
 Florist          |                 7 | 2020-05-18
 Manager          |                 3 | 2020-05-18
 Cashier          |                 7 | 2020-05-19
 Delivery Driver  |                 2 | 2020-05-19
 Florist          |                 3 | 2020-05-19
 Manager          |                 2 | 2020-05-19
 Cashier          |                 3 | 2020-05-20
 Delivery Driver  |                 2 | 2020-05-20
 Florist          |                 3 | 2020-05-20
 Manager          |                 1 | 2020-05-20
 Cashier          |                 2 | 2020-05-21
 Delivery Driver  |                 3 | 2020-05-21
 Florist          |                 6 | 2020-05-21
 Manager          |                 2 | 2020-05-21
 Cashier          |                 3 | 2020-05-22
 Delivery Driver  |                 1 | 2020-05-22
 Florist          |                 5 | 2020-05-22
 Manager          |                 4 | 2020-05-22
 Cashier          |                 2 | 2020-05-23
 Delivery Driver  |                 4 | 2020-05-23
 Florist          |                 4 | 2020-05-23
 Manager          |                 1 | 2020-05-23
 Cashier          |                 3 | 2020-05-24
 Delivery Driver  |                 4 | 2020-05-24
 Florist          |                 8 | 2020-05-24
 Manager          |                 1 | 2020-05-24
(31 rows)
```

Payments_view



```
flowershop=# select * from payments_view;
 paymentid | pid |      descp      | ponum |      ordertime      | cid | firstname |  lastname
-----------+-----+-----------------+-------+---------------------+-----+-----------+------------
         2 |  54 | Credit Card     |    33 | 2018-12-15 19:30:32 |   8 | Karalee   | MacPherson
         6 |  56 | Instore Credit  |    31 | 2018-12-18 05:05:49 |  21 | Taffy     | Eagleston
         6 |  58 | Instore Credit  |    69 | 2020-02-16 00:23:44 |  12 | Nicky     | Beresford
         2 |  61 | Credit Card     |    17 | 2020-01-23 06:16:08 |  10 | Katinka   | Amor
         3 |  62 | Debit Card      |    35 | 2019-08-29 11:44:46 |  14 | Elke      | Norris
         3 |   1 | Debit Card      |    11 | 2019-10-28 01:39:47 |  18 | Briant    | Ocklin
         4 |   3 | Check           |    52 | 2020-02-17 11:16:29 |  10 | Katinka   | Amor
         4 |   5 | Check           |    13 | 2020-02-22 06:23:56 |  24 | Marge     | Jaegar
         1 |   7 | Cash            |    40 | 2018-01-09 19:14:29 |  14 | Elke      | Norris
         1 |  11 | Cash            |    10 | 2020-02-22 04:06:11 |   7 | Laryssa   | Lovie
         2 |  12 | Credit Card     |    41 | 2019-09-27 15:41:05 |   7 | Laryssa   | Lovie
```

4.2.2 Stored Procedures and/or Functions

This section will contain three user defined procedures/functions and three triggers for Bakersfield Flowershop's database. The first procedure involves a procedure to insert a new product the shop will sell. The second deletes a customer from the customer table by their primary key.The last function will take the average of n number of cheapest products prices that Bakersfield Flowershop sells.

The three triggers following delete every record associated with a customer when the DELETE operation is called on them. The next trigger will update all tables where an employee's primary key appears if UPDATE is called altering the primary key of an existing employee. The last trigger involves a view in our database, and when data is altered in the view it will redirect the update in the underlying tables the view is pulling it's information from.

Insert Procedure

```
CREATE OR REPLACE PROCEDURE
insert_new_flower_product (
varchar,
decimal(12,2),
decimal(12,2),
VARCHAR(50),
DECIMAL(4,2),
VARCHAR(24),
VARCHAR(255))
LANGUAGE plpgsql
AS $$
BEGIN
    insert into flower_product(product_name, purchase_price,
     sell_price, color, length, product_image, description )
     values ($1, $2, $3, $4, $5, $6, $7);
    COMMIT;
END;
$$;
```

Delete Procedure

```
CREATE OR REPLACE PROCEDURE
remove_customer_record(
    Integer
)
LANGUAGE plpgsql
AS $$
BEGIN
    DELETE FROM customer
    WHERE customer_id = $1;
END;
$$;
```

Average Function

```
CREATE OR REPLACE FUNCTION average_of_products(integer)
RETURNS DECIMAL(4,2) AS $average$
DECLARE
    average DECIMAL(4,2);
BEGIN
    SELECT AVG (a.sell_price) INTO average FROM (
        SELECT sell_price FROM flower_product ORDER BY sell_price ASC
LIMIT $1
    ) AS a;
    RETURN average;
END;
$average$ LANGUAGE plpgsql;
```

Deletion Trigger

```
CREATE OR REPLACE FUNCTION remove_customer_records()
RETURNS TRIGGER as $BODY$
BEGIN
    DELETE FROM requires
    WHERE p_order_number = ANY(
        SELECT requires.p_order_number
        FROM requires
        INNER JOIN product_order ON product_order.p_order_number =
requires.p_order_number
        WHERE product_order.customer_id = OLD.customer_id
        );
    DELETE FROM recipient
    WHERE package_id = ANY(
        SELECT package.package_id
        FROM package
        INNER JOIN product_order ON product_order.p_order_number =
package.p_order_number
        INNER JOIN customer ON customer.customer_id =
product_order.customer_id
        WHERE product_order.customer_id = OLD.customer_id
    );
    DELETE FROM contains
    WHERE p_order_number = ANY(
        SELECT contains.p_order_number
        FROM contains
        INNER JOIN product_order ON contains.p_order_number =
product_order.p_order_number
        INNER JOIN customer ON customer.customer_id =
product_order.customer_id
        WHERE product_order.customer_id = OLD.customer_id
    );
    DELETE FROM package
    WHERE p_order_number = ANY(
        SELECT package.p_order_number
        FROM package
        INNER JOIN product_order ON product_order.p_order_number =
package.p_order_number
        WHERE product_order.customer_id = OLD.customer_id
    );

    DELETE FROM product_order WHERE customer_id = OLD.customer_id;
    RETURN OLD;
END;
$BODY$ LANGUAGE plpgsql;
```

```
DROP TRIGGER IF EXISTS remove_customer ON customer;
CREATE TRIGGER remove_customer
BEFORE DELETE ON customer
FOR EACH ROW EXECUTE PROCEDURE remove_customer_records();
```

Update Trigger

```
CREATE OR REPLACE FUNCTION update_employee_everywhere()
RETURNS trigger AS $BODY$
BEGIN
    -- Disable FK constraint just for trigger
    -- Not good idea normally
    ALTER TABLE product_order ALTER CONSTRAINT fk_order_employee
DEFERRABLE;
    ALTER TABLE package ALTER CONSTRAINT fk_package_employee_id
DEFERRABLE;
    ALTER TABLE work_history ALTER CONSTRAINT fk_employee_history
DEFERRABLE;
    ALTER TABLE supply_purchase_order ALTER CONSTRAINT
fk_purchase_order_employee DEFERRABLE;
    ALTER TABLE work_shift ALTER CONSTRAINT fk_employee_id DEFERRABLE;
    SET CONSTRAINTS fk_order_employee, fk_package_employee_id,
    fk_employee_history, fk_purchase_order_employee, fk_employee_id
DEFERRED;

    IF NEW.employee_id <> OLD.employee_id THEN
        UPDATE product_order SET employee_id = NEW.employee_id WHERE
        employee_id = OLD.employee_id;
        UPDATE package SET employee_id = NEW.employee_id WHERE
        employee_id = OLD.employee_id;
        UPDATE work_history  SET employee_id = NEW.employee_id WHERE
        employee_id = OLD.employee_id;
        UPDATE supply_purchase_order SET employee_id = NEW.employee_id
WHERE
```

```
        employee_id = OLD.employee_id;
        UPDATE work_shift SET employee_id = NEW.employee_id WHERE
        employee_id = OLD.employee_id;
    END IF;
    RETURN NEW;
    -- Fix FK constraint
    ALTER TABLE product_order ALTER CONSTRAINT fk_order_employee NOT
DEFERRABLE;
    ALTER TABLE package ALTER CONSTRAINT fk_package_employee_id NOT
DEFERRABLE;
    ALTER TABLE work_history ALTER CONSTRAINT fk_employee_history NOT
DEFERRABLE;
    ALTER TABLE supply_purchase_order ALTER CONSTRAINT
fk_purchase_order_employee NOT DEFERRABLE;
    ALTER TABLE work_shift ALTER CONSTRAINT fk_employee_id NOT DEFERRABLE;
END;
$BODY$ LANGUAGE plpgsql;
```

```
DROP TRIGGER IF EXISTS update_employee ON employee;
CREATE TRIGGER update_employee
BEFORE UPDATE ON employee
FOR EACH ROW EXECUTE PROCEDURE update_employee_everywhere();
```

Instead of Trigger

```
CREATE OR REPLACE FUNCTION update_employee_name()
RETURNS trigger AS $BODY$
BEGIN
    IF NEW.employee_name <> OLD.employee_name THEN
        UPDATE employee set fname = split_part(NEW.employee_name, ' ', 1),
        lname = split_part(NEW.employee_name, ' ', 2)
        WHERE employee_id = OLD.employee_id;
    END IF;

    RETURN NEW;
END;
```

```
$BODY$ LANGUAGE plpgsql;
```

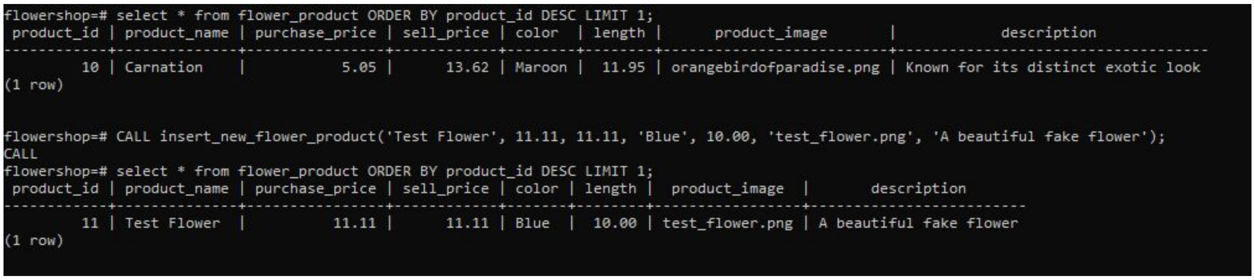
```
DROP TRIGGER IF EXISTS edit_employee_name ON view_manager_scheduling;
CREATE TRIGGER edit_employee_name
INSTEAD OF UPDATE ON view_manager_scheduling
FOR EACH ROW
    EXECUTE PROCEDURE update_employee_name();
```

4.2.3 Testing Results of Views, Functions and/or Procedures

This section will display the results of the altering of data using procedures and functions in the tables and views from the previous section. In the insert procedure we added a new product to our store that we sell. The delete procedure deletes an employee named ‘Fake Name’ by their primary key. The average function we pass in 6, which averages the price of the 6 cheapest products in our database.

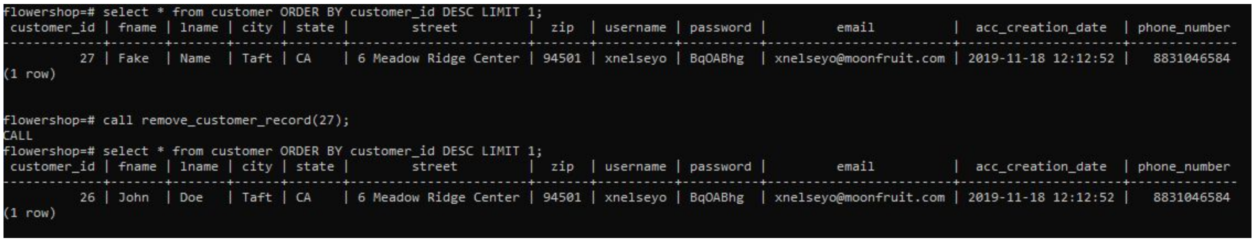
In our update trigger we change the primary keys of three employees in our database and cascade that update down to update all tables to match the new primary keys. The deletion trigger will remove all records associated with a customer when a DELETE is performed on a customer. The INSTEAD OF trigger we write takes an update to a view that changes an employee's name and redirects the update to change the name in our underlying employees table. The results of the instead of trigger affect the tables the view reads data from so the change will still be reflected in the view.

Insert Procedure Results



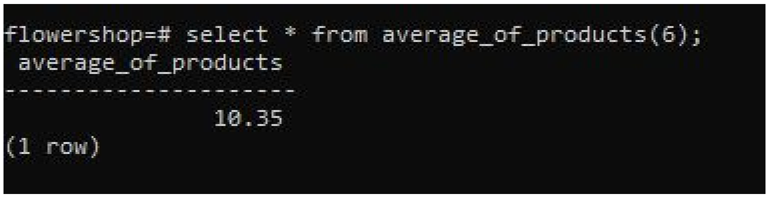
```
flowershop=# select * from flower_product ORDER BY product_id DESC LIMIT 1;
 product_id | product_name | purchase_price | sell_price | color  | length |      product_image       |            description
------------+--------------+----------------+------------+--------+--------+--------------------------+------------------------------------
         10 | Carnation    |           5.05 |      13.62 | Maroon |  11.95 | orangebirdofparadise.png | Known for its distinct exotic look
(1 row)


flowershop=# CALL insert_new_flower_product('Test Flower', 11.11, 11.11, 'Blue', 10.00, 'test_flower.png', 'A beautiful fake flower');
CALL
flowershop=# select * from flower_product ORDER BY product_id DESC LIMIT 1;
 product_id | product_name | purchase_price | sell_price | color | length |  product_image  |       description
------------+--------------+----------------+------------+-------+--------+-----------------+-------------------------
         11 | Test Flower  |          11.11 |      11.11 | Blue  |  10.00 | test_flower.png | A beautiful fake flower
(1 row)
```

Delete Procedure Results



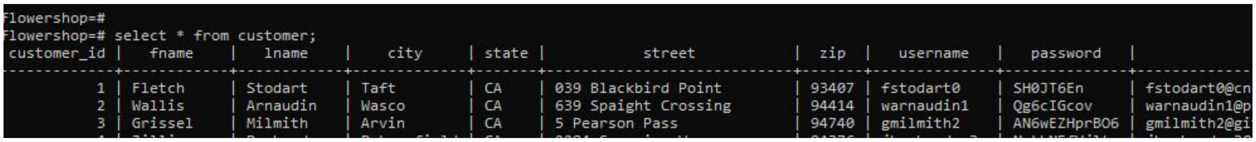
```
flowershop=# select * from customer ORDER BY customer_id DESC LIMIT 1;
 customer_id | fname | lname | city | state |        street         |  zip  | username | password |         email          |  acc_creation_date  | phone_number
-------------+-------+-------+------+-------+-----------------------+-------+----------+----------+------------------------+---------------------+--------------
          27 | Fake  | Name  | Taft | CA    | 6 Meadow Ridge Center | 94501 | xnelseyo | BqOABhg  | xnelseyo@moonfruit.com | 2019-11-18 12:12:52 |   8831046584
(1 row)


flowershop=# call remove_customer_record(27);
CALL
flowershop=# select * from customer ORDER BY customer_id DESC LIMIT 1;
 customer_id | fname | lname | city | state |        street         |  zip  | username | password |         email          |  acc_creation_date  | phone_number
-------------+-------+-------+------+-------+-----------------------+-------+----------+----------+------------------------+---------------------+--------------
          26 | John  | Doe   | Taft | CA    | 6 Meadow Ridge Center | 94501 | xnelseyo | BqOABhg  | xnelseyo@moonfruit.com | 2019-11-18 12:12:52 |   8831046584
(1 row)
```

Average Function Results

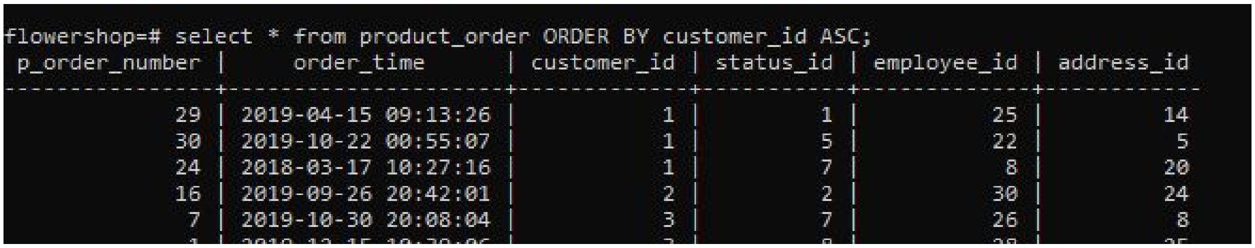


```
flowershop=# select * from average_of_products(6);
 average_of_products
---------------------
               10.35
(1 row)
```

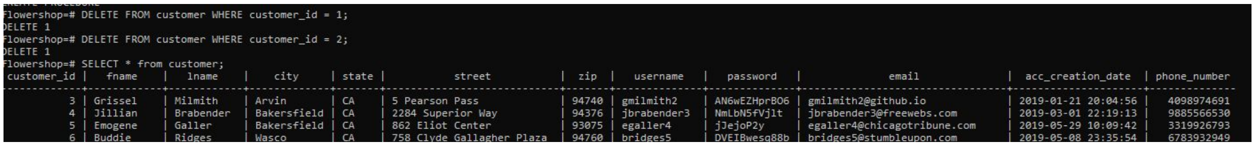
Deletion Trigger Results



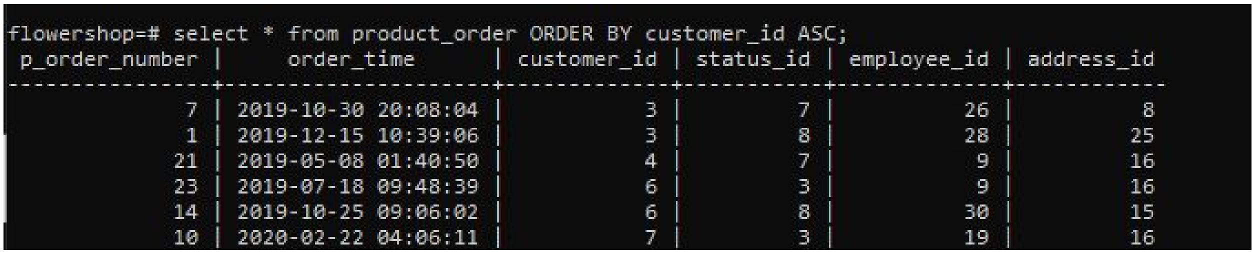
```
flowershop=#
flowershop=# select * from customer;
 customer_id |   fname    |   lname    |    city     | state |            street            |  zip  |   username   |   password   |
-------------+------------+------------+-------------+-------+------------------------------+-------+--------------+--------------+------------
           1 | Fletch     | Stodart    | Taft        | CA    | 039 Blackbird Point          | 93407 | fstodart0    | SH0JT6En     | fstodart0@cn
           2 | Wallis     | Arnaudin   | Wasco       | CA    | 639 Spaight Crossing         | 94414 | warnaudin1   | Qg6cIGcov    | warnaudin1@p
           3 | Grissel    | Milmith    | Arvin       | CA    | 5 Pearson Pass               | 94740 | gmilmith2    | AN6wEZHprBO6 | gmilmith2@gi
```



```
flowershop=# select * from product_order ORDER BY customer_id ASC;
 p_order_number |     order_time      | customer_id | status_id | employee_id | address_id
----------------+---------------------+-------------+-----------+-------------+------------
             29 | 2019-04-15 09:13:26 |           1 |         1 |          25 |         14
             30 | 2019-10-22 00:55:07 |           1 |         5 |          22 |          5
             24 | 2018-03-17 10:27:16 |           1 |         7 |           8 |         20
             16 | 2019-09-26 20:42:01 |           2 |         2 |          30 |         24
              7 | 2019-10-30 20:08:04 |           3 |         7 |          26 |          8
```

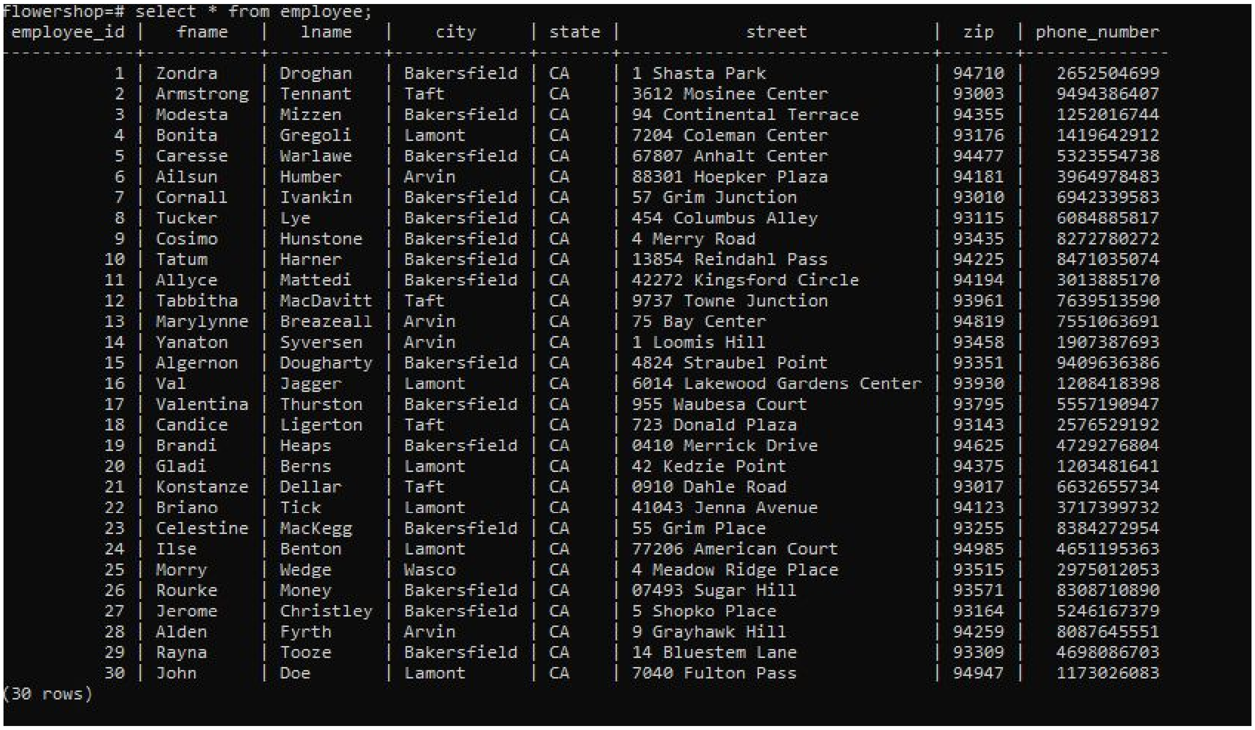


```
flowershop=# DELETE FROM customer WHERE customer_id = 1;
DELETE 1
flowershop=# DELETE FROM customer WHERE customer_id = 2;
DELETE 1
flowershop=# SELECT * from customer;
 customer_id |   fname    |   lname    |    city     | state |           street           |  zip  |   username   |   password   |             email             |  acc_creation_date  | phone_number
-------------+------------+------------+-------------+-------+----------------------------+-------+--------------+--------------+-------------------------------+---------------------+--------------
           3 | Grissel    | Milmith    | Arvin       | CA    | 5 Pearson Pass             | 94740 | gmilmith2    | AN6wEZHprBO6 | gmilmith2@github.io           | 2019-01-21 20:04:56 |   4098974691
           4 | Jillian    | Brabender  | Bakersfield | CA    | 2284 Superior Way          | 94376 | jbrabender3  | NmLbN5fVjlt  | jbrabender3@freewebs.com      | 2019-03-01 22:19:13 |   9885566530
           5 | Emogene    | Galler     | Bakersfield | CA    | 862 Eliot Center           | 93075 | egaller4     | jJejoP2y     | egaller4@chicagotribune.com   | 2019-05-29 10:09:42 |   3319926793
           6 | Buddie     | Ridges     | Wasco       | CA    | 758 Clyde Gallagher Plaza  | 94760 | bridges5     | DVEIBwesq88b | bridges5@stumbleupon.com      | 2019-05-08 23:35:54 |   6783932949
```

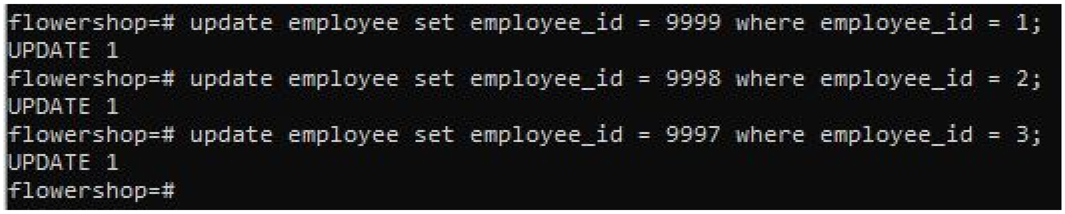


```
flowershop=# select * from product_order ORDER BY customer_id ASC;
 p_order_number |     order_time      | customer_id | status_id | employee_id | address_id
----------------+---------------------+-------------+-----------+-------------+------------
              7 | 2019-10-30 20:08:04 |           3 |         7 |          26 |          8
              1 | 2019-12-15 10:39:06 |           3 |         8 |          28 |         25
             21 | 2019-05-08 01:40:50 |           4 |         7 |           9 |         16
             23 | 2019-07-18 09:48:39 |           6 |         3 |           9 |         16
             14 | 2019-10-25 09:06:02 |           6 |         8 |          30 |         15
             10 | 2020-02-22 04:06:11 |           7 |         3 |          19 |         16
```

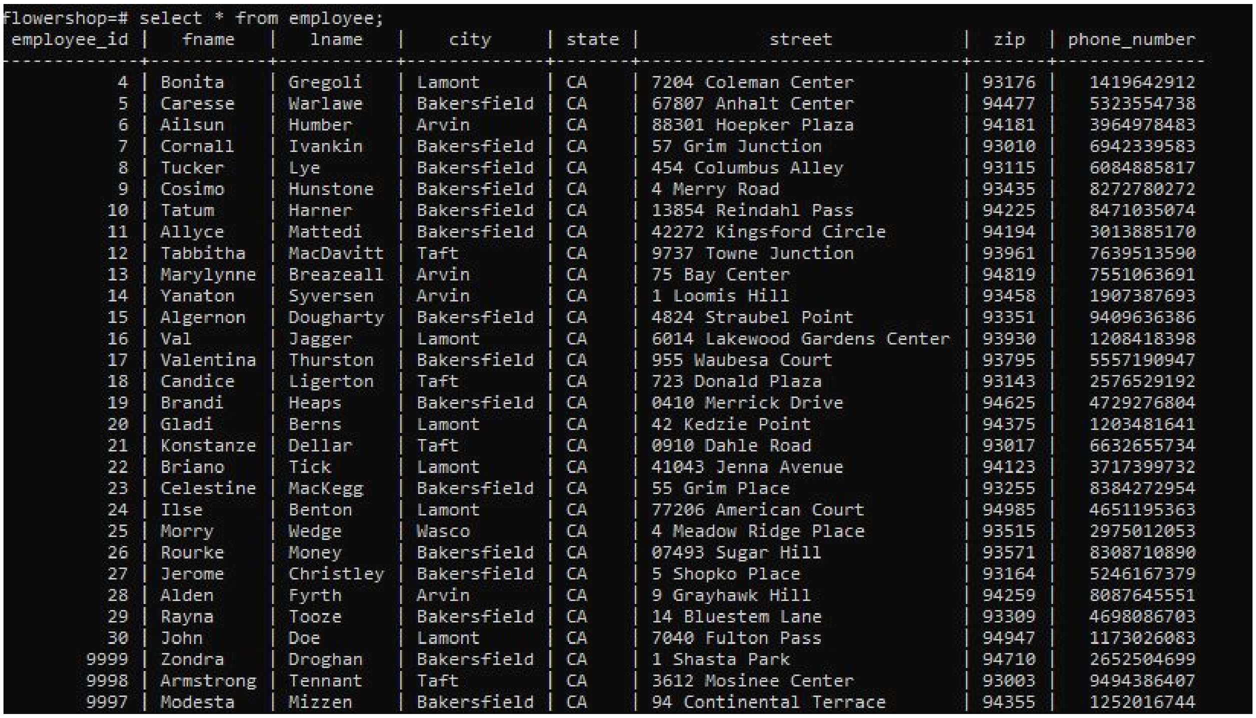
Update Trigger Results



```
flowershop=# select * from employee;
 employee_id |   fname    |   lname   |    city     | state |            street            |  zip  | phone_number
-------------+------------+-----------+-------------+-------+------------------------------+-------+--------------
           1 | Zondra     | Droghan   | Bakersfield | CA    | 1 Shasta Park                | 94710 |   2652504699
           2 | Armstrong  | Tennant   | Taft        | CA    | 3612 Mosinee Center          | 93003 |   9494386407
           3 | Modesta    | Mizzen    | Bakersfield | CA    | 94 Continental Terrace       | 94355 |   1252016744
           4 | Bonita     | Gregoli   | Lamont      | CA    | 7204 Coleman Center          | 93176 |   1419642912
           5 | Caresse    | Warlawe   | Bakersfield | CA    | 67807 Anhalt Center          | 94477 |   5323554738
           6 | Ailsun     | Humber    | Arvin       | CA    | 88301 Hoepker Plaza          | 94181 |   3964978483
           7 | Cornall    | Ivankin   | Bakersfield | CA    | 57 Grim Junction             | 93010 |   6942339583
           8 | Tucker     | Lye       | Bakersfield | CA    | 454 Columbus Alley           | 93115 |   6084885817
           9 | Cosimo     | Hunstone  | Bakersfield | CA    | 4 Merry Road                 | 93435 |   8272780272
          10 | Tatum      | Harner    | Bakersfield | CA    | 13854 Reindahl Pass          | 94225 |   8471035074
          11 | Allyce     | Mattedi   | Bakersfield | CA    | 42272 Kingsford Circle       | 94194 |   3013885170
          12 | Tabbitha   | MacDavitt | Taft        | CA    | 9737 Towne Junction          | 93961 |   7639513590
          13 | Marylynne  | Breazeall | Arvin       | CA    | 75 Bay Center                | 94819 |   7551063691
          14 | Yanaton    | Syversen  | Arvin       | CA    | 1 Loomis Hill                | 93458 |   1907387693
          15 | Algernon   | Dougharty | Bakersfield | CA    | 4824 Straubel Point          | 93351 |   9409636386
          16 | Val        | Jagger    | Lamont      | CA    | 6014 Lakewood Gardens Center | 93930 |   1208418398
          17 | Valentina  | Thurston  | Bakersfield | CA    | 955 Waubesa Court            | 93795 |   5557190947
          18 | Candice    | Ligerton  | Taft        | CA    | 723 Donald Plaza             | 93143 |   2576529192
          19 | Brandi     | Heaps     | Bakersfield | CA    | 0410 Merrick Drive           | 94625 |   4729276804
          20 | Gladi      | Berns     | Lamont      | CA    | 42 Kedzie Point              | 94375 |   1203481641
          21 | Konstanze  | Dellar    | Taft        | CA    | 0910 Dahle Road              | 93017 |   6632655734
          22 | Briano     | Tick      | Lamont      | CA    | 41043 Jenna Avenue           | 94123 |   3717399732
          23 | Celestine  | MacKegg   | Bakersfield | CA    | 55 Grim Place                | 93255 |   8384272954
          24 | Ilse       | Benton    | Lamont      | CA    | 77206 American Court         | 94985 |   4651195363
          25 | Morry      | Wedge     | Wasco       | CA    | 4 Meadow Ridge Place         | 93515 |   2975012053
          26 | Rourke     | Money     | Bakersfield | CA    | 07493 Sugar Hill             | 93571 |   8308710890
          27 | Jerome     | Christley | Bakersfield | CA    | 5 Shopko Place               | 93164 |   5246167379
          28 | Alden      | Fyrth     | Arvin       | CA    | 9 Grayhawk Hill              | 94259 |   8087645551
          29 | Rayna      | Tooze     | Bakersfield | CA    | 14 Bluestem Lane             | 93309 |   4698086703
          30 | John       | Doe       | Lamont      | CA    | 7040 Fulton Pass             | 94947 |   1173026083
(30 rows)
```

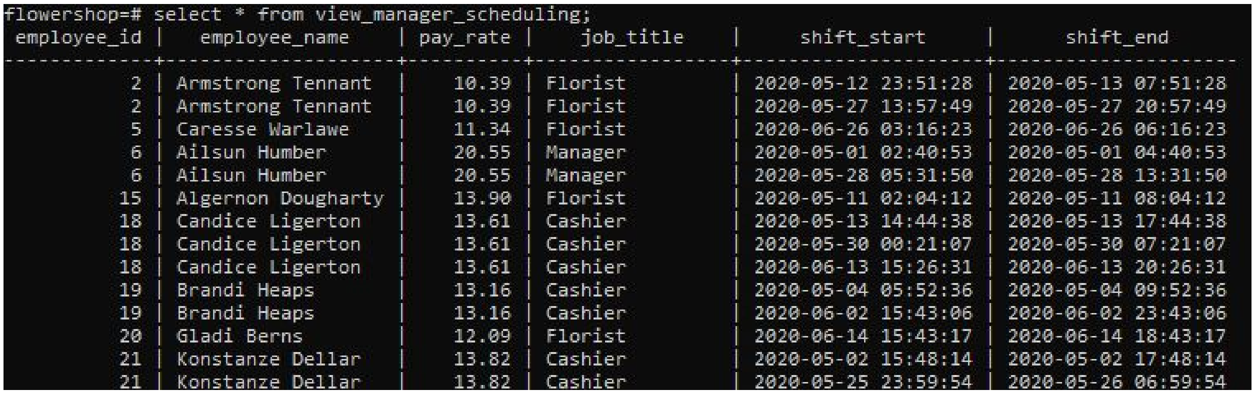


```
flowershop=# update employee set employee_id = 9999 where employee_id = 1;
UPDATE 1
flowershop=# update employee set employee_id = 9998 where employee_id = 2;
UPDATE 1
flowershop=# update employee set employee_id = 9997 where employee_id = 3;
UPDATE 1
flowershop=#
```

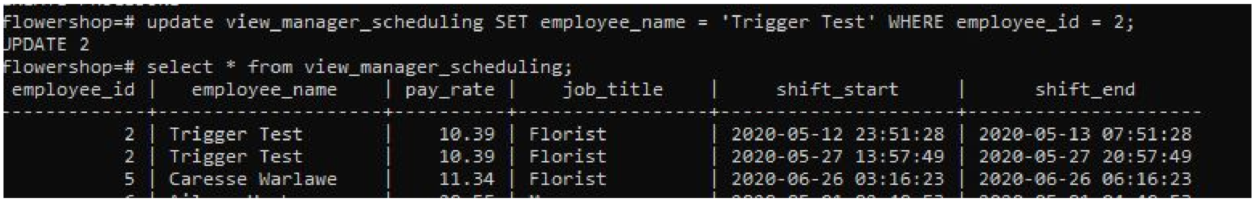


```
flowershop=# select * from employee;
 employee_id |   fname    |   lname   |    city     | state |            street            |  zip  | phone_number
-------------+------------+-----------+-------------+-------+------------------------------+-------+--------------
           4 | Bonita     | Gregoli   | Lamont      | CA    | 7204 Coleman Center          | 93176 |   1419642912
           5 | Caresse    | Warlawe   | Bakersfield | CA    | 67807 Anhalt Center          | 94477 |   5323554738
           6 | Ailsun     | Humber    | Arvin       | CA    | 88301 Hoepker Plaza          | 94181 |   3964978483
           7 | Cornall    | Ivankin   | Bakersfield | CA    | 57 Grim Junction             | 93010 |   6942339583
           8 | Tucker     | Lye       | Bakersfield | CA    | 454 Columbus Alley           | 93115 |   6084885817
           9 | Cosimo     | Hunstone  | Bakersfield | CA    | 4 Merry Road                 | 93435 |   8272780272
          10 | Tatum      | Harner    | Bakersfield | CA    | 13854 Reindahl Pass          | 94225 |   8471035074
          11 | Allyce     | Mattedi   | Bakersfield | CA    | 42272 Kingsford Circle       | 94194 |   3013885170
          12 | Tabbitha   | MacDavitt | Taft        | CA    | 9737 Towne Junction          | 93961 |   7639513590
          13 | Marylynne  | Breazeall | Arvin       | CA    | 75 Bay Center                | 94819 |   7551063691
          14 | Yanaton    | Syversen  | Arvin       | CA    | 1 Loomis Hill                | 93458 |   1907387693
          15 | Algernon   | Dougharty | Bakersfield | CA    | 4824 Straubel Point          | 93351 |   9409636386
          16 | Val        | Jagger    | Lamont      | CA    | 6014 Lakewood Gardens Center | 93930 |   1208418398
          17 | Valentina  | Thurston  | Bakersfield | CA    | 955 Waubesa Court            | 93795 |   5557190947
          18 | Candice    | Ligerton  | Taft        | CA    | 723 Donald Plaza             | 93143 |   2576529192
          19 | Brandi     | Heaps     | Bakersfield | CA    | 0410 Merrick Drive           | 94625 |   4729276804
          20 | Gladi      | Berns     | Lamont      | CA    | 42 Kedzie Point              | 94375 |   1203481641
          21 | Konstanze  | Dellar    | Taft        | CA    | 0910 Dahle Road              | 93017 |   6632655734
          22 | Briano     | Tick      | Lamont      | CA    | 41043 Jenna Avenue           | 94123 |   3717399732
          23 | Celestine  | MacKegg   | Bakersfield | CA    | 55 Grim Place                | 93255 |   8384272954
          24 | Ilse       | Benton    | Lamont      | CA    | 77206 American Court         | 94985 |   4651195363
          25 | Morry      | Wedge     | Wasco       | CA    | 4 Meadow Ridge Place         | 93515 |   2975012053
          26 | Rourke     | Money     | Bakersfield | CA    | 07493 Sugar Hill             | 93571 |   8308710890
          27 | Jerome     | Christley | Bakersfield | CA    | 5 Shopko Place               | 93164 |   5246167379
          28 | Alden      | Fyrth     | Arvin       | CA    | 9 Grayhawk Hill              | 94259 |   8087645551
          29 | Rayna      | Tooze     | Bakersfield | CA    | 14 Bluestem Lane             | 93309 |   4698086703
          30 | John       | Doe       | Lamont      | CA    | 7040 Fulton Pass             | 94947 |   1173026083
        9999 | Zondra     | Droghan   | Bakersfield | CA    | 1 Shasta Park                | 94710 |   2652504699
        9998 | Armstrong  | Tennant   | Taft        | CA    | 3612 Mosinee Center          | 93003 |   9494386407
        9997 | Modesta    | Mizzen    | Bakersfield | CA    | 94 Continental Terrace       | 94355 |   1252016744
```

Instead of Trigger Results



```
flowershop=# select * from view_manager_scheduling;
 employee_id |    employee_name     | pay_rate |    job_title     |     shift_start      |      shift_end
-------------+----------------------+----------+------------------+----------------------+----------------------
           2 | Armstrong Tennant    |    10.39 | Florist          | 2020-05-12 23:51:28 | 2020-05-13 07:51:28
           2 | Armstrong Tennant    |    10.39 | Florist          | 2020-05-27 13:57:49 | 2020-05-27 20:57:49
           5 | Caresse Warlawe      |    11.34 | Florist          | 2020-06-26 03:16:23 | 2020-06-26 06:16:23
           6 | Ailsun Humber        |    20.55 | Manager          | 2020-05-01 02:40:53 | 2020-05-01 04:40:53
           6 | Ailsun Humber        |    20.55 | Manager          | 2020-05-28 05:31:50 | 2020-05-28 13:31:50
          15 | Algernon Dougharty   |    13.90 | Florist          | 2020-05-11 02:04:12 | 2020-05-11 08:04:12
          18 | Candice Ligerton     |    13.61 | Cashier          | 2020-05-13 14:44:38 | 2020-05-13 17:44:38
          18 | Candice Ligerton     |    13.61 | Cashier          | 2020-05-30 00:21:07 | 2020-05-30 07:21:07
          18 | Candice Ligerton     |    13.61 | Cashier          | 2020-06-13 15:26:31 | 2020-06-13 20:26:31
          19 | Brandi Heaps         |    13.16 | Cashier          | 2020-05-04 05:52:36 | 2020-05-04 09:52:36
          19 | Brandi Heaps         |    13.16 | Cashier          | 2020-06-02 15:43:06 | 2020-06-02 23:43:06
          20 | Gladi Berns          |    12.09 | Florist          | 2020-06-14 15:43:17 | 2020-06-14 18:43:17
          21 | Konstanze Dellar     |    13.82 | Cashier          | 2020-05-02 15:48:14 | 2020-05-02 17:48:14
          21 | Konstanze Dellar     |    13.82 | Cashier          | 2020-05-25 23:59:54 | 2020-05-26 06:59:54
```



```
flowershop=# update view_manager_scheduling SET employee_name = 'Trigger Test' WHERE employee_id = 2;
UPDATE 2
flowershop=# select * from view_manager_scheduling;
 employee_id |    employee_name     | pay_rate |    job_title     |     shift_start      |      shift_end
-------------+----------------------+----------+------------------+----------------------+----------------------
           2 | Trigger Test         |    10.39 | Florist          | 2020-05-12 23:51:28 | 2020-05-13 07:51:28
           2 | Trigger Test         |    10.39 | Florist          | 2020-05-27 13:57:49 | 2020-05-27 20:57:49
           5 | Caresse Warlawe      |    11.34 | Florist          | 2020-06-26 03:16:23 | 2020-06-26 06:16:23
```

4.3 Stored Function, Procedures, and Triggers of Three DBMS (Microsoft SQL, MySQL, and Oracle)

As we discussed in the beginning of this phase, there exists numerous Database Management Systems. Microsoft SQL Server, MySQL, and Oracle are just three of many. Microsoft SQL Server is developed by Microsoft which uses Transact-Structured Query Language or T-SQL. MySQL uses Structured Query Language or SQL. Oracle uses Procedural Language-Structured Query Language or PL/SQL.

The first subsection will cover the major differences in T-SQL, SQL, and PL/SQL. The next subsection will go over the ways in which these three are similar. The last section will go over the generalized syntax of how to do procedures, triggers, and functions in all three DBMS.

4.3.1 Differences between languages T-SQL, SQL, PL/SQL

There are a number of key differences between languages T-SQL, SQL, and PL/SQL. SQL defines what needs to be done while PL/SQL defines how things need to be done and mainly used to create an application. SQL, on the other hand, is mainly used to manipulate data. Triggers in SQL do not allow two triggers with the same trigger timing, event or statement to be defined on a table. As for PL/SQL triggers, it allows multiple triggers with the same trigger timing and event to be defined on a table.

T-SQL provides more functionality than SQL. It has functions for mathematical operations. It provides much more control over how the application works. It also allows for multiple rows to be inserted into a table using the BULK INSERT statement. In regards to statements, PL/SQL uses INSERT INTO while T-SQL uses SELECT INTO statement. Important to also note that T-SQL is only supported in Microsoft SQL Server. Same applies to PL/SQL which is supported only in Oracle.

4.3.2 Similarities between T-SQL, SQL, and PL/SQL

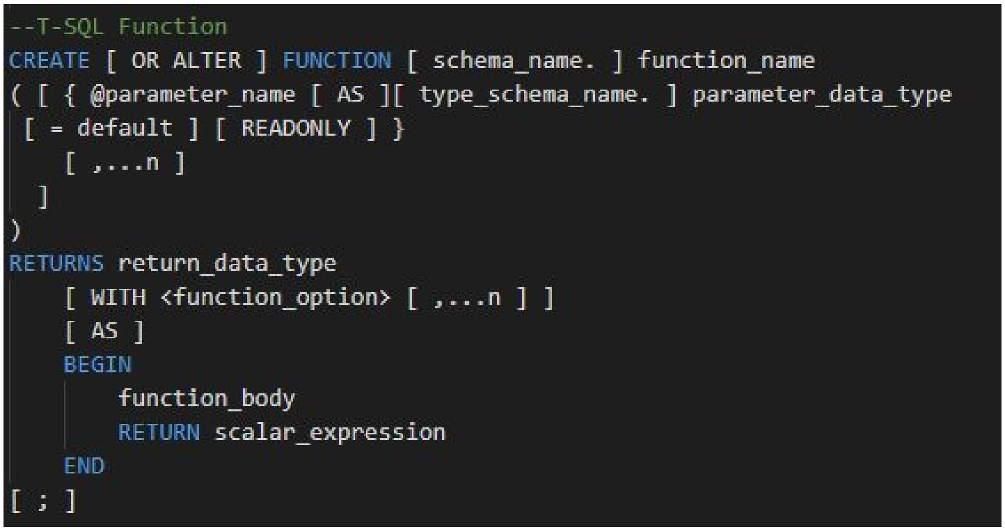
While T-SQL and PL/SQL are only supported by Microsoft SQL Server and Oracle respectively, there exists similarities among both languages and SQL. SQL is supported across all databases management systems like Oracle and MySQL. PL/SQL and T-SQL are both propietary extensions to SQL and allow grouping of SQL statements. Both languages also provide storage and execution of their code inside a database. They are convenient to create or write applications for their database vendors. In addition, SQL, T-SQL and PL-SQL are all capable of running on Windows and LInux.

4.3.3 Syntax of Stored Functions, Procedures, and Triggers of the Three DBMS

All three DBMS's have similarities and differences in their implementations and functionality that they provide. This extends to the syntax of the ways in which they define functions, stored procedures, and triggers. This section will provide the generalized syntax of the languages used in Microsoft SQL, MySQL, and Oracle.

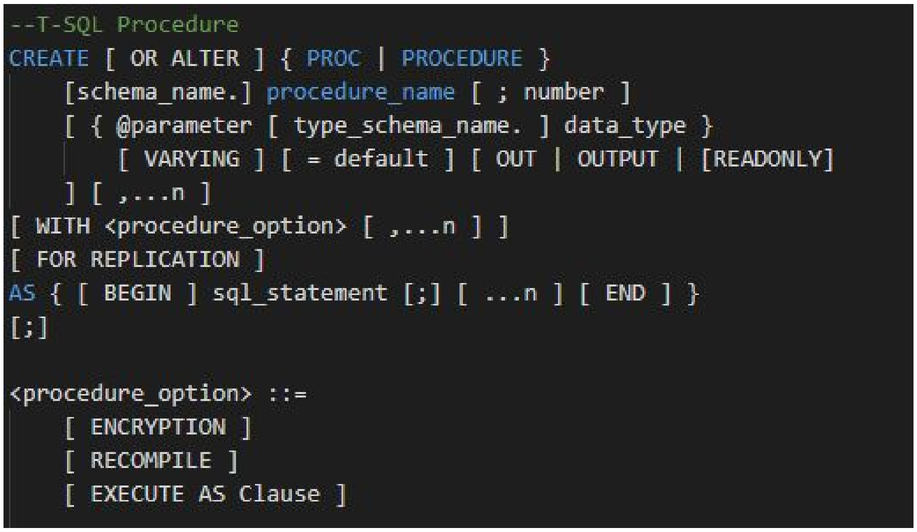
Microsoft SQL server

Stored Function



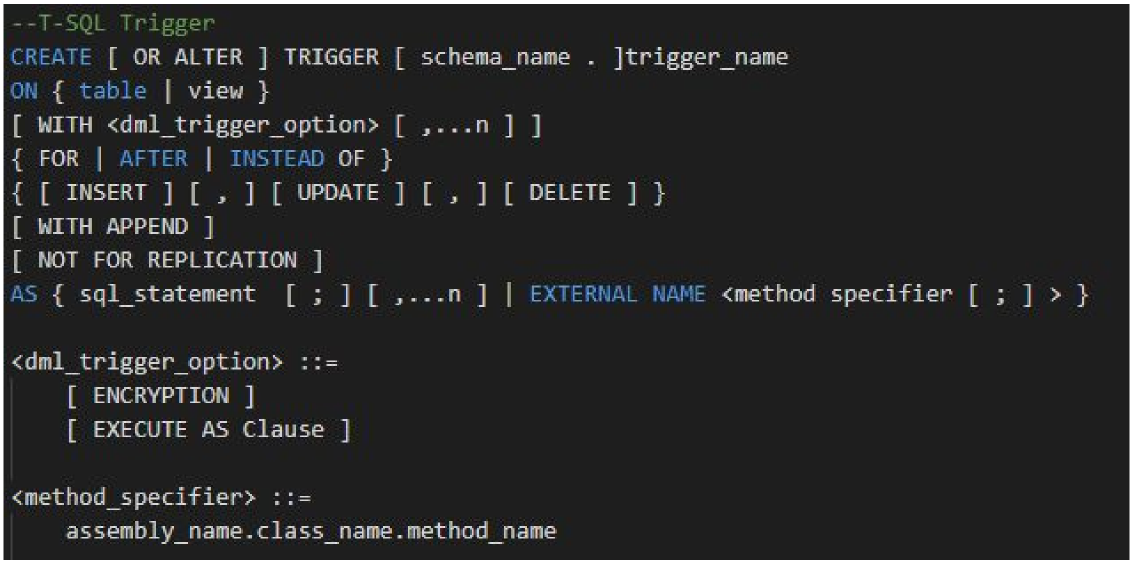
```
--T-SQL Function
CREATE [ OR ALTER ] FUNCTION [ schema_name. ] function_name
( [ { @parameter_name [ AS ][ type_schema_name. ] parameter_data_type
 [ = default ] [ READONLY ] }
    [ ,...n ]
  ]
)
RETURNS return_data_type
    [ WITH <function_option> [ ,...n ] ]
    [ AS ]
    BEGIN
        function_body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

Procedure



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

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

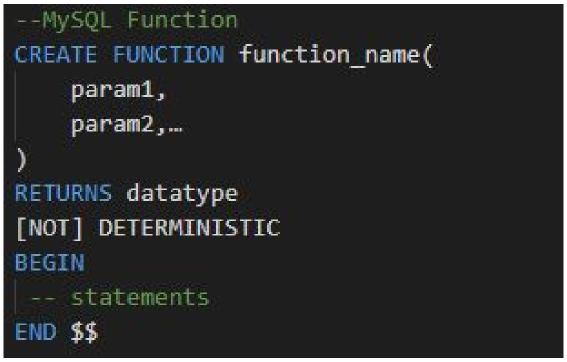
Trigger



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

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

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

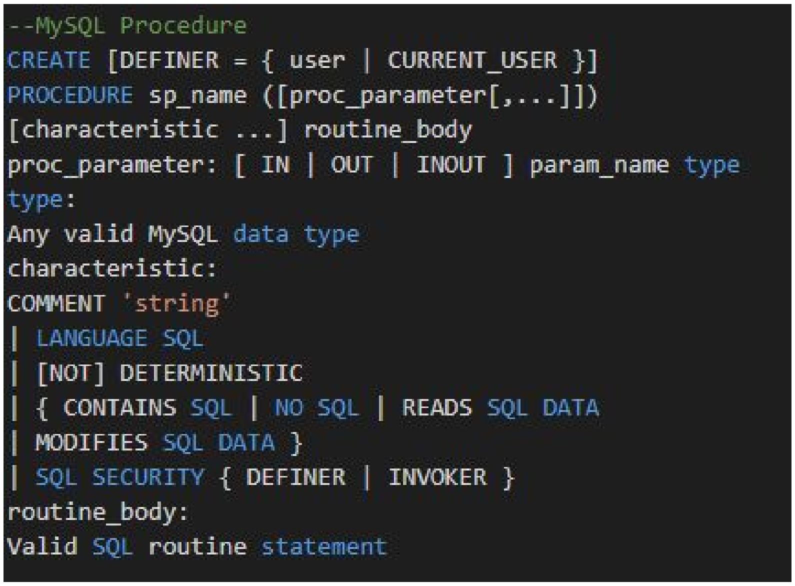
MySQL

Stored Function



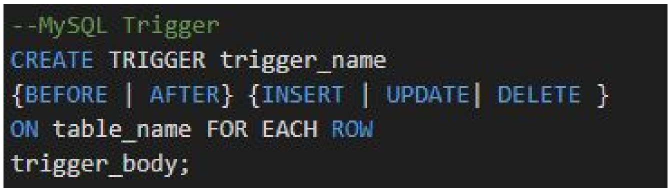
```
--MySQL Function
CREATE FUNCTION function_name(
    param1,
    param2,…
)
RETURNS datatype
[NOT] DETERMINISTIC
BEGIN
  -- statements
END $$
```

Procedure



```
--MySQL Procedure
CREATE [DEFINER = { user | CURRENT_USER }]
PROCEDURE sp_name ([proc_parameter[,...]])
[characteristic ...] routine_body
proc_parameter: [ IN | OUT | INOUT ] param_name type
type:
Any valid MySQL data type
characteristic:
COMMENT 'string'
| LANGUAGE SQL
| [NOT] DETERMINISTIC
| { CONTAINS SQL | NO SQL | READS SQL DATA
| MODIFIES SQL DATA }
| SQL SECURITY { DEFINER | INVOKER }
routine_body:
Valid SQL routine statement
```

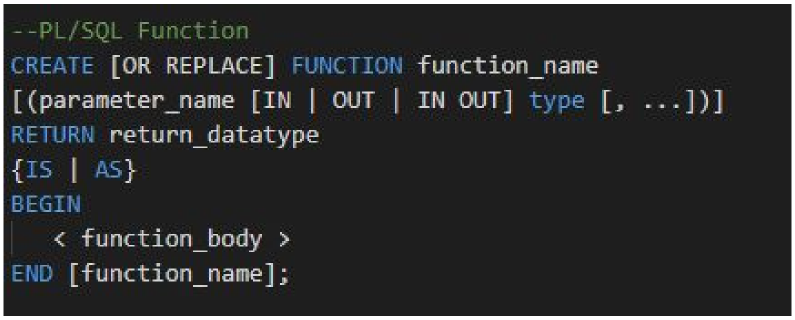
Trigger



```
--MySQL Trigger
CREATE TRIGGER trigger_name
{BEFORE | AFTER} {INSERT | UPDATE| DELETE }
ON table_name FOR EACH ROW
trigger_body;
```

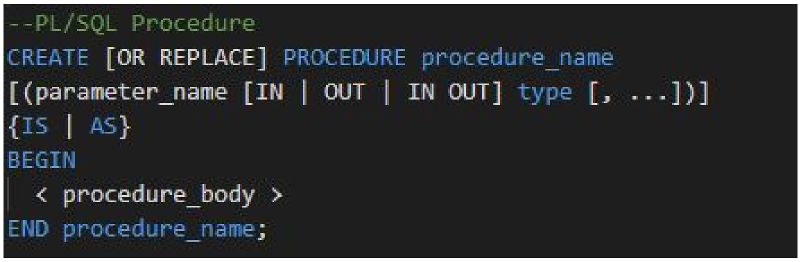
Oracle

Stored Function



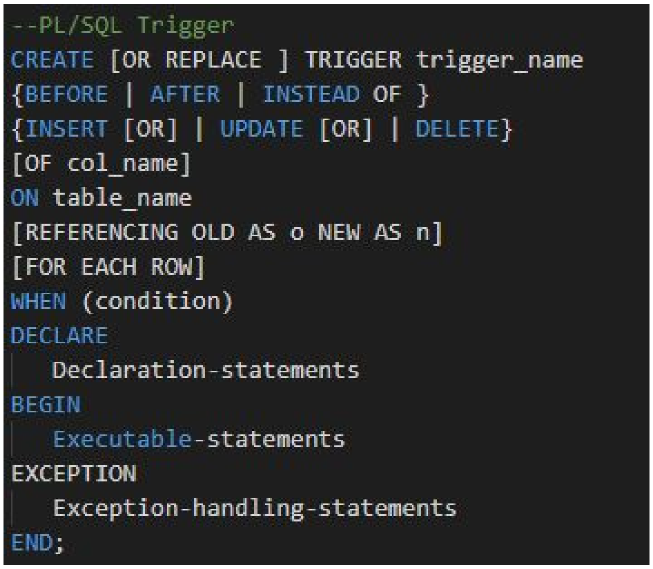
```
--PL/SQL Function
CREATE [OR REPLACE] FUNCTION function_name
[(parameter_name [IN | OUT | IN OUT] type [, ...])]
RETURN return_datatype
{IS | AS}
BEGIN
   < function_body >
END [function_name];
```

Procedure



```
--PL/SQL Procedure
CREATE [OR REPLACE] PROCEDURE procedure_name
[(parameter_name [IN | OUT | IN OUT] type [, ...])]
{IS | AS}
BEGIN
  < procedure_body >
END procedure_name;
```

Triggers



```
--PL/SQL Trigger
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)
DECLARE
   Declaration-statements
BEGIN
   Executable-statements
EXCEPTION
   Exception-handling-statements
END;
```

Phase 5: Graphic User Interface Design and Implementation

The final phase of this document will go over the GUI for a manager of Bakersfield Flower Shops and the way it interacts with the database. The first section will discuss some of the functionality and the usergroup I designed the database for, the second section will go the code I used to create the database, and the final section will go over how I feel I performed and how much I learned in this class.

5.1 Functionalities and User group of the GUI application

To make this front end website application I used HTML, CSS with the bootstrap library, Javascript with the JQuery design library, Postgres, and Node.js with the Express.js Framework. HTML, CSS, and JQuery were used to design the front end application and implement various ways that the front end could interact with the backend. Node.js is a server side web development framework that allows queries to databases to be executed using javascript, massively simplifying the developments of applications that use a backend because a developer can use javascript for both front end and back end code. Express.js is a framework for Node.js that makes using node even simpler and allows for rapid development of full-stack web apps.

The user group for this application is for managers of a local flower shop. The part of a manager's job I designed to interact with our database is a dashboard for managers that will allow them to schedule staff to calendar days, and to print out reports for the current revenue and expenditures of Bakersfield Flowershop.

5.1.1 Itemized descriptions of GUI application, and reports generated

I will provide brief descriptions for various sections featured in my applications. Because I developed the application all on one page I'll give a broad overview here, but in the following section alongside screenshots I'll include more detail for different parts of the database alongside the screenshot.

Dashboard overview

Dashboard where a manager can control the schedule of the employees for Bakersfield Flowershop. Features a clickable list of employees, buttons increasing the starting time and ending time, an insertion button to input that employee and time into the database, a row of buttons for the calendar week along the bottom. Also has buttons that allow for auto scheduling for a day and activating two modals to generate the reports I've chosen.

Scheduling modal

Activated by a button on the dashboard. Features a menu where you can select what job types you want featured on an employee schedule report.

Scheduling Report

The schedule for the current work week. Each day has 4 sections by default showing Managers, Cashier, Florists, and Delivery Drivers and their hours they work for the day.

Revenue Model

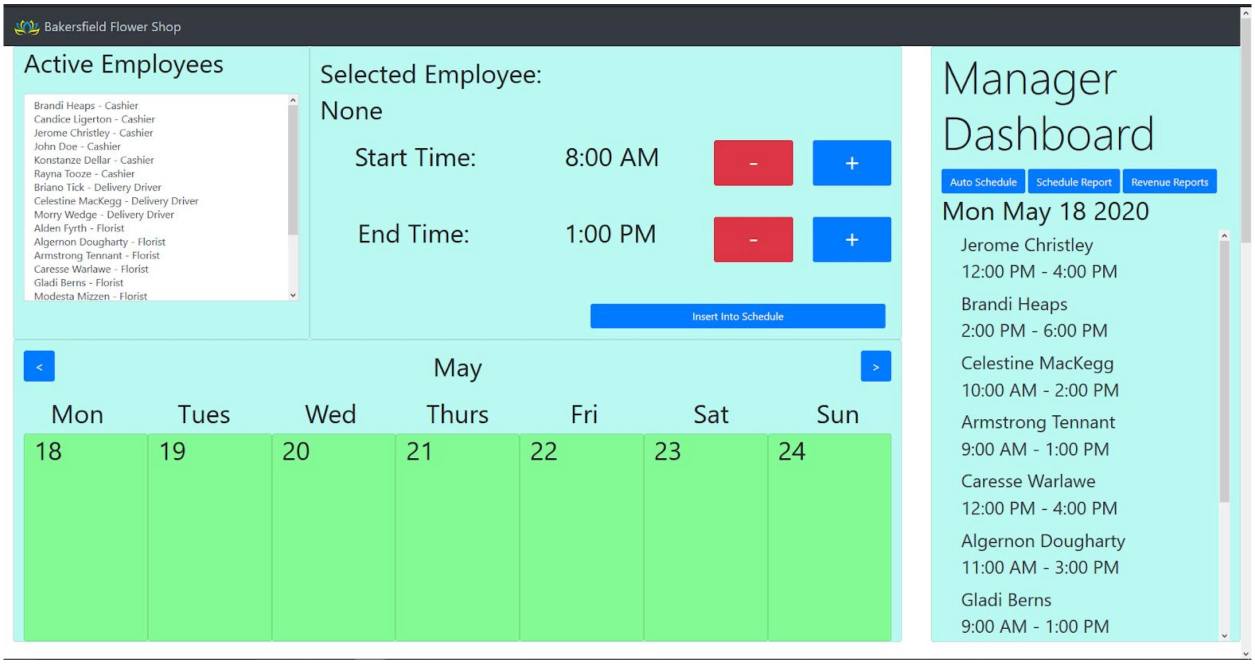
Activated by a button on the Manager dashboard. Features a button to randomly generate ingoing and outgoing payments.

Revenue Report

Revenue report that shows twelve customers, total amount paid to the store, and the date they last paid. Also shows the totals paid out to Suppliers of Bakersfield Flowershop. The shows total revenue and total expenditure, then another section showing total profit.

5.1.2 Screenshots of the application

Dashboard overview



Active Employees

Brandi Heaps - Cashier
Candice Ligerton - Cashier
Jerome Christley - Cashier
John Doe - Cashier
Konstanze Dellar - Cashier
Rayna Tooze - Cashier
Briano Tick - Delivery Driver
Celestine MacKegg - Delivery Driver
Morry Wedge - Delivery Driver
Alden Fyrth - Florist
Algernon Dougharty - Florist
Armstrong Tennant - Florist
Caresse Warlawe - Florist
Gladi Berns - Florist
Modesta Mizzen - Florist

Selected Employee:
None

Start Time: 8:00 AM

End Time: 1:00 PM

May

Mon	Tues	Wed	Thurs	Fri	Sat	Sun
18	19	20	21	22	23	24

Manager Dashboard

Mon May 18 2020

Jerome Christley
12:00 PM - 4:00 PM

Brandi Heaps
2:00 PM - 6:00 PM

Celestine MacKegg
10:00 AM - 2:00 PM

Armstrong Tennant
9:00 AM - 1:00 PM

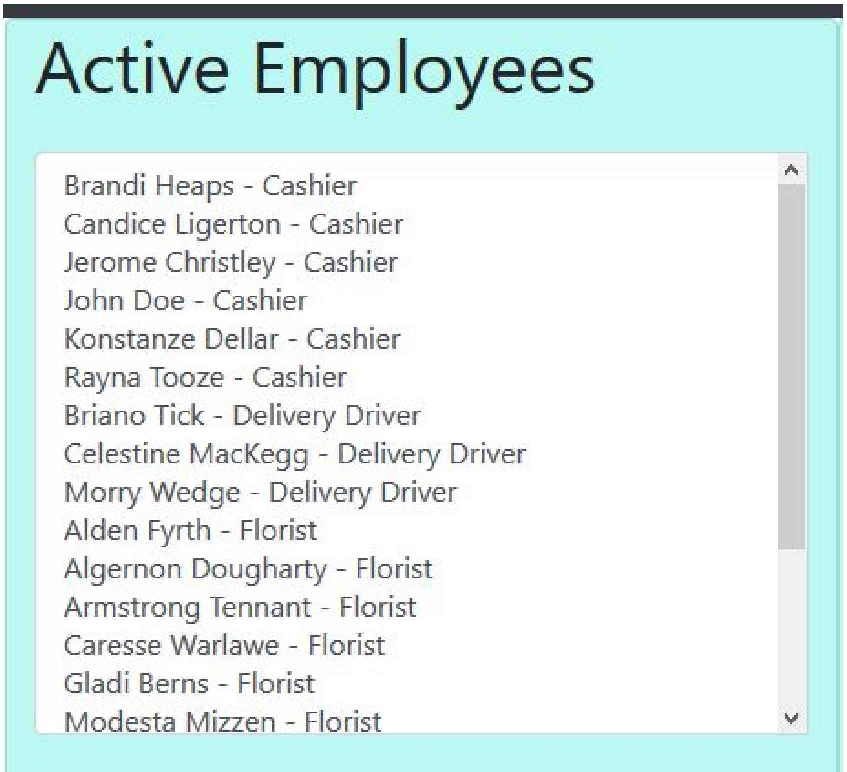
Caresse Warlawe
12:00 PM - 4:00 PM

Algernon Dougharty
11:00 AM - 3:00 PM

Gladi Berns
9:00 AM - 1:00 PM

Broad look at the interface, shows all sections of interface talked about in previous section.

Active Employees

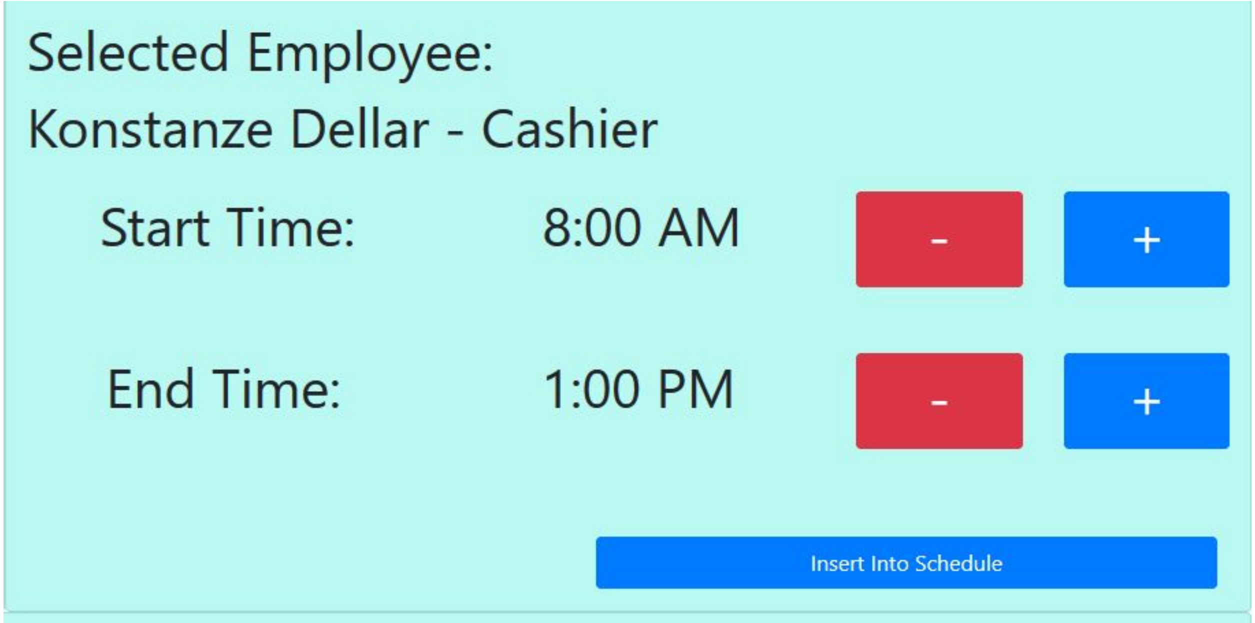


Active Employees

Brandi Heaps - Cashier
Candice Ligerton - Cashier
Jerome Christley - Cashier
John Doe - Cashier
Konstanze Dellar - Cashier
Rayna Tooze - Cashier
Briano Tick - Delivery Driver
Celestine MacKegg - Delivery Driver
Morry Wedge - Delivery Driver
Alden Fyrth - Florist
Algernon Dougharty - Florist
Armstrong Tennant - Florist
Caresse Warlawe - Florist
Gladi Berns - Florist
Modesta Mizzen - Florist

This list is filled by a query that returns a list of the employees and their names and job titles. By only showing employees without an end date in the database it ensures only active employees can be assigned to work a shift.

Set hours for selected employee



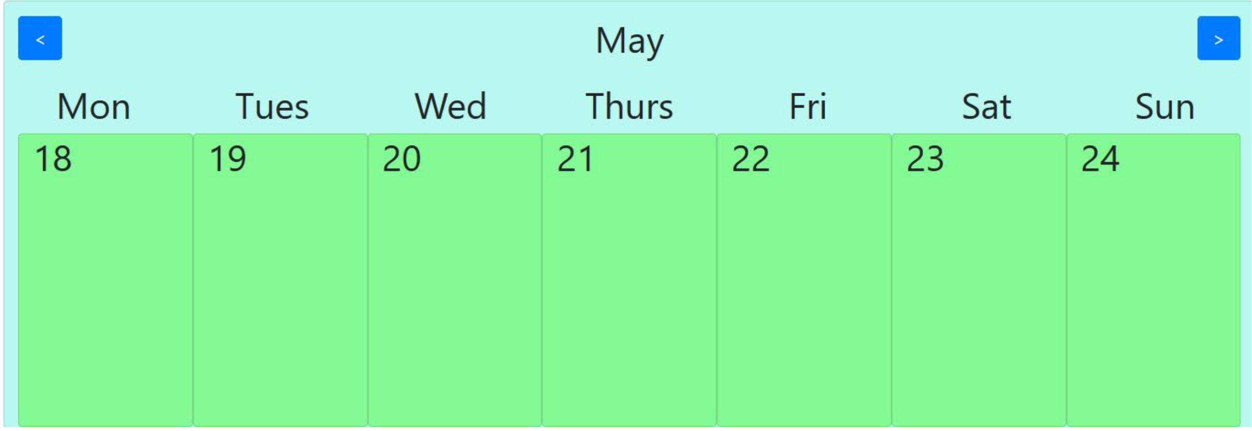
Selected Employee:
Konstanze Dellar - Cashier

Start Time: 8:00 AM

End Time: 1:00 PM

The employee selected by the list on the left list can then have their hours and then inserted into the database.

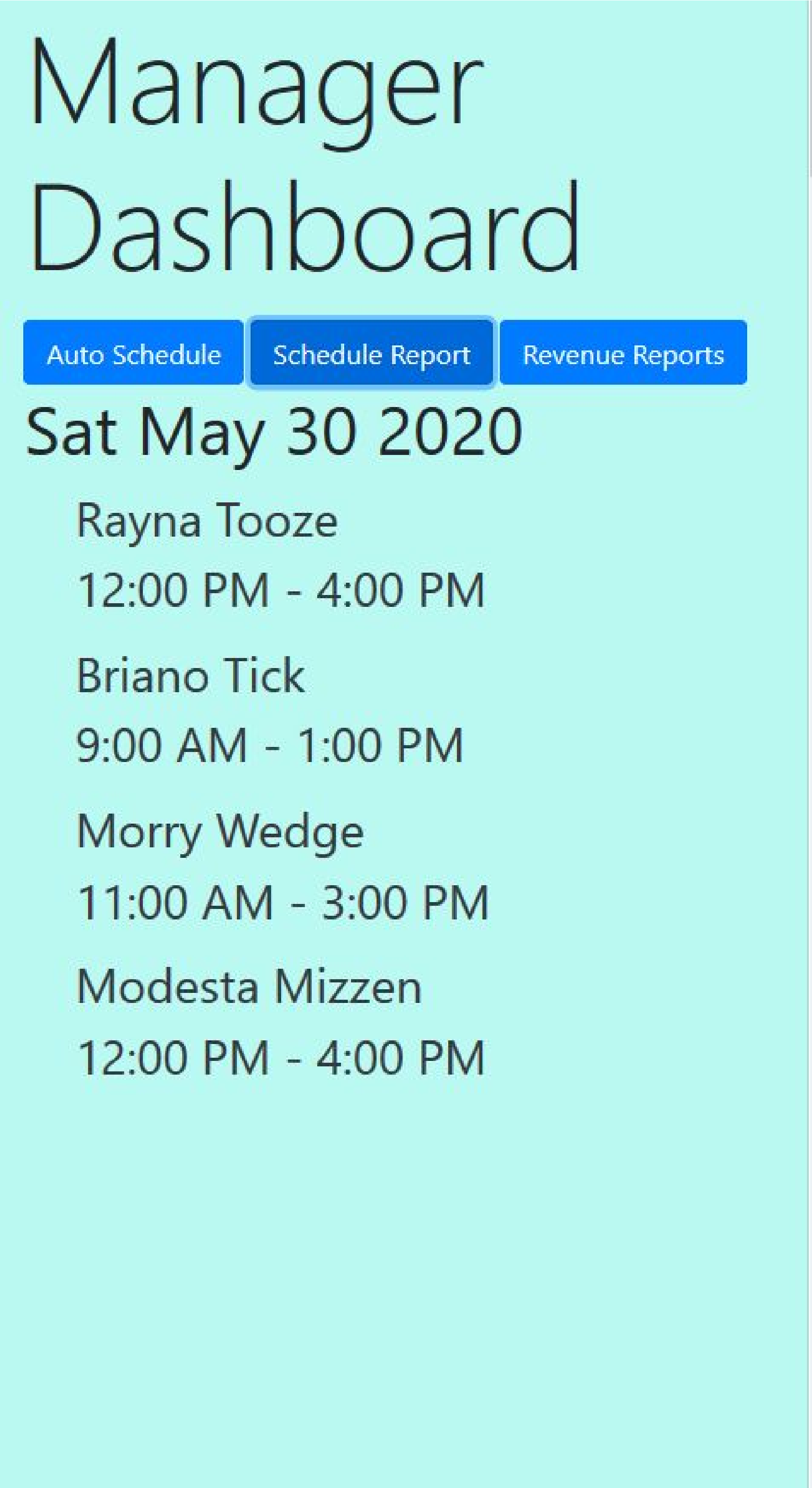
Select Day to edit schedule



Mon	Tues	Wed	Thurs	Fri	Sat	Sun
18	19	20	21	22	23	24

The date boxes here are able to be clicked and they will change the date of current focus for the list on the right. When box is chosen it executes a query to get the employees for the chosen day which then fills up a list on the right hand side with employees names and hours set for the day.

Daily schedule and links to modals



Manager Dashboard

Sat May 30 2020

Rayna Tooze

12:00 PM - 4:00 PM

Briano Tick

9:00 AM - 1:00 PM

Morry Wedge

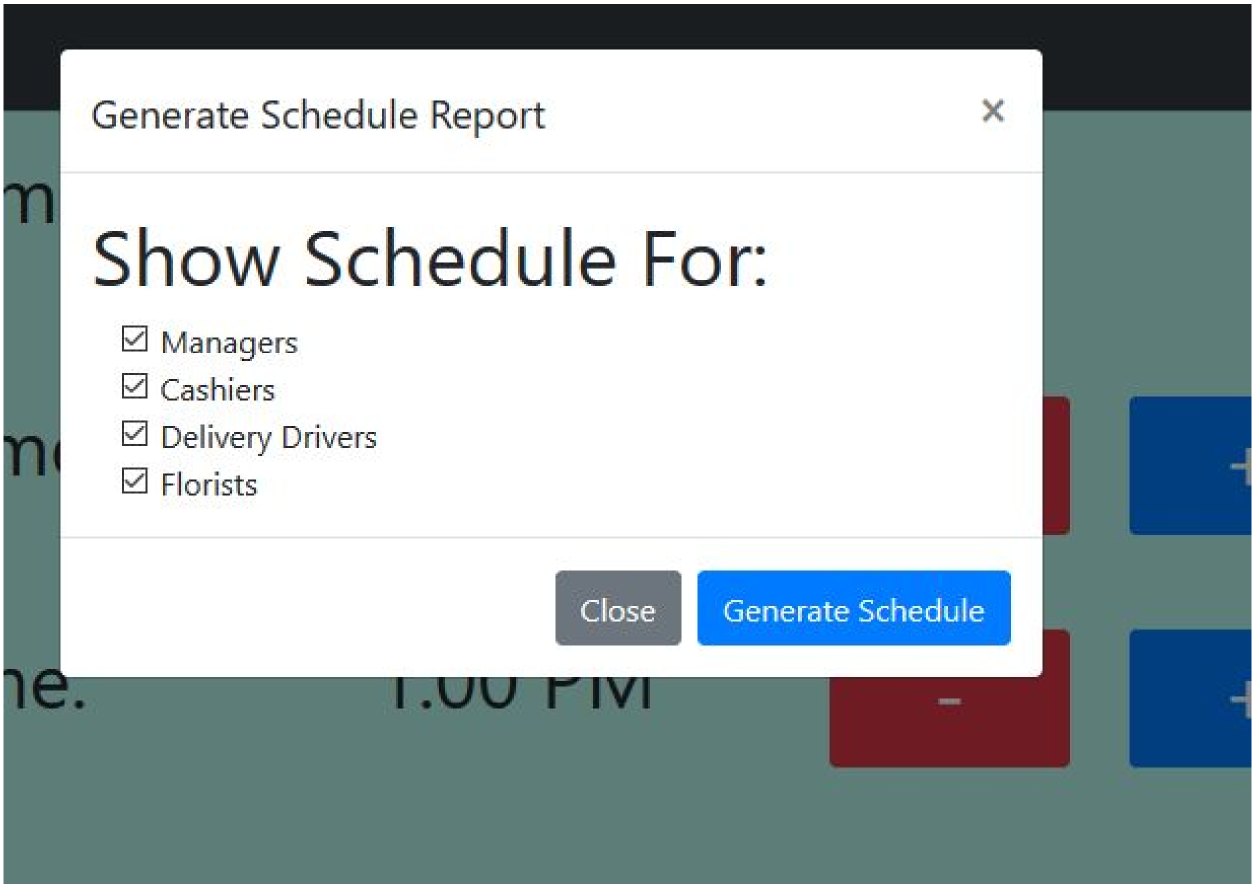
11:00 AM - 3:00 PM

Modesta Mizzen

12:00 PM - 4:00 PM

List of employees working that day is populated by the query discussed from the previous screen shot. The date shown changes and the employees for the day update when selected. There are also 3 buttons here that allow you to randomly generate employees in the schedule for the week, and the other two bring up modals that let you generate reports from data in the database.

Schedule Report Modal



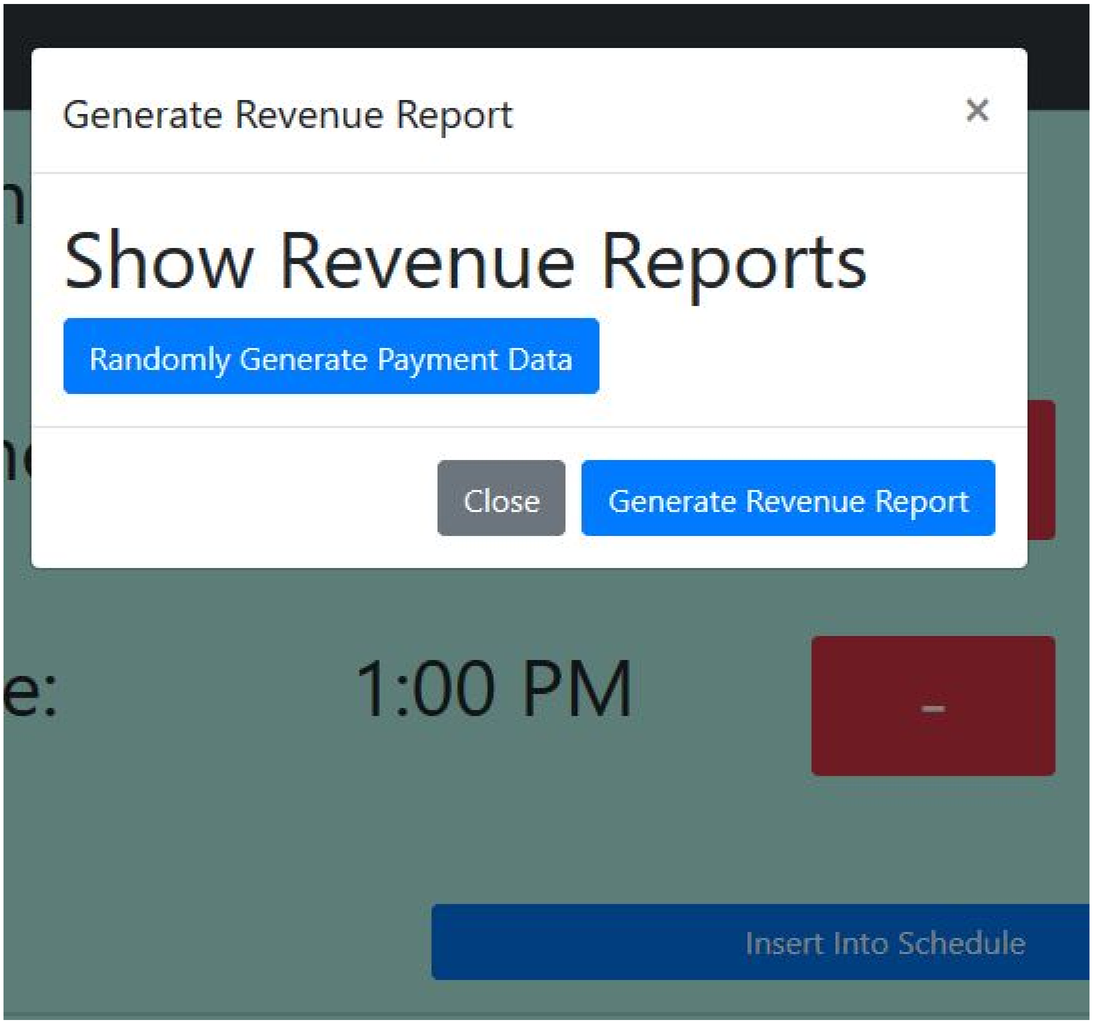
Generate Schedule Report

Show Schedule For:

☑ Managers
☑ Cashiers
☑ Delivery Drivers
☑ Florists

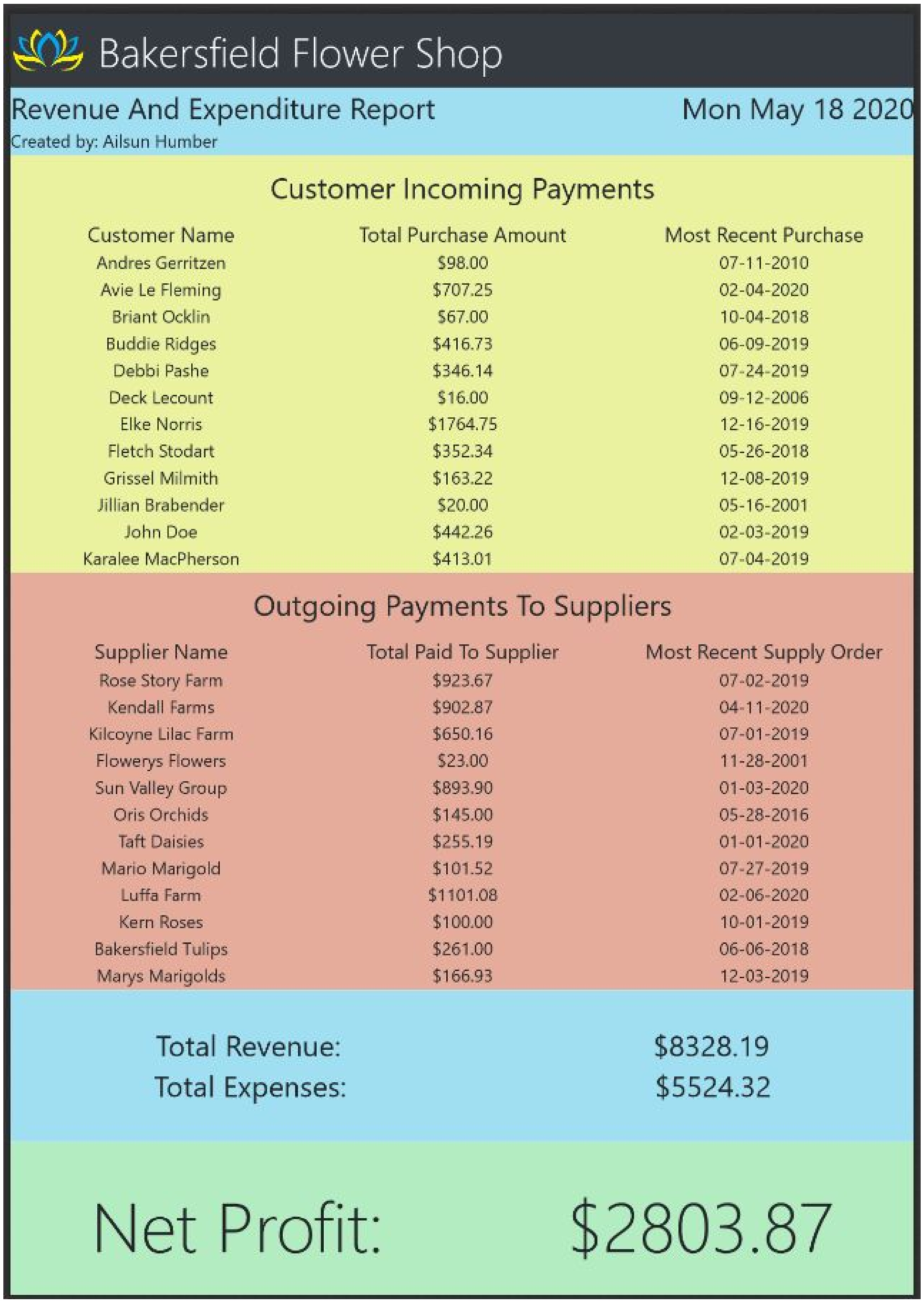
Modal allows you to choose which job types you want to show in the schedule then produces that report showing who is working what days by day, job type, and hours.

Revenue Report Modal



This modal allows the user to generate a revenue report from data in the database, and allows the user to randomly add payments into the database.

Revenue Report



Bakersfield Flower Shop

Revenue And Expenditure Report

Mon May 18 2020

Created by: Ailsun Humber

Customer Incoming Payments

Customer Name	Total Purchase Amount	Most Recent Purchase
Andres Gerritzen	$98.00	07-11-2010
Avie Le Fleming	$707.25	02-04-2020
Briant Ocklin	$67.00	10-04-2018
Buddie Ridges	$416.73	06-09-2019
Debbi Pashe	$346.14	07-24-2019
Deck Lecount	$16.00	09-12-2006
Elke Norris	$1764.75	12-16-2019
Fletch Stodart	$352.34	05-26-2018
Grissel Milmith	$163.22	12-08-2019
Jillian Brabender	$20.00	05-16-2001
John Doe	$442.26	02-03-2019
Karalee MacPherson	$413.01	07-04-2019

Outgoing Payments To Suppliers

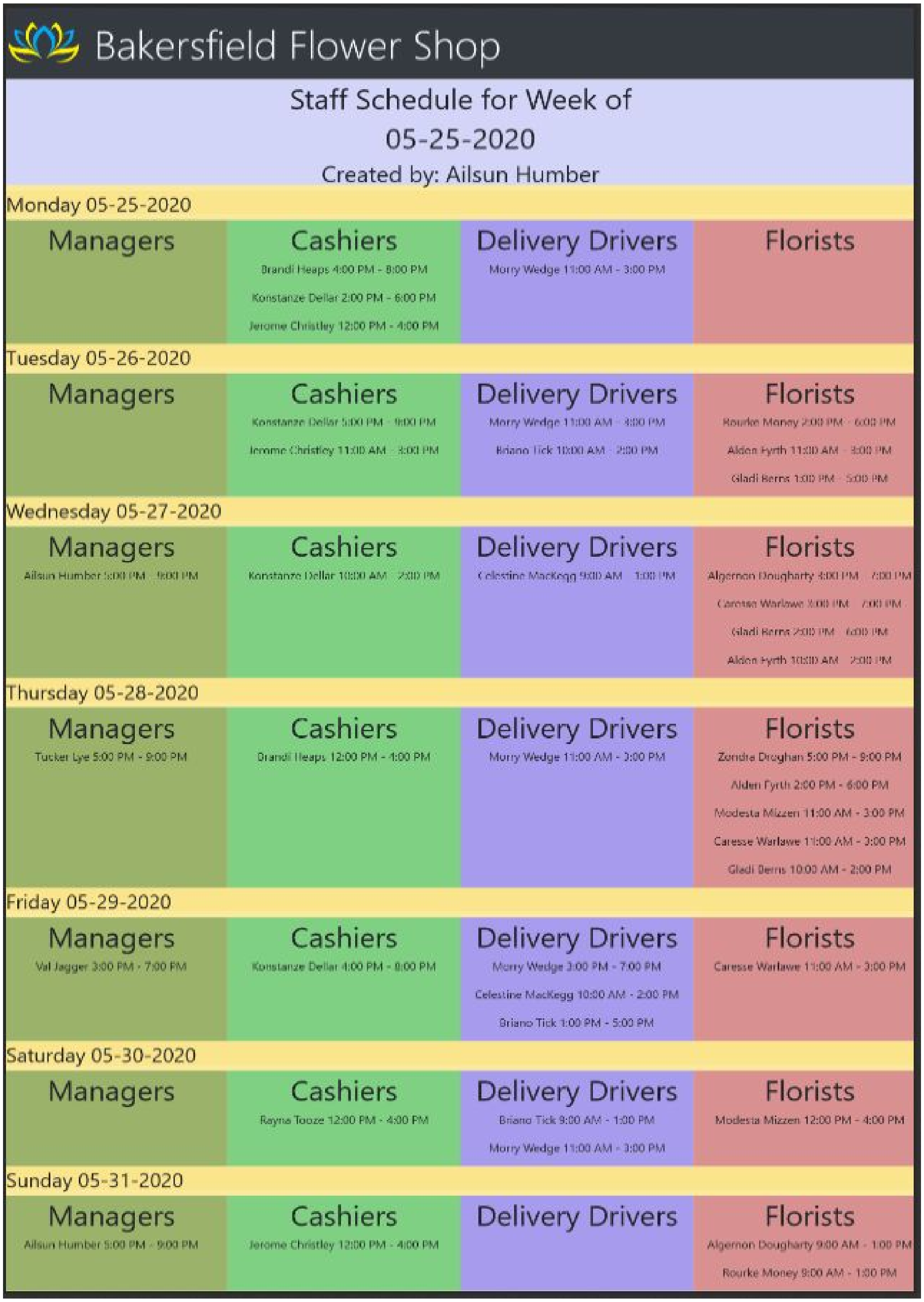
Supplier Name	Total Paid To Supplier	Most Recent Supply Order
Rose Story Farm	$923.67	07-02-2019
Kendall Farms	$902.87	04-11-2020
Kilcoyne Lilac Farm	$650.16	07-01-2019
Flowerys Flowers	$23.00	11-28-2001
Sun Valley Group	$893.90	01-03-2020
Oris Orchids	$145.00	05-28-2016
Taft Daisies	$255.19	01-01-2020
Mario Marigold	$101.52	07-27-2019
Luffa Farm	$1101.08	02-06-2020
Kern Roses	$100.00	10-01-2019
Bakersfield Tulips	$261.00	06-06-2018
Marys Marigolds	$166.93	12-03-2019

Total Revenue:	$8328.19
Total Expenses:	$5524.32

Net Profit: $2803.87

Revenue Report showing some of the customers and their most recent payment, some of the payments out to suppliers, and the net profit for the store.

Schedule Report



Bakersfield Flower Shop

Staff Schedule for Week of
05-25-2020
Created by: Ailsun Humber

Monday 05-25-2020

Managers	Cashiers	Delivery Drivers	Florists
	Brandi Heaps 4:00 PM - 8:00 PM Konstanze Dellar 2:00 PM - 6:00 PM Jerome Christley 12:00 PM - 4:00 PM	Morry Wedge 11:00 AM - 3:00 PM	

Tuesday 05-26-2020

Managers	Cashiers	Delivery Drivers	Florists
	Konstanze Dellar 5:00 PM - 9:00 PM Jerome Christley 11:00 AM - 3:00 PM	Morry Wedge 11:00 AM - 3:00 PM Briano Tick 10:00 AM - 2:00 PM	Rourke Money 2:00 PM - 6:00 PM Alden Fyrth 11:00 AM - 3:00 PM Gladi Berns 1:00 PM - 5:00 PM

Wednesday 05-27-2020

Managers	Cashiers	Delivery Drivers	Florists
Ailsun Humber 5:00 PM - 9:00 PM	Konstanze Dellar 10:00 AM - 2:00 PM	Celestine MacKegg 9:00 AM - 1:00 PM	Algernon Dougharty 3:00 PM - 7:00 PM Caresse Warlawe 3:00 PM - 7:00 PM Gladi Berns 2:00 PM - 6:00 PM Alden Fyrth 10:00 AM - 2:00 PM

Thursday 05-28-2020

Managers	Cashiers	Delivery Drivers	Florists
Tucker Lye 5:00 PM - 9:00 PM	Brandi Heaps 12:00 PM - 4:00 PM	Morry Wedge 11:00 AM - 3:00 PM	Zondra Droghan 5:00 PM - 9:00 PM Alden Fyrth 2:00 PM - 6:00 PM Modesta Mizzen 11:00 AM - 3:00 PM Caresse Warlawe 11:00 AM - 3:00 PM Gladi Berns 10:00 AM - 2:00 PM

Friday 05-29-2020

Managers	Cashiers	Delivery Drivers	Florists
Val Jagger 3:00 PM - 7:00 PM	Konstanze Dellar 4:00 PM - 8:00 PM	Morry Wedge 3:00 PM - 7:00 PM Celestine MacKegg 10:00 AM - 2:00 PM Briano Tick 1:00 PM - 5:00 PM	Caresse Warlawe 11:00 AM - 3:00 PM

Saturday 05-30-2020

Managers	Cashiers	Delivery Drivers	Florists
	Rayna Tooze 12:00 PM - 4:00 PM	Briano Tick 9:00 AM - 1:00 PM Morry Wedge 11:00 AM - 3:00 PM	Modesta Mizzen 12:00 PM - 4:00 PM

Sunday 05-31-2020

Managers	Cashiers	Delivery Drivers	Florists
Ailsun Humber 5:00 PM - 9:00 PM	Jerome Christley 12:00 PM - 4:00 PM		Algernon Dougharty 9:00 AM - 1:00 PM Rourke Money 9:00 AM - 1:00 PM

Shows the days and job types of each of the scheduled employees of the store.

5.1.3 Tables, Views, Stored Subprograms, and Triggers Used

All the code for the tables, views, functions, and tables used in the database. In 5.2.1 I'll go over the purpose of the views and procedures I used. This section will only contain the code.

```
CREATE TABLE IF NOT EXISTS customer (
    customer_id SERIAL PRIMARY KEY not null,
    fname VARCHAR(50) not null,
    lname VARCHAR(50) not null,
    city VARCHAR(50) not null,
    state char(2) not null,
    street VARCHAR(50) not null,
    zip integer not null,
    username VARCHAR(50),
    password VARCHAR(50),
    email VARCHAR(50),
    acc_creation_date timestamp,
    phone_number bigint not null

 );
```

```
 CREATE TABLE IF NOT EXISTS employee (
    employee_id SERIAL PRIMARY KEY not null,
    fname VARCHAR(50) not null,
    lname VARCHAR(50) not null,
    city VARCHAR(11) not null,
    state VARCHAR(50) not null,
    street VARCHAR(50) not null,
    zip INT not null,
    phone_number bigint not null
);
```

```
CREATE TABLE IF NOT EXISTS delivery_address (
    address_id SERIAL PRIMARY KEY not null,
    city VARCHAR(50) not null,
    street VARCHAR(50) not null,
    state VARCHAR(50) not null,
    zip INT not null
 );
```

```
CREATE TABLE IF NOT EXISTS flower_product (
    product_id SERIAL PRIMARY KEY not null,
    product_name VARCHAR(50) not null,
    purchase_price decimal(12,2) not null,
    sell_price decimal(12,2) not null,
    color VARCHAR(50) not null,
    length DECIMAL(4,2) not null,
    product_image VARCHAR(24) not null,
    description VARCHAR(255) not null
);
```

```
CREATE TABLE IF NOT EXISTS order_status (
    status_id INT PRIMARY KEY not null,
    status VARCHAR(50) not null
);
```

```
CREATE TABLE IF NOT EXISTS payment_type (
    payment_type_id INT PRIMARY KEY not null,
    description VARCHAR(50) not null
);
```

```
CREATE TABLE IF NOT EXISTS supplier (
    supply_id SERIAL PRIMARY KEY not null,
    vendor_name VARCHAR(50) not null,
    city VARCHAR(50) not null,
    state VARCHAR(50) not null,
    street VARCHAR(50) not null,
    zip INT not null,
    phone_number bigint not null
);
```

```
CREATE TABLE IF NOT EXISTS product_order (
    p_order_number SERIAL PRIMARY KEY not null,
    order_time timestamp not null,

    customer_id integer not null,
    status_id integer not null,
    employee_id integer not null,
    address_id integer not null,

    CONSTRAINT fk_order_customer FOREIGN KEY (customer_id)
        REFERENCES customer(customer_id),
    CONSTRAINT fk_order_status FOREIGN KEY (status_id)
        REFERENCES order_status(status_id),
    CONSTRAINT fk_order_employee FOREIGN KEY (employee_id)
        REFERENCES employee(employee_id),
    CONSTRAINT fk_order_address FOREIGN KEY (address_id)
        REFERENCES delivery_address(address_id)
);
```

```
CREATE TABLE IF NOT EXISTS incoming_payment (
    incoming_payment_id SERIAL PRIMARY KEY not null,
    sales_tax DECIMAL(10,4) not null

);
```

```
CREATE TABLE IF NOT EXISTS outgoing_payment (
    outgoing_payment_id SERIAL PRIMARY KEY not null,
    supplier_invoice_id INT


);
```

```
CREATE TABLE IF NOT EXISTS payment(
    payment_time timestamp not null,
    amount decimal(12,2) not null default 0,
    payment_type_id integer not null,
    incoming_payment_id integer references
incoming_payment(incoming_payment_id) UNIQUE,
```

```sql
    outgoing_payment_id integer references
outgoing_payment(outgoing_payment_id) UNIQUE,

    CONSTRAINT ck_pay_amount CHECK (amount > 0)
);
```

```sql
CREATE TABLE IF NOT EXISTS package (
    package_id SERIAL PRIMARY KEY not null,
    expected_time timestamp not null,
    message VARCHAR(19) not null,
    p_order_number INT not null,
    employee_id INT not null,

    CONSTRAINT fk_package_order_number FOREIGN KEY (p_order_number)
        REFERENCES product_order(p_order_number),
    CONSTRAINT fk_package_employee_id FOREIGN KEY (employee_id)
        REFERENCES employee(employee_id)
);
```

```sql
CREATE TABLE IF NOT EXISTS recipient (
    recipient_id SERIAL PRIMARY KEY not null,
    fname VARCHAR(50) not null,
    lname VARCHAR(50) not null,
    phone_number bigint not null,
    package_id integer not null,

    CONSTRAINT fk_recipient_package FOREIGN KEY (package_id)
        REFERENCES package(package_id)
);
```

```sql
CREATE TABLE IF NOT EXISTS requires (
    p_order_number integer not null,
    incoming_payment_id integer not null UNIQUE,
    CONSTRAINT pk_supply_product_order
        PRIMARY KEY (p_order_number, incoming_payment_id),

    CONSTRAINT fk_requires_order_number FOREIGN KEY (p_order_number)
        REFERENCES product_order(p_order_number),
    CONSTRAINT fk_requires_payment FOREIGN KEY (incoming_payment_id)
        REFERENCES incoming_payment(incoming_payment_id)

);
```

```

CREATE TABLE IF NOT EXISTS work_history (
    history_id serial PRIMARY KEY not null,
    start_date timestamp not null,
    end_date timestamp,
    job_title VARCHAR(50) not null,
    pay_rate decimal(12,2) not null,
    employee_id INT not null,

    CONSTRAINT fk_employee_history FOREIGN KEY (employee_id)
        REFERENCES employee(employee_id)
);
```

```
CREATE TABLE IF NOT EXISTS contains (
    p_order_number integer not null,
    product_id integer not null,
    quantity_item integer not null,
    point_of_sale_price decimal(12,2),

    CONSTRAINT pk_contains
        PRIMARY KEY (p_order_number, product_id),

    CONSTRAINT fk_contains_order_number FOREIGN KEY (p_order_number)
        REFERENCES product_order(p_order_number),

    CONSTRAINT fk_contains_product FOREIGN KEY (product_id)
        REFERENCES flower_product(product_id)
);
```

```

CREATE TABLE IF NOT EXISTS supply_purchase_order (
    supply_purchase_id SERIAL PRIMARY KEY not null,
    supply_purchase_time timestamp not null,
    employee_id int not null,
    supply_id int not null,

    CONSTRAINT fk_purchase_order_employee FOREIGN KEY (employee_id)
        REFERENCES employee(employee_id),
    CONSTRAINT fk_purchase_order_supplier FOREIGN KEY (supply_id)
        REFERENCES supplier(supply_id)
);
```

```
 CREATE TABLE IF NOT EXISTS needs (
    supply_purchase_id INT not null,
    outgoing_payment_id INT not null UNIQUE,

    CONSTRAINT pk_supply_needs_payment
        PRIMARY KEY (supply_purchase_id, outgoing_payment_id),

    CONSTRAINT fk_needs_supply_purchase FOREIGN KEY (supply_purchase_id)
        REFERENCES supply_purchase_order(supply_purchase_id),
    CONSTRAINT fk_needs_payment FOREIGN KEY (outgoing_payment_id)
        REFERENCES outgoing_payment(outgoing_payment_id)
 );
```

```
 CREATE TABLE IF NOT EXISTS refills (
    supply_purchase_id integer not null,
    product_id integer not null,
    quantity_item integer not null,
    supply_price decimal(12,2) not null,

    CONSTRAINT pk_refills_supply_purchase_product
        PRIMARY KEY (supply_purchase_id, product_id),

    CONSTRAINT fk_supply_purchase FOREIGN KEY (supply_purchase_id)
        REFERENCES supply_purchase_order(supply_purchase_id),
    CONSTRAINT fk_product FOREIGN KEY (product_id)
        REFERENCES flower_product(product_id)
);
```

```
CREATE TABLE IF NOT EXISTS work_shift (
    shift_id serial not null,
    employee_id integer not null,
    shift_date date not null,
    begin_time time not null,
    end_time time not null,

    CONSTRAINT pk_employee_shift
        PRIMARY KEY (shift_id, employee_id),

    CONSTRAINT fk_employee_id FOREIGN KEY (employee_id)
        REFERENCES employee(employee_id),
        -- make sure employee doesn't work same day twice business rule
    CONSTRAINT id_day_check UNIQUE(employee_id, shift_date)
);
```

```
CREATE VIEW view_manager_scheduling AS
SELECT employee.employee_id, employee.fname || ' ' || employee.lname AS
employee_name,
work_history.pay_rate, work_history.job_title, work_shift.shift_date as
day,
work_shift.begin_time as shift_start, work_shift.end_time as shift_end
FROM employee
INNER JOIN work_history ON work_history.employee_id = employee.employee_id
INNER JOIN work_shift ON work_shift.employee_id = employee.employee_id
WHERE work_history.end_date IS NULL
order by employee_id, shift_start
;

CREATE VIEW view_number_employees_working AS
```

```
SELECT work_history.job_title, COUNT(work_history.job_title)
count_of_job_type, work_shift.shift_date as day
FROM employee
INNER JOIN work_history ON work_history.employee_id = employee.employee_id
INNER JOIN work_shift ON work_shift.employee_id = employee.employee_id
GROUP BY work_history.job_title, work_shift.shift_date
ORDER BY work_shift.shift_date
;
```

```
CREATE VIEW view_positive_revenue AS
SELECT customer.fname || ' ' || customer.lname as customer_name,
to_char(MAX(payment.payment_time::date), 'MM-DD-YYYY') last_bought,
SUM(payment.amount) revenue
FROM incoming_payment
INNER JOIN payment ON incoming_payment.incoming_payment_id =
payment.incoming_payment_id
INNER JOIN requires ON requires.incoming_payment_id =
incoming_payment.incoming_payment_id
INNER JOIN product_order ON requires.p_order_number =
product_order.p_order_number
LEFT JOIN customer ON product_order.customer_id = customer.customer_id
GROUP BY customer_name
ORDER BY customer_name
;
```

```
CREATE VIEW view_expenditure AS
SELECT supplier.vendor_name, to_char(MAX(payment.payment_time::date),
'MM-DD-YYYY') last_paid_to, SUM(payment.amount) expenditure
FROM outgoing_payment
INNER JOIN payment ON outgoing_payment.outgoing_payment_id =
payment.outgoing_payment_id
INNER JOIN needs ON needs.outgoing_payment_id =
outgoing_payment.outgoing_payment_id
INNER JOIN supply_purchase_order ON
supply_purchase_order.supply_purchase_id = needs.supply_purchase_id
LEFT JOIN supplier ON supply_purchase_order.supply_id = supplier.supply_id
GROUP BY supplier.vendor_name
```

```
;
```

```
CREATE OR REPLACE PROCEDURE fill_work_shift(
    startDate timestamp
)
LANGUAGE plpgsql
AS $$
DECLARE
startTime time := '8:00 AM';
endDate date := startDate + '7 Days';
chosen_emp_id int := 1;
BEGIN

    FOR COUNTER IN 1..30 LOOP
    chosen_emp_id := ((SELECT floor(random() * (SELECT count(*) FROM
employee))));
        IF chosen_emp_id = 0 THEN
            chosen_emp_id := 1; -- EDGE CASE RANDOM PK-0
        END IF;

        startTime := date_trunc('hour', (select time ' 8:00:00' +
       random() * (time ' 18:00:00' -
                   time '8:00:00')));

       INSERT INTO work_shift(employee_id, shift_date, begin_time,
end_time) values
            (chosen_emp_id,
            (select  startDate +
       random() * ( endDate -
                    startDate)),
                   startTime,
                   startTime + '4 hours')
                   ON CONFLICT ON CONSTRAINT id_day_check DO NOTHING;
    END LOOP;
END;
$$;
```

```
CREATE OR REPLACE PROCEDURE fillOutgoingPaymentsRandomly()
LANGUAGE plpgsql
AS $$
DECLARE
outgoing_payment_insert integer := 1;
supply_order_num integer := ((SELECT floor(random() * (SELECT count(*)
FROM supply_purchase_order) + 1)));
BEGIN

    FOR COUNTER IN 1..70 LOOP
        supply_order_num := ((SELECT floor(random() * (SELECT count(*)
FROM supply_purchase_order))));
        IF supply_order_num = 0 THEN
            supply_order_num := 1; -- EDGE CASE RANDOM PK-0
        END IF;

        insert into outgoing_payment(supplier_invoice_id) values
((floor(random() * 1000000 + 1)::int));

        outgoing_payment_insert := ((SELECT count(*) FROM
outgoing_payment));
        insert into needs(supply_purchase_id, outgoing_payment_id) values
(supply_order_num, outgoing_payment_insert);


        insert into payment(payment_time, amount, payment_type_id,
outgoing_payment_id)
         values ( (select timestamp '2000-01-10 20:00:00' +
       random() * (timestamp '2020-01-20 20:00:00' -
                   timestamp '2000-01-10 10:00:00')), (SELECT
floor(random() * 10 + 15)), (SELECT floor(random() * 10 + 1)),
outgoing_payment_insert);
   END LOOP;
END;
$$;
```

```
CREATE OR REPLACE PROCEDURE fillIncomingPaymentsRandomly()
```

```
LANGUAGE plpgsql
AS $$
DECLARE
incoming_payment_insert integer := 1;
product_order_num integer := ((SELECT floor(random() * (SELECT count(*)
FROM product_order) + 1)));
BEGIN

    FOR COUNTER IN 1..100 LOOP
        product_order_num := ((SELECT floor(random() * (SELECT count(*)
FROM product_order))));
        IF product_order_num = 0 THEN
            product_order_num := 1; -- EDGE CASE RANDOM PK-0
        END IF;

        insert into incoming_payment(sales_tax) values (.0700);

        incoming_payment_insert := ((SELECT count(*) FROM
incoming_payment));
        insert into requires(p_order_number, incoming_payment_id) values
(product_order_num, incoming_payment_insert);


        insert into payment(payment_time, amount, payment_type_id,
incoming_payment_id)
         values ( (select timestamp '2000-01-10 20:00:00' +
       random() * (timestamp '2020-01-20 20:00:00' -
                   timestamp '2000-01-10 10:00:00')), (SELECT
floor(random() * 10 + 15)), (SELECT floor(random() * 10 + 1)),
incoming_payment_insert);
   END LOOP;
END;
$$;
```

```
CREATE OR REPLACE FUNCTION check_work_shift()
RETURNS TRIGGER AS $BODY$
DECLARE
```

```
placehold time := '8:00 AM';
BEGIN
    IF OLD.begin_time > OLD.end_time THEN
        placehold := end_time;
        UPDATE work_shift set begin_time = end_time, end_time = placehold
        WHERE work_shift.shift_id = NEW.shift_id;
    END IF;
    RETURN NEW;
END;
$BODY$ LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS time_switch ON work_shift;
CREATE TRIGGER time_switch
before insert ON work_shift
FOR EACH ROW EXECUTE PROCEDURE check_work_shift();
```

5.2 Programming Sections

This section will go over the server side, middle-tier-and and client-side programming I used when implementing my application. My server side programming was done with Node.js with the express framework, the middle tier programming was done using a package for node called pg that allows node.js to connect to postgres and do sql queries. My client side programming was complete using HTML, CSS, Bootstrap, JQuery, and javascript.

5.2.1 Server-side Programming

This section is the same code as the previous section, but I will go into more detail over the views and subprograms purposes as it relates to our database.

```
CREATE VIEW view_manager_scheduling AS
SELECT employee.employee_id, employee.fname || ' ' || employee.lname AS
employee_name,
work_history.pay_rate, work_history.job_title, work_shift.shift_date as
day,
work_shift.begin_time as shift_start, work_shift.end_time as shift_end
FROM employee
```

```
INNER JOIN work_history ON work_history.employee_id = employee.employee_id
INNER JOIN work_shift ON work_shift.employee_id = employee.employee_id
WHERE work_history.end_date IS NULL
order by employee_id, shift_start
;
```

This view was used to abstract away a join between the work_shift, employee, and work history for the front end calendar. I built several of the sections of the manager interface by querying the view rather than having to join all 3 everytime I queried postgres.

```
CREATE VIEW view_number_employees_working AS
SELECT work_history.job_title, COUNT(work_history.job_title)
count_of_job_type, work_shift.shift_date as day
FROM employee
INNER JOIN work_history ON work_history.employee_id = employee.employee_id
INNER JOIN work_shift ON work_shift.employee_id = employee.employee_id
GROUP BY work_history.job_title, work_shift.shift_date
ORDER BY work_shift.shift_date
;
```

This view allows a manager to see the number of employees working by job type each day.

```
CREATE VIEW view_positive_revenue AS
SELECT customer.fname || ' ' || customer.lname as customer_name,
to_char(MAX(payment.payment_time::date), 'MM-DD-YYYY') last_bought,
SUM(payment.amount) revenue
FROM incoming_payment
INNER JOIN payment ON incoming_payment.incoming_payment_id =
payment.incoming_payment_id
INNER JOIN requires ON requires.incoming_payment_id =
incoming_payment.incoming_payment_id
INNER JOIN product_order ON requires.p_order_number =
product_order.p_order_number
LEFT JOIN customer ON product_order.customer_id = customer.customer_id
GROUP BY customer_name
ORDER BY customer_name
;
```

This view shows all positive payments that are incoming to the store from customer orders. My revenue report was partially generated from the view here.

```
CREATE VIEW view_expenditure AS
SELECT supplier.vendor_name, to_char(MAX(payment.payment_time::date),
'MM-DD-YYYY') last_paid_to, SUM(payment.amount) expenditure
FROM outgoing_payment
INNER JOIN payment ON outgoing_payment.outgoing_payment_id =
payment.outgoing_payment_id
INNER JOIN needs ON needs.outgoing_payment_id =
outgoing_payment.outgoing_payment_id
INNER JOIN supply_purchase_order ON
supply_purchase_order.supply_purchase_id = needs.supply_purchase_id
LEFT JOIN supplier ON supply_purchase_order.supply_id = supplier.supply_id
GROUP BY supplier.vendor_name
;
```

This view shows all outgoing payments to suppliers and their names. My revenue report generation also used this view.

```
CREATE OR REPLACE PROCEDURE fill_work_shift(
    startDate timestamp
)
LANGUAGE plpgsql
AS $$
DECLARE
startTime time := '8:00 AM';
endDate date := startDate + '7 Days';
chosen_emp_id int := 1;
BEGIN

    FOR COUNTER IN 1..30 LOOP
    chosen_emp_id := ((SELECT floor(random() * (SELECT count(*) FROM
employee))));
        IF chosen_emp_id = 0 THEN
            chosen_emp_id := 1; -- EDGE CASE RANDOM PK-0
        END IF;
        startTime := date_trunc('hour', (select time ' 8:00:00' +
       random() * (time ' 18:00:00' -
                   time '8:00:00')));

       INSERT INTO work_shift(employee_id, shift_date, begin_time,
end_time) values
             (chosen_emp_id,
             (select  startDate +
       random() * ( endDate -
                    startDate)),
                   startTime,
                   startTime + '4 hours')
                   ON CONFLICT ON CONSTRAINT id_day_check DO NOTHING;
    END LOOP;
END;
$$;
```

This function takes a day as a parameter then randomly generates shifts to be input into the work_shift table.

```
CREATE OR REPLACE PROCEDURE fillOutgoingPaymentsRandomly()
LANGUAGE plpgsql
AS $$
DECLARE
outgoing_payment_insert integer := 1;
supply_order_num integer := ((SELECT floor(random() * (SELECT count(*)
FROM supply_purchase_order) + 1)));
BEGIN

    FOR COUNTER IN 1..70 LOOP
        supply_order_num := ((SELECT floor(random() * (SELECT count(*)
FROM supply_purchase_order))));
        IF supply_order_num = 0 THEN
            supply_order_num := 1; -- EDGE CASE RANDOM PK-0
        END IF;

        insert into outgoing_payment(supplier_invoice_id) values
((floor(random() * 1000000 + 1)::int));

        outgoing_payment_insert := ((SELECT count(*) FROM
outgoing_payment));
        insert into needs(supply_purchase_id, outgoing_payment_id) values
(supply_order_num, outgoing_payment_insert);


        insert into payment(payment_time, amount, payment_type_id,
outgoing_payment_id)
         values ( (select timestamp '2000-01-10 20:00:00' +
       random() * (timestamp '2020-01-20 20:00:00' -
                   timestamp '2000-01-10 10:00:00')), (SELECT
floor(random() * 10 + 15)), (SELECT floor(random() * 10 + 1)),
outgoing_payment_insert);
   END LOOP;
END;
$$;
```

This function randomly generates outgoing payments in the store over a range of 20 years. On the front end there is a button that calls this function to help show that my reports were dynamic.

```
CREATE OR REPLACE PROCEDURE fillIncomingPaymentsRandomly()
LANGUAGE plpgsql
AS $$
DECLARE
incoming_payment_insert integer := 1;
product_order_num integer := ((SELECT floor(random() * (SELECT count(*)
FROM product_order) + 1)));
BEGIN

    FOR COUNTER IN 1..100 LOOP
        product_order_num := ((SELECT floor(random() * (SELECT count(*)
FROM product_order))));
        IF product_order_num = 0 THEN
            product_order_num := 1; -- EDGE CASE RANDOM PK-0
        END IF;
        insert into incoming_payment(sales_tax) values (.0700);

        incoming_payment_insert := ((SELECT count(*) FROM
incoming_payment));
        insert into requires(p_order_number, incoming_payment_id) values
(product_order_num, incoming_payment_insert);


        insert into payment(payment_time, amount, payment_type_id,
incoming_payment_id)
         values ( (select timestamp '2000-01-10 20:00:00' +
       random() * (timestamp '2020-01-20 20:00:00' -
                  timestamp '2000-01-10 10:00:00')), (SELECT
floor(random() * 10 + 15)), (SELECT floor(random() * 10 + 1)),
incoming_payment_insert);
   END LOOP;
END;
$$;
```

This procedure randomly generates incoming payments and inserts them into the payments table. It is called by a button on my front end.

```
CREATE OR REPLACE FUNCTION check_work_shift()
RETURNS TRIGGER AS $BODY$
DECLARE
placehold time := '8:00 AM';
BEGIN
    IF OLD.begin_time > OLD.end_time THEN
        placehold := end_time;
        UPDATE work_shift set begin_time = end_time, end_time = placehold
        WHERE work_shift.shift_id = NEW.shift_id;
    END IF;
    RETURN NEW;
END;
$BODY$ LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS time_switch ON work_shift;
```

```
CREATE TRIGGER time_switch
before insert ON work_shift
FOR EACH ROW EXECUTE PROCEDURE check_work_shift();
```

This is a trigger I used to make my randomly generated work shifts be consistent. If a end time of a shift appears between a beginning time of a shift it will switch the two times so that the data going into the database is consistent.

5.2.2 Middle Tier Programming

The middle tier of my database was done using node.js with the express framework, and a package for node called pg. PG is a package for node that allows for an easy connection between postgres and allows queries to be performed easily.

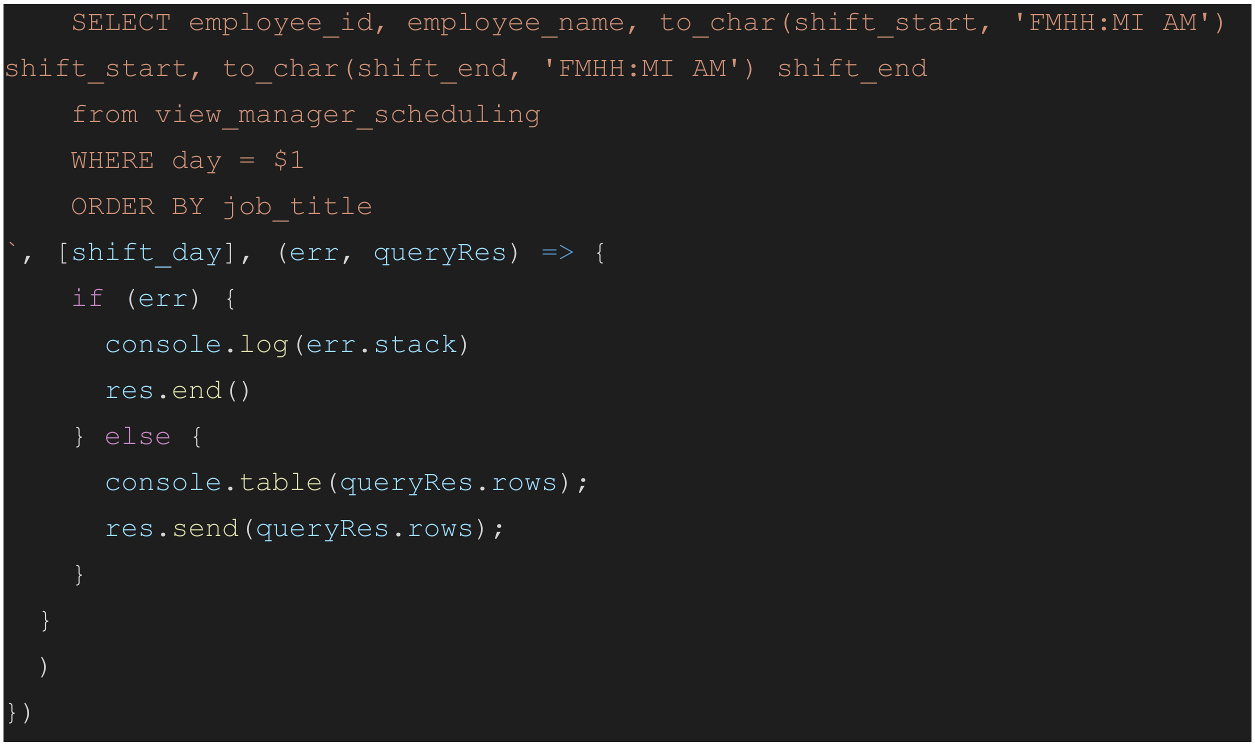
Code for connecting to database
With Node and a package in it called PG connecting to a database is really simple to get setup and started.
The Code:

```
const { Client } = require('pg');
const client = new Client({
    user: 'joey',
    password: 'password',
    host: 'localhost',
    port: 5432,
    database: 'flowershop',
  });

  client.connect();
```

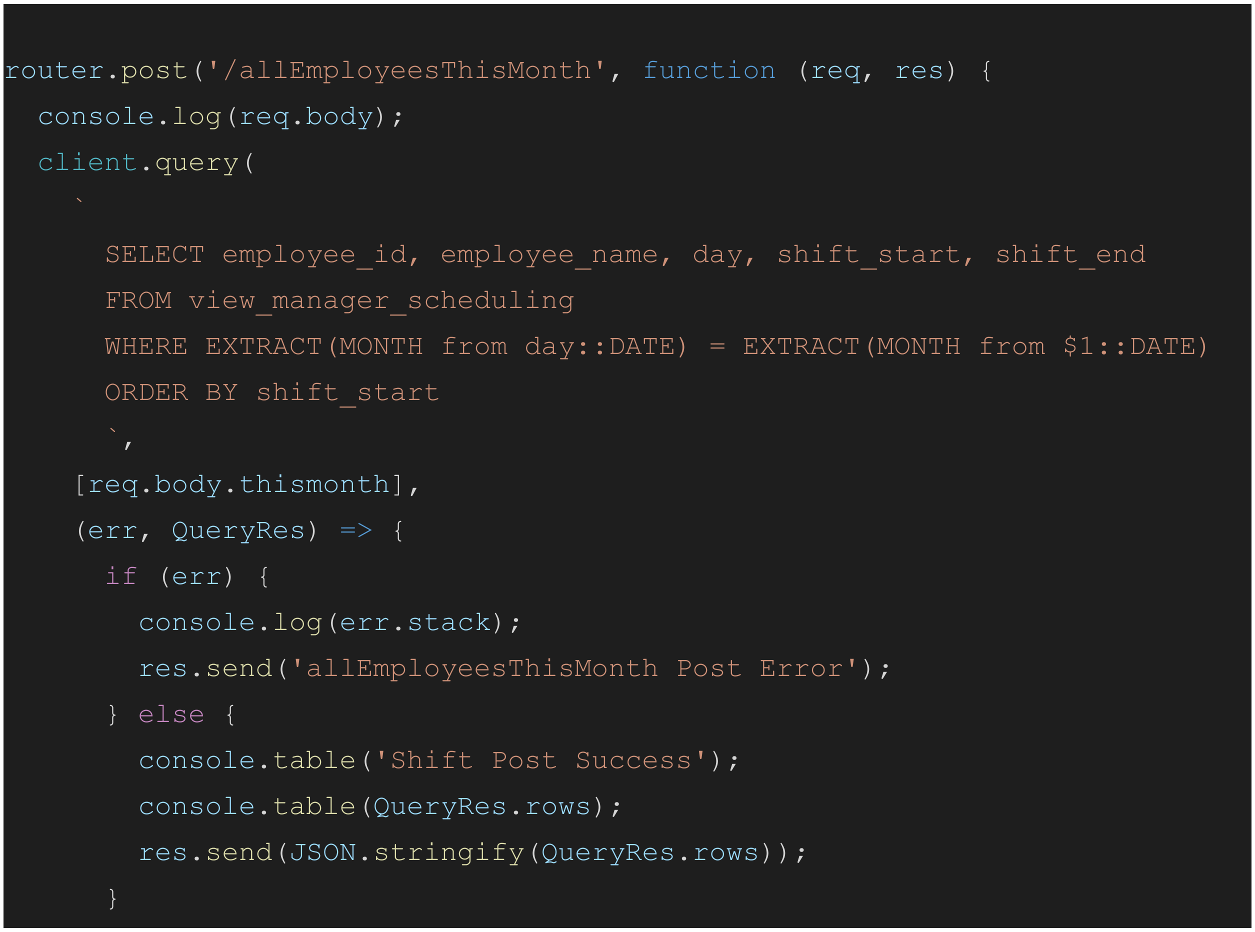
Essentially what this does is creates an object constructor that can be used to create objects of the client type associated with the pg library. I then create a client object that holds the information of my database that can be queried to.

Code sections which use view

```
router.post('/getEmployeeForDay', (req, res) => {
  let shift_day = req.body.day;
  console.log(shift_day);
  client.query(`
```



```
    SELECT employee_id, employee_name, to_char(shift_start, 'FMHH:MI AM')
shift_start, to_char(shift_end, 'FMHH:MI AM') shift_end
    from view_manager_scheduling
    WHERE day = $1
    ORDER BY job_title
`, [shift_day], (err, queryRes) => {
    if (err) {
      console.log(err.stack)
      res.end()
    } else {
      console.table(queryRes.rows);
      res.send(queryRes.rows);
    }
  }
  )
})
```

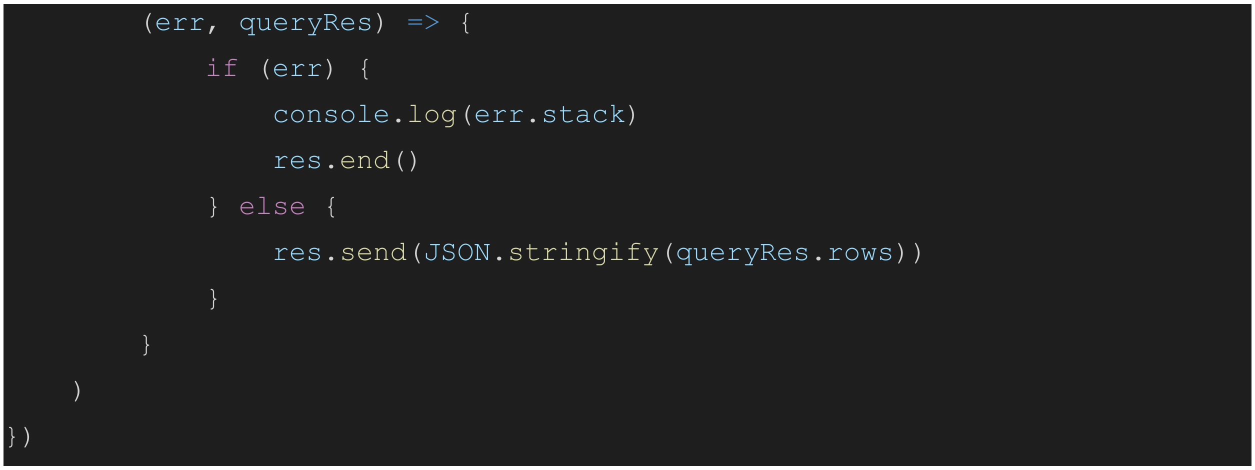


```
router.post('/allEmployeesThisMonth', function (req, res) {
  console.log(req.body);
  client.query(
    `
      SELECT employee_id, employee_name, day, shift_start, shift_end
      FROM view_manager_scheduling
      WHERE EXTRACT(MONTH from day::DATE) = EXTRACT(MONTH from $1::DATE)
      ORDER BY shift_start
      `,
    [req.body.thismonth],
    (err, QueryRes) => {
      if (err) {
        console.log(err.stack);
        res.send('allEmployeesThisMonth Post Error');
      } else {
        console.table('Shift Post Success');
        console.table(QueryRes.rows);
        res.send(JSON.stringify(QueryRes.rows));
      }
    }
  );
});
```

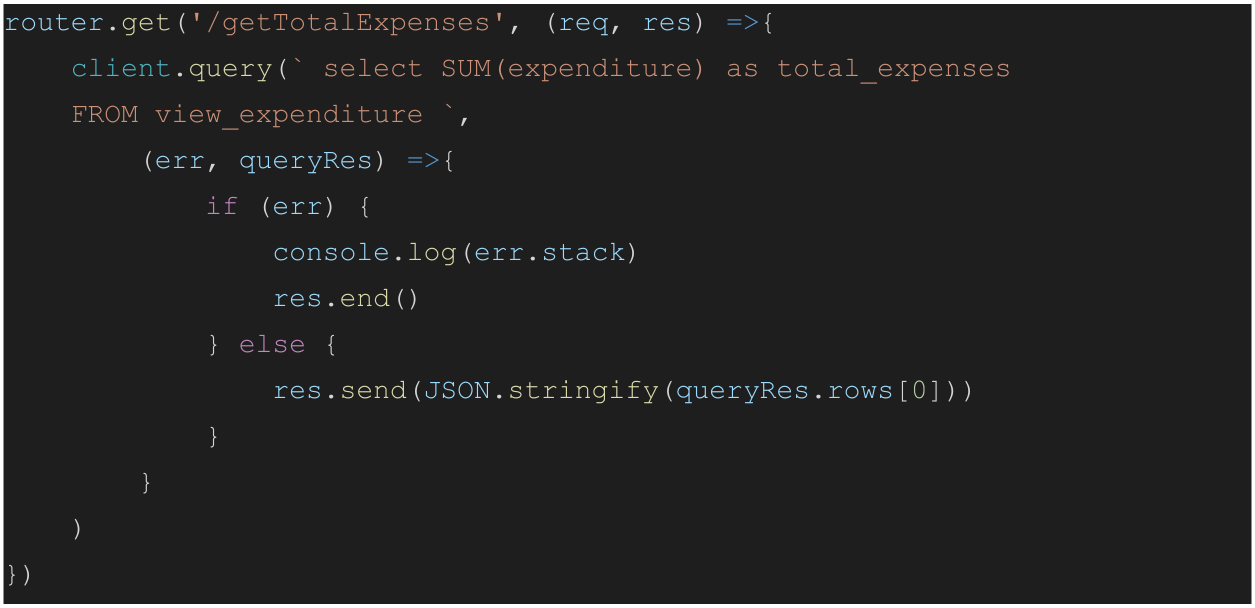
```
router.get('/getRevenue', (req, res) => {
    client.query(`select * from view_positive_revenue LIMIT 12`,
        (err, queryRes) => {
            if (err) {
                console.log(err.stack)
                res.end()
            } else {
                console.table(queryRes.rows);
                res.send(JSON.stringify(queryRes.rows))
            }
        }
    )
})
```

```
router.get('/getTotalRevenue', (req, res) => {
    client.query(`SELECT SUM(revenue) as total_revenue FROM
view_positive_revenue`,
        (err,queryRes) => {
            if (err){
                console.log(err.stack)
                res.end()
            } else {
                res.send(JSON.stringify(queryRes.rows[0]))
            }
        }
    )
})
```

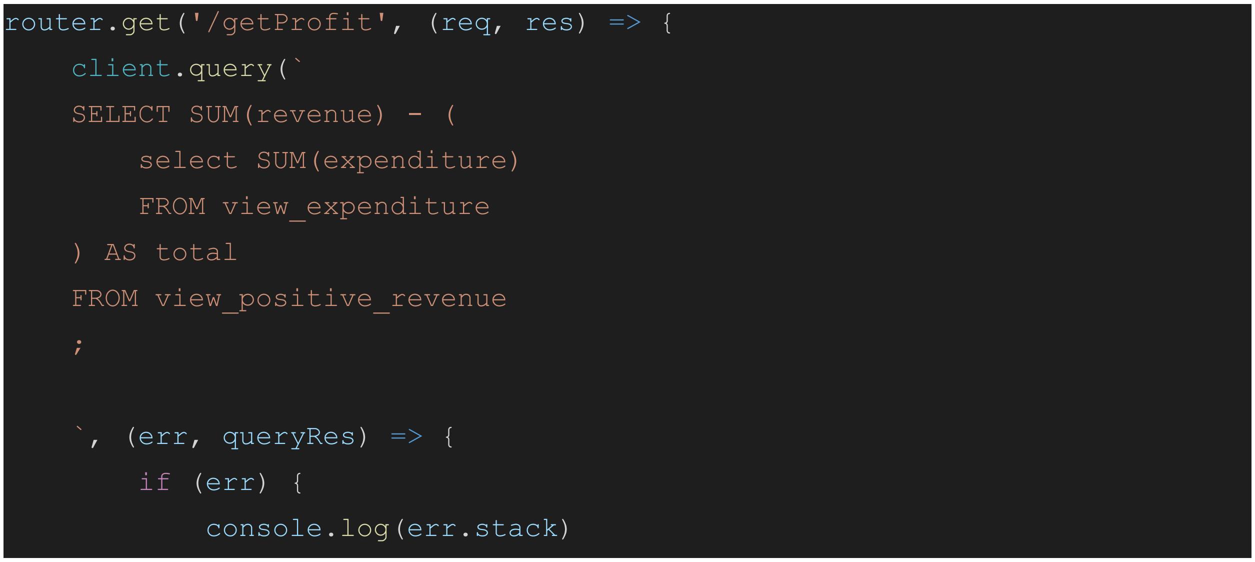
```
router.get('/getExpenses', (req, res) => {

    client.query(`SELECT * FROM view_expenditure`,
```



```
        (err, queryRes) => {
            if (err) {
                console.log(err.stack)
                res.end()
            } else {
                res.send(JSON.stringify(queryRes.rows))
            }
        }
    )
})
```



```
router.get('/getTotalExpenses', (req, res) =>{
    client.query(` select SUM(expenditure) as total_expenses
    FROM view_expenditure `,
        (err, queryRes) =>{
            if (err) {
                console.log(err.stack)
                res.end()
            } else {
                res.send(JSON.stringify(queryRes.rows[0]))
            }
        }
    )
})
```

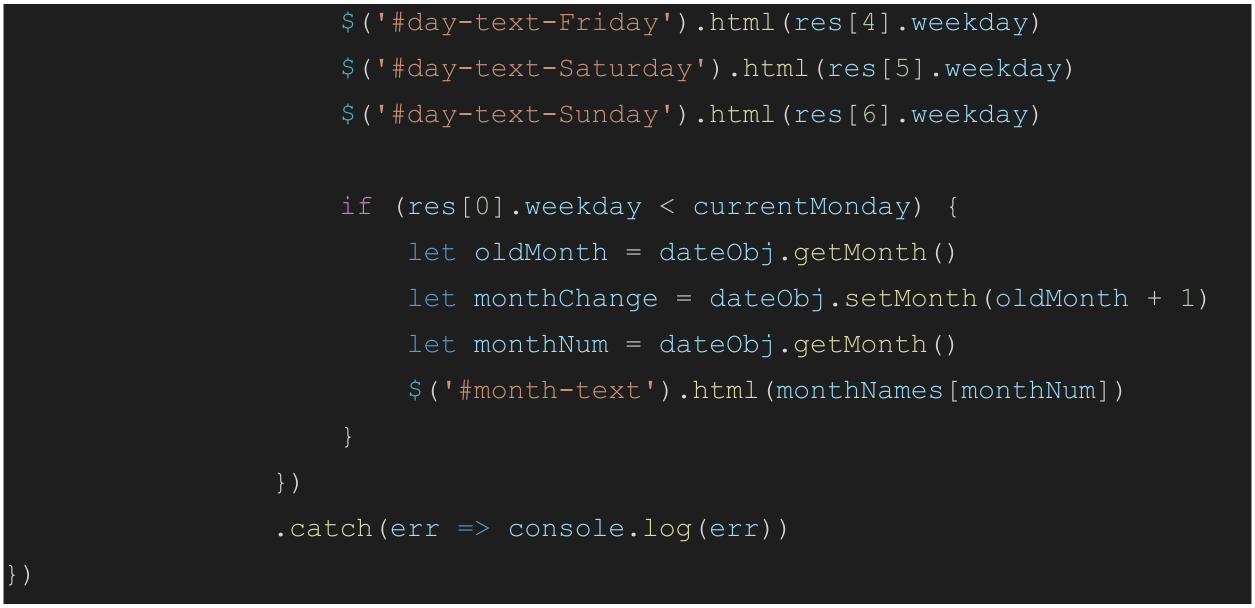


```
router.get('/getProfit', (req, res) => {
    client.query(`
    SELECT SUM(revenue) - (
        select SUM(expenditure)
        FROM view_expenditure
    ) AS total
    FROM view_positive_revenue
    ;

    `, (err, queryRes) => {
        if (err) {
            console.log(err.stack)
            res.end()
        } else {
            res.send(JSON.stringify(queryRes.rows[0]))
        }
    }

    )
})
```

5.2.3 Client-side programming

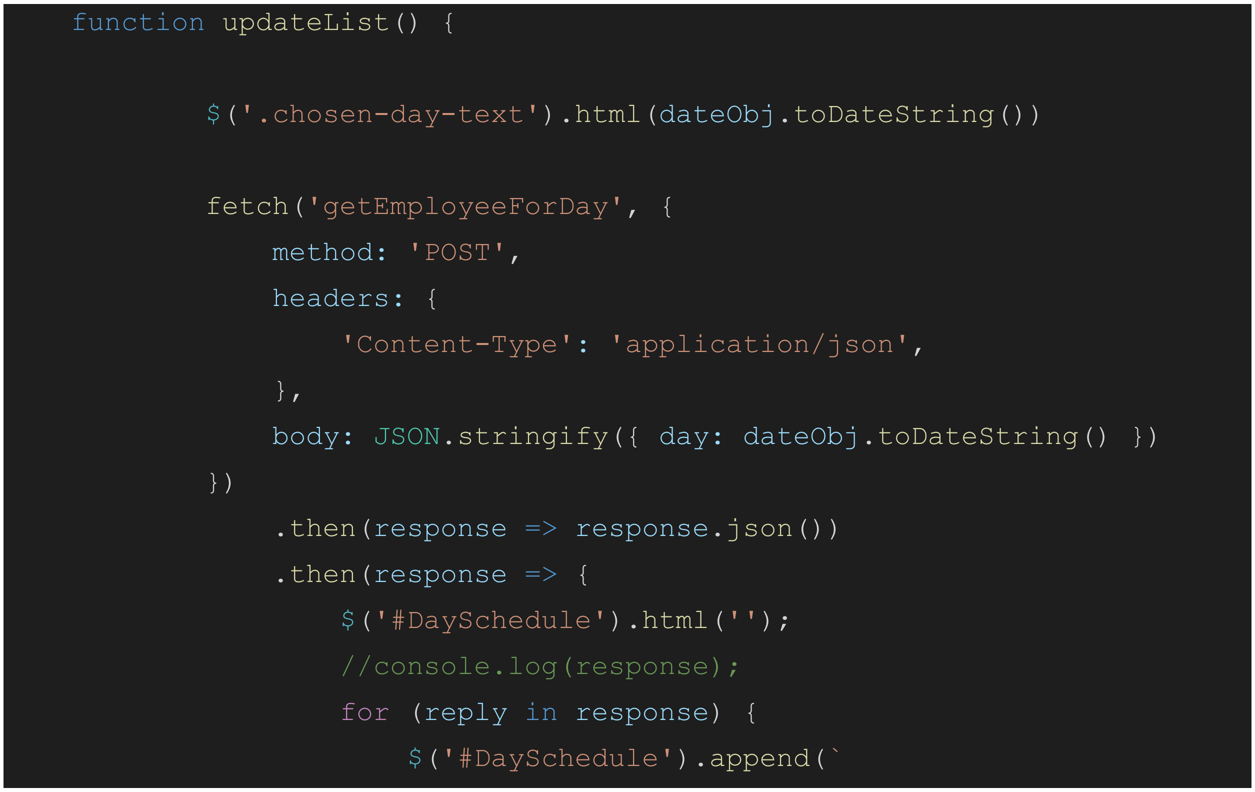
My client side code was pretty extensive and there is a lot of interaction on the interface with the user. I'll provide a few snippets showing the interaction but not all could fit here reasonably.

```
  $('#dateRightArrow').click(function () {
            let currentMonday = $("#day-text-Monday").html();
            console.log(currentMonday);
            let mondayMonth = $('#month-text').html()
            let queryString = mondayMonth + " " + currentMonday + " " +
"2020";
            console.log(queryString);
            // console.log('got to here')
            fetch('/getCalendarBlock', {
                method: 'POST',
                headers: {
                    'Content-Type': 'application/json',
                },
                body: JSON.stringify({ sentDay: queryString, timeDiff: 7
})
            })
                .then((res) => res.json())
                .then((res) => {
                    $('#day-text-Monday').html(res[0].weekday)
                    $('#day-text-Tuesday').html(res[1].weekday)
                    $('#day-text-Wednesday').html(res[2].weekday)
                    $('#day-text-Thursday').html(res[3].weekday)
```



```
                    $('#day-text-Friday').html(res[4].weekday)
                    $('#day-text-Saturday').html(res[5].weekday)
                    $('#day-text-Sunday').html(res[6].weekday)

                    if (res[0].weekday < currentMonday) {
                        let oldMonth = dateObj.getMonth()
                        let monthChange = dateObj.setMonth(oldMonth + 1)
                        let monthNum = dateObj.getMonth()
                        $('#month-text').html(monthNames[monthNum])
                    }
                })
                .catch(err => console.log(err))
})
```

This is a function that gets called whenever the right arrow on the front end gets clicked. Essentially it shifts all the shown days over by one. I discovered postgres handles date and time manipulation significantly better than javascripts built in options, so I bound this button to a query that gets the days of the week and returns them to the front end.



```
    function updateList() {

            $('.chosen-day-text').html(dateObj.toDateString())

            fetch('getEmployeeForDay', {
                method: 'POST',
                headers: {
                    'Content-Type': 'application/json',
                },
                body: JSON.stringify({ day: dateObj.toDateString() })
            })
                .then(response => response.json())
                .then(response => {
                    $('#DaySchedule').html('');
                    //console.log(response);
                    for (reply in response) {
                        $('#DaySchedule').append(`
                <a class="right-side-emp-list"
id="emp-id-right-column-${response[reply].employee_id}"
style="text-decoration: none;" href="#">
                    <div class="col justify-content-between py-1">
                        <h3
class="text-dark">${response[reply].employee_name}</h3>
                        <h3
class="text-dark">${response[reply].shift_start} -
${response[reply].shift_end}</h3>
                    </div>
                 </a>
              `)
                    }
                })
        }
```

This is a function that gets called everytime one of the calendar days along the bottom of my interface is clicked. I use a “template literal” that lets me write html inside of a javascript function and insert a query response into a page. This allows my page to be fairly dynamic, as every time a day is clicked it destroys the old list of names shown on the right and then inserts the new list of names scheduled for the day instead.

5.3 Survey Questions

Of the outcomes I would put the order of knowledge from best to worst as 2, 1, 3, 4 but all were improved upon in this class. I genuinely feel I learned all of these well enough to put a 10 for each, but there’s always room for improvement.

Outcome	**Joseph Shafer’s Answers**
An ability to analyze a problem, and identify and define the computing requirements and	10

specifications appropriate to its solution.	
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.	10
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.	10