DTSC 691 Project Submission

Mini-Mart Database

Table of Contents

Background

The goal of my project is to showcase my ability in developing and implementing a functional database management system that will help manage a fictional convenience store business named Mini-Mart. The database will be easy to use, while also providing some unique insights on how to make the business more profitable. The database will support the relationships between the business and the community it serves. It will document and track employee information, item inventory levels, customer purchase history, among other things. The use of my database will increase accuracy, accountability, profits, and more.

Currently, the business uses an Excel spreadsheet to keep track of everything. However, as we already know, Microsoft Excel is not a viable tool to support the management of an entire database system. Along with having a limit to the number of records an Excel spreadsheet can contain, the performance of the application slows down considerably as you incrementally reach those limits. The lack of version control and the lack of data manipulation tools in Excel also prevents a business from running optimally. Another issue that comes up when using Excel is that the tables across the organization do not automatically connect to each other. This prevents the seamless transfer of information from one table to another (Holmes, 2020).

Databases in general have a lot of benefits to any organization. Through Postgresql, Mini-Mart will have fewer data redundancies and fewer inconsistencies than a typical file-processing system. The business' information will be easily accessible to anyone who is authorized. A centralized database will make it efficient to retrieve all sorts of information in comparison to isolated data locations that make collecting and analyzing data more cumbersome than it has to be.

To complement the business' new database, a Tableau dashboard will be developed so stakeholders can quickly access relevant metrics that will help them support the business further. With the implementation of these new tools and resources, The Mini-Mart will be able to better serve their customers and generate more profits moving forward.

Problem Objective

Have a Title Page with a Table of Contents section for the entire project

Data and Functional Requirements

- Describe the information that will be stored in the database while considering the relationships and constraints for the entities
- List the functional requirements

Develop an ER Model using LucidChart

Construct a Relational Database Schema Diagram using LucidChart

SQL DDL

- Create DDL statements that define the relations identified in the schema
    - Create eight tables storing different types of information
    - Create two views definitions and describe who would use these views
    - Create two general SQL functions and describe its purpose
    - Create one trigger and describe its purpose
    - Create one stored procedure and describe its purpose and function

SQL DML

- For each table, provide the following:
    - Five insert statements
    - One update statement
    - One delete statement
    - Consider referential integrity constraints for all statements

Construct eight SQL queries

- For each query, state the query in plain english along with the business relevance

Develop a web application in Python using the Flask framework to interact with the database

- The code will connect to the database and will perform the following actions:
    - Read some data from the database and display it to the user
    - Change some data in the database by updating or deleting record(s)

Create a Tableau dashboard that displays relevant analytics to measure the business' performance and that helps make informed decisions

- Connect the database to the Tableau workbook
- Develop an easy-to-consume dashboard displaying relevant business metrics

Record a video presentation walking through the entire project and its requirements

- Video will be recorded in Zoom and uploaded to the Brightspace Media Gallery

The entire project will be submitted in a single document using the Project Submission Template as a guide. The code provided in the document will be able to be copy and pasted for testing and evaluation purposes.

The main goal of this project is to create a working postgres database that will manage the business efficiently and effectively. It will provide stakeholders with relevant information of the performance of the business while also identifying needs of the convenience store.

Data Description

A. Data and Functional Requirements

1. Stores are characterized by a store name, address, phone number, and previous year sales along with current year sales.

2. Customers are characterized by a customer ID#, name, address, phone number, birthday, a date for when they became customers and the amount of transactions they have with the business along with their total cumulative spend.

3. Employees are characterized by an employee ID#, job ID#, name, address, phone number, birthday, a date for when they became an employee and the amount of vacation hours they have along with the amount of sick hours they have.

4. Jobs are characterized by a job ID#, job title, employee ID#, rate, and the store they are associated with.

5. Vendors are characterized by a vendor ID#, name, address, phone number and a date for when they became a partnered vendor for the business.

6. Items are characterized by an item ID#, name, inventory amount, quantity sold amount, description, selling price and a cost price along with the vendor id# that is associated with the item.

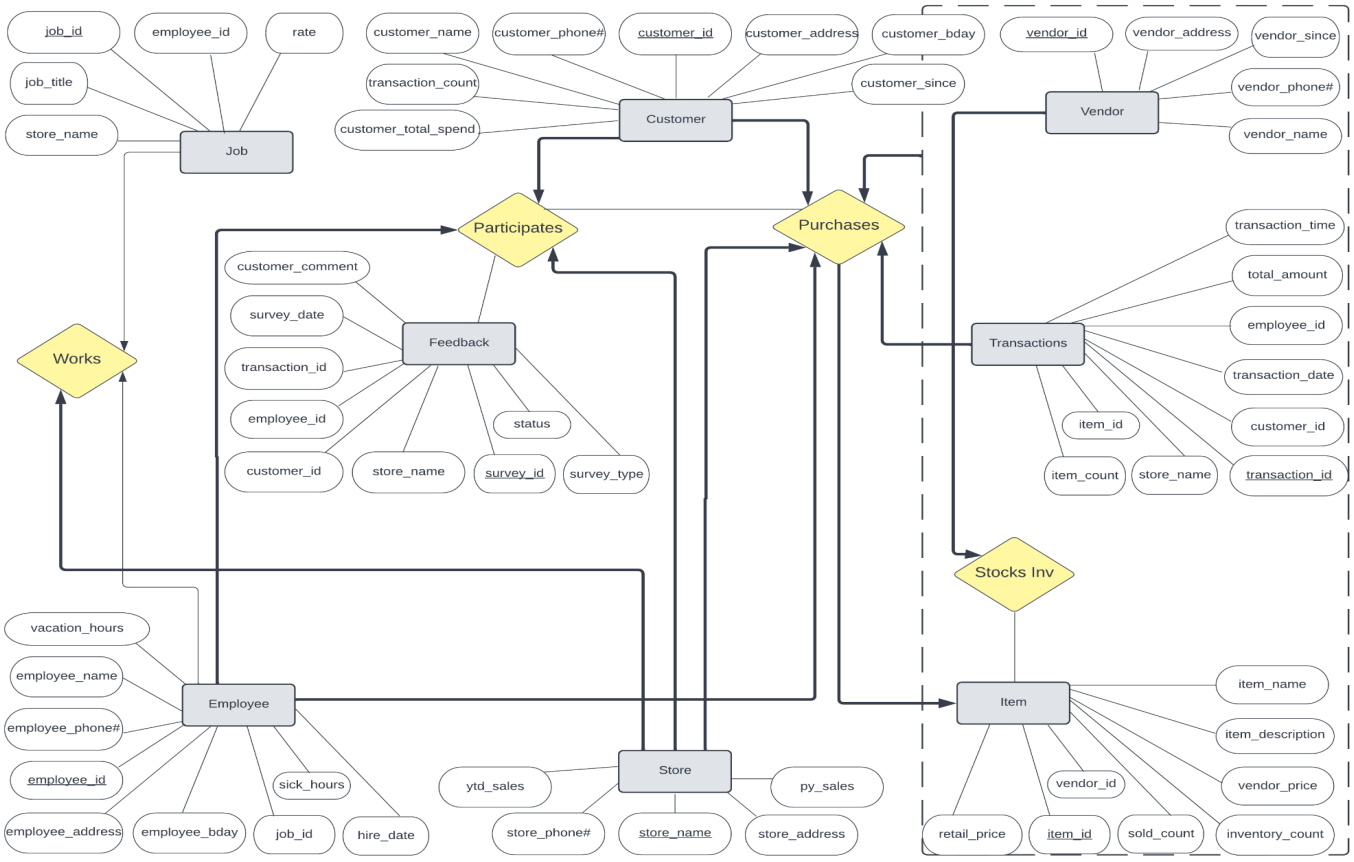
7. Transactions are characterized by a transaction ID#, the business associated with the transaction, date, time, which employee performs the transaction, tracks the customer of the transaction, the item ID#, the quantity of items for each associated item ID#, and the total amount of the transaction overall.

8. Feedback surveys are characterized by a survey ID#, the business associated with the survey, the type of survey (+/- or other), the comments made by the customer who submitted the feedback, their customer ID#, the transaction that is associated with the survey, which employee performed the transaction, the date for when the survey was completed, and a current status for the survey.

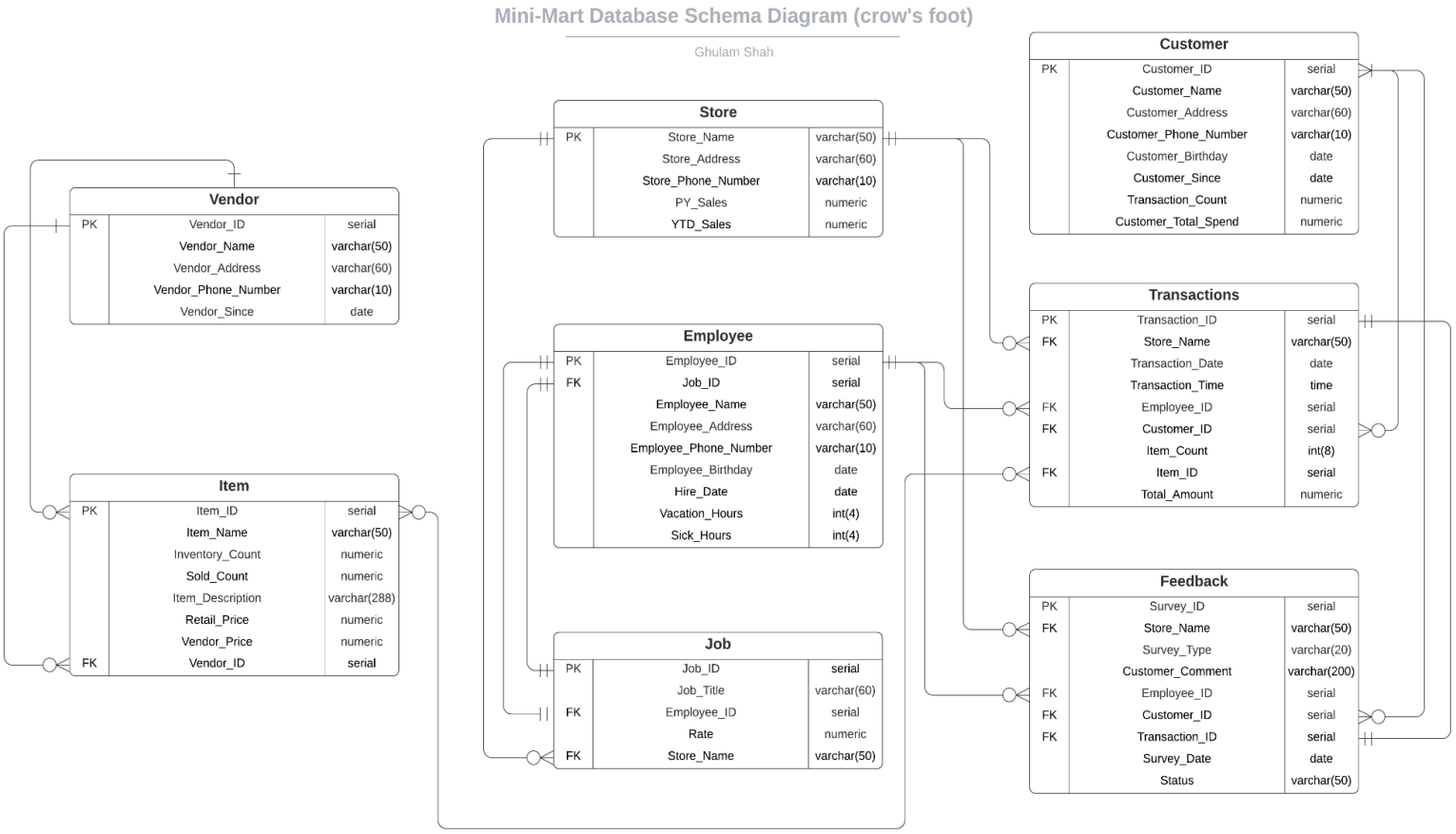
9. Each transaction can only be performed by one employee. Only one transaction can be associated with a feedback survey. Only one customer can be associated with a single transaction.

10. Only one employee can have one job ID and only one job ID can be associated with one employee.
11. A vendor can provide the business with many items. A customer can have many transactions with the store. A transaction can have many items. An employee can perform several different transactions.
12. A store can have many customers, employees, surveys, jobs, transactions, vendors and items.
13. We would like to track our customer's transactions, feedback surveys, and the cumulative amount they have spent at the store.

B. ER Model



C. Relational Database Schema



D. Changes from Project Proposal to Project Submission

- Removed unnecessary/repetitive columns from Vendor and Feedback tables.
- Changed table name from Transaction to Transactions since the former is a reserved keyword in postgresql.
- Changed column name from Comment to Customer_Comment since the former is a reserved keyword in postgresql.
- Changed the data type for most primary keys in the database from integer to serial. Also changed money to numeric for other fields.
- Will no longer use a powerpoint slideshow in my presentation.
- Additional software utilized: Microsoft Excel and Cobbl.
- Used PostgreSQL 12 instead of PostgreSQL 14.

Software

I utilized a data modeling tool called **LucidChart** to generate my ER diagram and my database schema. This tool helped me generate the Data Definition Language used to create the MiniMart database. The majority of the project was built in **PostgreSQL 12** - **pgAdmin 4** version 6.7, which manages the database for me. To create some fictional data I used **Cobbl.io** to generate two csv files for two of my data tables. After building the Data Manipulation Language, I implemented a connection from the database to create a **Tableau** dashboard to support the business. For my web application I utilized the **Flask** framework in **Python** 3.8.3 using **PyCharm** Edu 2020.1. I also included the use of **SQLAlchemy** and **Psycopg2** to support my Flask application. I recorded my project walkthrough presentation using **Zoom**.

Analysis

A. Data Definition and Data Manipulation Language

```
---- Table Creation ----

CREATE TABLE Vendor (
  Vendor_ID serial UNIQUE,
  Vendor_Name varchar(50),
  Vendor_Address varchar(60),
  Vendor_Phone_Number varchar(10),
  Vendor_Since date,
  PRIMARY KEY (Vendor_ID)
);

CREATE TABLE Store (
  Store_Name varchar(50) NOT NULL UNIQUE,
  Store_Address varchar(60),
  Store_Phone_Number varchar(10),
  PY_Sales numeric,
  YTD_Sales numeric,
  PRIMARY KEY (Store_Name)
);
```

```
CREATE TABLE Employee (
  Employee_ID serial UNIQUE,
  Job_ID serial NOT NULL,
  Employee_Name varchar(50),
  Employee_Address varchar(60),
  Employee_Phone_Number varchar(10),
  Employee_Birthday date,
  Hire_Date date,
  Vacation_Hours int NOT NULL,
  Sick_Hours int NOT NULL,
  PRIMARY KEY (Employee_ID)
);

CREATE TABLE Transactions (
  Transaction_ID serial UNIQUE,
  Store_Name varchar(50),
  Transaction_Date date,
  Transaction_Time time,
  Employee_ID serial NOT NULL,
  Customer_ID serial NOT NULL,
  Item_Count int NOT NULL,
  Item_ID serial NOT NULL,
  Total_Amount numeric,
  PRIMARY KEY (Transaction_ID),
  CONSTRAINT FK_Transactions_Employee_ID
    FOREIGN KEY (Employee_ID)
      REFERENCES Employee(Employee_ID),
  CONSTRAINT FK_Transactions_Store_Name
    FOREIGN KEY (Store_Name)
      REFERENCES Store(Store_Name)
);

CREATE TABLE Customer (
  Customer_ID serial UNIQUE,
  Customer_Name varchar(50),
  Customer_Address varchar(60),
```

```
  Customer_Phone_Number varchar(10),
  Customer_Birthday date,
  Customer_Since date,
  Transaction_Count numeric,
  Customer_Total_Spend numeric,
  PRIMARY KEY (Customer_ID)
);

CREATE TABLE Feedback (
  Survey_ID serial UNIQUE,
  Store_Name varchar(50),
  Survey_Type varchar(20),
  Customer_Comment varchar(200),
  Employee_ID serial NOT NULL,
  Customer_ID serial NOT NULL,
  Transaction_ID serial NOT NULL,
  Survey_Date date,
  Status varchar(50),
  PRIMARY KEY (Survey_ID),
  CONSTRAINT FK_Feedback_Store_Name
    FOREIGN KEY (Store_Name)
      REFERENCES Store(Store_Name),
  CONSTRAINT FK_Feedback_Employee_ID
    FOREIGN KEY (Employee_ID)
      REFERENCES Employee(Employee_ID),
  CONSTRAINT FK_Feedback_Transaction_ID
    FOREIGN KEY (Transaction_ID)
      REFERENCES Transactions(Transaction_ID)
);

CREATE TABLE Job (
  Job_ID serial UNIQUE,
  Job_Title varchar(60),
  Employee_ID serial NOT NULL,
  Rate numeric,
  Store_Name varchar(50),
  PRIMARY KEY (Job_ID),
```

```
  CONSTRAINT FK_Job_Store_Name
    FOREIGN KEY (Store_Name)
      REFERENCES Store(Store_Name)
);

CREATE TABLE Item (
  Item_ID serial UNIQUE,
  Item_Name varchar(50),
  Inventory_Count numeric,
  Sold_Count numeric,
  Item_Description varchar(288),
  Retail_Price numeric,
  Vendor_Price numeric,
  Vendor_ID serial NOT NULL,
  PRIMARY KEY (Item_ID),
  CONSTRAINT FK_Item_Item_ID
    FOREIGN KEY (Vendor_ID)
      REFERENCES Vendor(Vendor_ID)
);




-- Insert / Update / Delete Statements --

---- Store ----

-- Insert Values --

INSERT INTO Store (Store_Name, Store_Address, Store_Phone_Number,
PY_Sales, YTD_Sales)
VALUES ('The Mini-Mart', '105 Reed Street, Dover, DE 19904', '3024215555',
1000453.02, 350650.08);

-- (No need to insert any more rows into this table)

-- Update --
```

```
UPDATE Store
SET Store_Phone_Number = 3027505555;

-- Delete --

-- (No need to ever delete anything from the Store table) --


---- Employee ----

-- Insert Values --

INSERT INTO Employee (Employee_ID, Job_ID, Employee_Name,
Employee_Address, Employee_Phone_Number,
                                Employee_Birthday, Hire_Date, Vacation_Hours,
Sick_Hours)
VALUES ('00011', '005', 'Chris Tansey', '8175 Smith Lane, Dover, DE 19925',
'3028914510',
        '10/25/1973', '05/12/2015', '85', '51');

INSERT INTO Employee (Employee_ID, Job_ID, Employee_Name,
Employee_Address, Employee_Phone_Number,
                                Employee_Birthday, Hire_Date, Vacation_Hours,
Sick_Hours)
VALUES ('00017', '007', 'John Klomich', '216 East Landing Street, Dover, DE
19925', '3029104155',
        '11/29/1975', '09/10/2017', '45', '33');

INSERT INTO Employee (Employee_ID, Job_ID, Employee_Name,
Employee_Address, Employee_Phone_Number,
                                Employee_Birthday, Hire_Date, Vacation_Hours,
Sick_Hours)
VALUES ('00024', '025', 'Scott Wolfram', '136 Clover Lane, Newark, DE 19999',
'8569491877',
        '01/09/1982', '06/19/2018', '33', '24');
```

```
INSERT INTO Employee (Employee_ID, Job_ID, Employee_Name,
Employee_Address, Employee_Phone_Number,
                                Employee_Birthday, Hire_Date, Vacation_Hours,
Sick_Hours)
VALUES ('00043', '104', 'Chris Kershaw', '16 Hilton Road, Newark, DE 19999',
'8566481516',
        '05/23/1987', '03/23/2019', '22', '18');

INSERT INTO Employee (Employee_ID, Job_ID, Employee_Name,
Employee_Address, Employee_Phone_Number,
                                Employee_Birthday, Hire_Date, Vacation_Hours,
Sick_Hours)
VALUES ('00055', '109', 'Marcia McCord', '55 Holly Lane, Newark, DE 19999',
'8562823355',
        '04/01/1985', '04/22/2019', '20', '14');

INSERT INTO Employee (Employee_ID, Job_ID, Employee_Name,
Employee_Address, Employee_Phone_Number,
                                Employee_Birthday, Hire_Date, Vacation_Hours,
Sick_Hours)
VALUES ('00067', '121', 'Jake Harris', '24 Waverly Place, Newark, DE 19999',
'8567901544',
        '03/11/1989', '12/03/2021', '6', '3');

-- Update --

UPDATE Employee
SET Employee_Name = 'Marcia Parks'
WHERE Employee_ID = 00055;

-- Delete --

DELETE FROM Employee
WHERE Employee_ID = 00067
RETURNING *;
```

---- Job ----

-- Insert Values --

```
INSERT INTO Job (Job_ID, Job_Title, Employee_ID, Rate, Store_Name)
VALUES ('005', 'Store Manager', '00011', '40.00', 'The Mini-Mart');

INSERT INTO Job (Job_ID, Job_Title, Employee_ID, Rate, Store_Name)
VALUES ('007', 'Assistant Manager', '00017', '32.00', 'The Mini-Mart');

INSERT INTO Job (Job_ID, Job_Title, Employee_ID, Rate, Store_Name)
VALUES ('025', 'Shift Supervisor', '00024', '22.00', 'The Mini-Mart');

INSERT INTO Job (Job_ID, Job_Title, Employee_ID, Rate, Store_Name)
VALUES ('104', 'Cashier', '00043', '13.00', 'The Mini-Mart');

INSERT INTO Job (Job_ID, Job_Title, Employee_ID, Rate, Store_Name)
VALUES ('109', 'Cashier', '00055', '11.00', 'The Mini-Mart');

INSERT INTO Job (Job_ID, Job_Title, Employee_ID, Rate, Store_Name)
VALUES ('210', 'Custodian', '00101', '9.00', 'The Mini-Mart');
```

-- Update --

```
UPDATE Job
SET Rate = 12.00
WHERE Job_ID = 109;
```

-- Delete --

```
DELETE FROM Job
WHERE Job_ID = 210
RETURNING *;
```

---- Vendor ----

```
-- Insert Values --

INSERT INTO Vendor (Vendor_ID, Vendor_Name, Vendor_Address,
Vendor_Phone_Number, Vendor_Since)
VALUES ('770001', 'Ticonderoga', '15 Pencil Lane, Los Angeles, CA 90034',
'3108956644', '01-01-2018');

INSERT INTO Vendor (Vendor_ID, Vendor_Name, Vendor_Address,
Vendor_Phone_Number, Vendor_Since)
VALUES ('770222', 'Nestle', '4 Candy Drive, Boston, MA 02134', '8578953377',
'01-01-2018');

INSERT INTO Vendor (Vendor_ID, Vendor_Name, Vendor_Address,
Vendor_Phone_Number, Vendor_Since)
VALUES ('770002', 'Campbells', '100 Soup Avenue, Trenton, NJ 08534',
'6098951122', '01-01-2018');

INSERT INTO Vendor (Vendor_ID, Vendor_Name, Vendor_Address,
Vendor_Phone_Number, Vendor_Since)
VALUES ('779999', 'Tobacco House', '77 Smoke Circle, Newark, NJ 08134',
'6098959988', '01-01-2018');

INSERT INTO Vendor (Vendor_ID, Vendor_Name, Vendor_Address,
Vendor_Phone_Number, Vendor_Since)
VALUES ('770003', 'Hersheys', '22 Chocolate Avenue, Hershey Park, PA 07033',
'7178950055', '01-01-2018');

-- Update --

UPDATE Vendor
SET Vendor_Address = '500 Tasty Avenue, Trenton, NJ 08533'
WHERE Vendor_ID = 770002;

-- Delete --

-- (No need to ever delete anything from the Vendor table) --
```

---- Item ----

-- Insert Values --

```
INSERT INTO Item (Item_ID, Item_Name, Inventory_Count, Sold_Count,
Item_Description, Retail_Price, Vendor_Price, Vendor_ID)
VALUES ('335789', 'Ticonderoga Pencils', '20', '10', '10-Pack Ticonderoga #2
Pencils with Eraser', 2.99, 1.49, '770001');

INSERT INTO Item (Item_ID, Item_Name, Inventory_Count, Sold_Count,
Item_Description, Retail_Price, Vendor_Price, Vendor_ID)
VALUES ('114169', 'Crunch Bar', '30', '15', '2.5 oz Nestle Milk Chocolate Crunch
Bar', 1.49, 0.75, '770222');

INSERT INTO Item (Item_ID, Item_Name, Inventory_Count, Sold_Count,
Item_Description, Retail_Price, Vendor_Price, Vendor_ID)
VALUES ('116185', 'Chicken Noodle Soup', '40', '20', '12 oz Campbells Chicken
Noodle Soup', 2.49, 1.45, '770002');

INSERT INTO Item (Item_ID, Item_Name, Inventory_Count, Sold_Count,
Item_Description, Retail_Price, Vendor_Price, Vendor_ID)
VALUES ('116198', 'Vegetable Medley Soup', '50', '25', '12 oz Campbells Vegetable
Medley Soup', 2.49, 1.45, '770002');

INSERT INTO Item (Item_ID, Item_Name, Inventory_Count, Sold_Count,
Item_Description, Retail_Price, Vendor_Price, Vendor_ID)
VALUES ('335744', 'Bic Ball Point Pen', '22', '11', '2-Pack Black Bic Ball Point Pen
with Clip', 2.99, 1.49, '770001');

INSERT INTO Item (Item_ID, Item_Name, Inventory_Count, Sold_Count,
Item_Description, Retail_Price, Vendor_Price, Vendor_ID)
VALUES ('995599', 'Fresh Cigar', '5', '0', '1-Pack Fresh Tobacco Cigar', 5.99, 2.79,
'779999');

INSERT INTO Item (Item_ID, Item_Name, Inventory_Count, Sold_Count,
Item_Description, Retail_Price, Vendor_Price, Vendor_ID)
```

```
VALUES ('335755', 'Bic White Out Marker', '0', '8', 'One Bic White Out Marker',
2.99, 1.49, '770001');

-- Update --

UPDATE Item
SET Retail_Price = 1.99, Vendor_Price = 0.99
WHERE Item_ID = 335744;

-- Delete --

-- (No need to ever delete anything from the Vendor table) --


---- Customer ----

-- Insert Values --

INSERT INTO Customer (Customer_ID, Customer_Name, Customer_Address,
Customer_Phone_Number,
                                Customer_Birthday, Customer_Since,
Transaction_Count, Customer_Total_Spend)
VALUES ('3311001', 'Harold Jackson', '4185 Moffett Lane, Dover, DE 19904',
'8568880055',
            '02-01-1982', '01-02-2018', '136', 1031.96);

INSERT INTO Customer (Customer_ID, Customer_Name, Customer_Address,
Customer_Phone_Number,
                                Customer_Birthday, Customer_Since,
Transaction_Count, Customer_Total_Spend)
VALUES ('3311002', 'Donny Flapper', '4198 Moffett Lane, Dover, DE 19904',
'8568883300',
            '04-11-1973', '01-03-2018', '78', 602.33);

INSERT INTO Customer (Customer_ID, Customer_Name, Customer_Address,
Customer_Phone_Number,
```

```
                                Customer_Birthday, Customer_Since,
Transaction_Count, Customer_Total_Spend)
VALUES ('3311003', 'Melissa Horton', '222 Deerborn Drive, Dover, DE 19904',
'8563212200',
            '06-18-1992', '01-03-2018', '99', 1604.34);

INSERT INTO Customer (Customer_ID, Customer_Name, Customer_Address,
Customer_Phone_Number,
                                Customer_Birthday, Customer_Since,
Transaction_Count, Customer_Total_Spend)
VALUES ('3311004', 'Sarah Johnson', '481 Blunder Blvd, Dover, DE 19906',
'8569876254',
            '08-22-1979', '01-05-2018', '68', 536.44);

INSERT INTO Customer (Customer_ID, Customer_Name, Customer_Address,
Customer_Phone_Number,
                                Customer_Birthday, Customer_Since,
Transaction_Count, Customer_Total_Spend)
VALUES ('3311005', 'Christina Maxwell', '6980 Power Circle, Dover, DE 19907',
'8569874477',
            '09-12-1998', '01-06-2018', '155', 1596.44);

INSERT INTO Customer (Customer_ID, Customer_Name, Customer_Address,
Customer_Phone_Number,
                                Customer_Birthday, Customer_Since,
Transaction_Count, Customer_Total_Spend)
VALUES ('0000001', 'No Membership', 'No Address', '0000000000',
            '01-01-2018', '01-01-2018', '15348', 150599.99);

-- Update --

UPDATE Customer
SET Customer_Address = '555 Baker Avenue, Dover, DE 19908'
WHERE Customer_ID = 3311005;

-- Delete --
```

```
-- (No need to ever delete anything from the Customer table,
-- but this specific customer requested to have all of their info removed from our
database) --

DELETE FROM Customer
WHERE Customer_ID = 3311004
RETURNING *;


---- Transactions ----

-- Insert Values --

INSERT INTO Transactions (Transaction_ID, Store_Name, Transaction_Date,
Transaction_Time,
                                    Employee_ID, Customer_ID, Item_Count, Item_ID,
Total_Amount)
VALUES ('999000100', 'The Mini-Mart', '01-02-2018', '12:34PM', '00011',
'0000001', '1', '335789', 2.99);

INSERT INTO Transactions (Transaction_ID, Store_Name, Transaction_Date,
Transaction_Time,
                                    Employee_ID, Customer_ID, Item_Count, Item_ID,
Total_Amount)
VALUES ('999000138', 'The Mini-Mart', '01-04-2018', '2:34PM', '00011',
'0000001', '1', '114169', 1.49);

INSERT INTO Transactions (Transaction_ID, Store_Name, Transaction_Date,
Transaction_Time,
                                    Employee_ID, Customer_ID, Item_Count, Item_ID,
Total_Amount)
VALUES ('999000148', 'The Mini-Mart', '01-05-2018', '11:34AM', '00017',
'0000001', '1', '116185', 2.49);

INSERT INTO Transactions (Transaction_ID, Store_Name, Transaction_Date,
Transaction_Time,
```

```
                                        Employee_ID, Customer_ID, Item_Count, Item_ID,
Total_Amount)
VALUES ('999000172', 'The Mini-Mart', '01-06-2018', '4:39PM', '00017',
'0000001', '2', '116198', 4.98);

INSERT INTO Transactions (Transaction_ID, Store_Name, Transaction_Date,
Transaction_Time,
                                        Employee_ID, Customer_ID, Item_Count, Item_ID,
Total_Amount)
VALUES ('999000199', 'The Mini-Mart', '01-07-2018', '3:15PM', '00024',
'0000001', '1', '335744', 2.99);

-- Update --

UPDATE Transactions
SET Transaction_Time = '5:57PM'
WHERE Transaction_ID = 999000199;

-- Delete --

-- (No need to ever delete anything from the Transactions table. Refunds should be
an alternative solution) --


---- Feedback ----

-- Insert Values --

INSERT INTO Feedback (Survey_ID, Store_Name, Survey_Type,
Customer_Comment, Employee_ID,
                                            Customer_ID, Transaction_ID, Survey_Date,
Status)
VALUES ('440001', 'The Mini-Mart', 'Positive', 'Friendly Staff!', '00011',
            '3311001', '999000100', '01-05-2018', 'Acknowledged');

INSERT INTO Feedback (Survey_ID, Store_Name, Survey_Type,
Customer_Comment, Employee_ID,
```

```
                                        Customer_ID, Transaction_ID, Survey_Date,
Status)
VALUES ('440002', 'The Mini-Mart', 'Negative', 'Wait time was too long...', '00011',
            '3311002', '999000138', '01-09-2018', 'Acknowledged');

INSERT INTO Feedback (Survey_ID, Store_Name, Survey_Type,
Customer_Comment, Employee_ID,
                                        Customer_ID, Transaction_ID, Survey_Date,
Status)
VALUES ('440003', 'The Mini-Mart', 'Negative', 'Rude attitude!', '00017',
            '3311003', '999000148', '01-11-2018', 'Pending');

INSERT INTO Feedback (Survey_ID, Store_Name, Survey_Type,
Customer_Comment, Employee_ID,
                                        Customer_ID, Transaction_ID, Survey_Date,
Status)
VALUES ('440004', 'The Mini-Mart', 'Positive', 'Very helpful!', '00017',
            '3311001', '999000172', '01-13-2018', 'Acknowledged');

INSERT INTO Feedback (Survey_ID, Store_Name, Survey_Type,
Customer_Comment, Employee_ID,
                                        Customer_ID, Transaction_ID, Survey_Date,
Status)
VALUES ('4400015', 'The Mini-Mart', 'Positive', 'Nice and helpful!', '00024',
            '3311005', '999000199', '01-16-2018', 'Acknowledged');

-- Update --

UPDATE Feedback
SET Status = 'Training'
WHERE Survey_ID = 440003;

-- Delete --

-- (No need to ever delete anything from the Feedback table) --
```

```
-- Trigger --

-- When an employee is hired by the business and their information is inserted into the database, --
-- a job_id is automatically generated for them because of this trigger --
-- The job table tracks all financial information related to the employees whereas the employee table tracks all personal employee information --

CREATE OR REPLACE FUNCTION add_employee_trigger_func()
	RETURNS TRIGGER
	LANGUAGE plpgsql
AS
$$
BEGIN

   INSERT INTO Job (job_id, job_title, employee_id, rate, store_name)
   VALUES(NEW.job_id, NULL, NEW.employee_id, NULL, NULL);

RETURN NEW;
END;
$$;

CREATE TRIGGER add_employee_trigger

	AFTER INSERT
	ON Employee
	FOR EACH ROW
	EXECUTE PROCEDURE add_employee_trigger_func();


-- Using the Trigger --

INSERT INTO Employee (Employee_ID, Job_ID, Employee_Name,
Employee_Address, Employee_Phone_Number,
                                Employee_Birthday, Hire_Date, Vacation_Hours,
Sick_Hours)
```

```
VALUES (68, 141, 'Jim Biglin', '24 Waverly Place, Newark, DE 19999',
'8567903344',
        '03/22/1989', '10/03/2021', 6, 3);

--

UPDATE Job
SET Job_title = 'Cashier', Rate = 11.50, Store_name = 'The Mini-Mart'
WHERE Job_ID = 141;

--



-- Stored Procedure --

-- This stored procedure inserts a new customer into the customer table --
-- The customer will provide us with four pieces of their information: Name,
Address, Phone Number, and Date of Birth --
-- Since customer_id is a serial datatype, a new customer_id will be generated to
maintain referential integrity --

DROP PROCEDURE IF EXISTS sp_add_customer;
CREATE PROCEDURE sp_add_customer (x VARCHAR(50), y VARCHAR(60), z
VARCHAR(10), n DATE)
LANGUAGE plpgsql
AS $$
BEGIN
INSERT INTO Customer (customer_name, customer_address,
customer_phone_number,
                                        customer_birthday, customer_since,
transaction_count, customer_total_spend)
  VALUES (x, y, z, n, current_date, 0, 0);
END
$$;

-- Calling the stored procedure --
```

```
CALL sp_add_customer ('Hakeem Amir', '111 James Street, Los Angeles, CA
90034', '6098553459', '10-23-1987');

--



-- * Import  transactions.csv and items.csv files now * --



-- Function 1 --

-- This function will find the employee who had the most sales on any given day --
-- A date is to be passed in as a parameter to this function --

CREATE OR REPLACE FUNCTION most_sales_day(x date)
	RETURNS TABLE (Total_Sales numeric, Employee_Name varchar)
	LANGUAGE plpgsql
AS
$$
BEGIN
	RETURN QUERY
	SELECT SUM(Total_Amount) AS Total_Sales, Employee.Employee_Name
	FROM Transactions
	INNER JOIN Employee ON Transactions.Employee_ID =
Employee.Employee_ID
	WHERE Transaction_Date = x
	GROUP BY Employee.Employee_Name
	ORDER BY 1 DESC
	LIMIT 1;
END
$$;

-- Calling Function 1 --
```

```
select most_sales_day('03-30-2020');

--



-- Function 2 --

-- This function will return the items that are below the inventory threshold the
user passes in --
-- An integer is passed in the function which sets the limit for the inventory_count
and the function will return which items fall below that value --
-- The goal is to be able to inform the employee what items they might be selling
out of based on current quantity levels --
-- This function can also help organize the stock room as it can tell you what items
we have most of as well --

CREATE OR REPLACE FUNCTION min_inv_count(y int)
	RETURNS TABLE (Item_ID int, Item_Name varchar, Inventory_Count
numeric)
	LANGUAGE plpgsql
AS
$$
BEGIN
	RETURN QUERY
	SELECT Item.Item_ID, Item.Item_Name, Item.Inventory_Count
	FROM Item
	WHERE Item.Inventory_Count <= y
	GROUP BY Item.Item_ID
	ORDER BY 3 DESC;
END
$$;

-- Calling Function 2 --

select min_inv_count(27);
```

```
--



-- View 1 --

-- A table of all the items that the business has an inventory_count of zero for --
-- This view will list those items along with the corresponding vendor to contact so we can order more of that item --
-- This view should be used by any employee who is authorized to order more inventory --

CREATE OR REPLACE VIEW out_of_stock_order AS
SELECT Item_ID, Item_Name, Item.Vendor_ID, Vendor_Name,
Vendor_Phone_Number, Vendor_Address, Inventory_Count
FROM Item INNER JOIN Vendor ON Item.Vendor_ID = Vendor.Vendor_ID
WHERE Inventory_Count = 0
ORDER BY 3;

-- Querying View 1 --

select * from out_of_stock_order;

--



-- View 2 --

-- A table of all the next upcoming birthdays for all the employees of the business --
-- This view can be used by any manager to offer an employee an opportunity to use some vacation time instead of working on their birthday --
-- This view can also help remind managers to celebrate or wish their employees birthday whether they see them --

CREATE OR REPLACE VIEW Next_Employee_Birthdays AS
```

```
SELECT Employee_Name, Employee_Birthday, Vacation_Hours
FROM Employee
ORDER BY Employee_Birthday DESC;

-- Querying View 2 --

select * from next_employee_birthdays;

--



-- View 3 --

-- A view of all the transactional records performed by the employees --
-- This view will assist in running queries and locating specific transactions
historically --

CREATE OR REPLACE VIEW employee_sales AS
SELECT Transactions.store_name, transaction_date, transaction_time,
employee.employee_id,
customer_id, item_count, item_id, total_amount, job_id, employee_name
FROM Transactions INNER JOIN Employee ON Transactions.employee_id =
Employee.employee_id
INNER JOIN Store ON Transactions.store_name = Store.store_name;

-- Querying View 3 --

select * from employee_sales;

--
```

B. Queries

-- Query 1 --

-- This query provides information regarding every employee of the business. --
-- Along with their name and employee ID, we obtain their job title, rate, vacation hours and more. --
-- The results are ordered from lowest to highest rates. --
-- This gives us an idea on which employees have the most available paid time off as well. --

```
SELECT e.employee_name, e.employee_id, j.job_title, j.job_id,
                j.rate, e.hire_date, e.vacation_hours, e.sick_hours
FROM Employee AS e
INNER JOIN Job AS j ON e.employee_id = j.employee_id
ORDER BY rate;
```

--

-- Query 2 --

-- This query provides specfic transactional data grouped by the transaction_id and --
-- orderded by the date and time the transaction took place. --
-- This table also gives us an idea on how many items were sold in each transaction along with the total cost. --

```
SELECT transaction_id, COUNT(item_id) AS item_amount, total_amount,
transaction_date, transaction_time
FROM Transactions
GROUP BY transaction_id
ORDER BY transaction_date, transaction_time;
```

--

-- Query 3 --

-- This query allows us to see the total amount of sales by each employee who is not a Manager or an Assistant Manager. --
-- We also get the number of transactions and the average amount per transaction through a calculation. --
-- This query will give us an idea which non-management employee completes the most transactions too. --

```
SELECT DISTINCT(Employee_Name), SUM(Total_amount) AS Total_Amount,
                        COUNT(Transaction_ID) AS Transaction_Count,
                        (SUM(Total_amount)/COUNT(Transaction_ID)) AS
AVG_Amount
FROM Employee FULL OUTER JOIN Transactions ON Employee.Employee_ID =
Transactions.Employee_ID
FULL OUTER JOIN Job ON Employee.Employee_ID = Job.Employee_ID
WHERE Job.Job_ID != '5' AND Job.Job_ID != '7'
GROUP BY 1
ORDER BY 2 DESC;
```

--

-- Query 4 --

-- This query utilizes the union operator to combine two different queries into one table. --
-- Here we find every transaction that is associated with a survey left by a customer. --
-- We can also learn what type of feedback was left for the business along with the employee name who performed the transaction. --

```
SELECT Employee.employee_id, transaction_id, employee_name
```

```
FROM Employee INNER JOIN Feedback on Employee.employee_id =
Feedback.employee_id

UNION ALL

SELECT survey_id, transaction_id, survey_type
FROM Feedback
WHERE Feedback.transaction_id != 0
ORDER BY 2;

--



-- Query 5 --

-- This query will tell us how many items were sold by each employee on each day.
--
-- The table is organized by the employee who has sold the most items. --
-- Since this is a new system for the business, the previous method (MS Excel) did
not track items sold --

select ee.employee_name, sum(item_count) as Items_Sold, transaction_date
from transactions as tt inner join employee as ee on tt.employee_id =
ee.employee_id
group by transaction_date, employee_name
order by 2 desc;

--



-- Query 6 --

-- The previous system did not provide the customer an opportunity to give
feedback, but the new system does --
-- This query has a nested subquery inside of it. --
```

```
-- This query counts the amount of surveys each customer has submitted ordered
by the amount of surveys left. --

SELECT customer_id, customer_name,
        (SELECT count (*)
         FROM Feedback WHERE feedback.customer_id = customer.customer_id)
         AS survey_amount
FROM Customer
GROUP BY customer_id
ORDER BY survey_amount DESC;

--



-- Query 7 --

-- This query uses the EXCEPT operator. --
-- This will find all the customers of the store who have not left feedback. --
-- The first select statment finds all the unique customer_ids and
customer_names, --
-- while the second select statment finds all the customers who have submitted
feedback. --
-- However, the EXCEPT operator excludes everyone in the second select
statement. --

SELECT DISTINCT customer_id, customer_name, customer_since,
transaction_count
FROM Customer
EXCEPT
SELECT DISTINCT Customer.customer_id, customer.customer_name,
customer_since, transaction_count
FROM Feedback INNER JOIN Customer ON Customer.customer_id =
Feedback.customer_id;

--
```

-- Query 8 --

-- This query uses the LIKE operator to identify all the customers located in the city of Dover. --
-- This query also uses the DISTINCT statement to locate unqiue customers only. --

```
SELECT DISTINCT customer_name, customer.customer_id, customer_address,
customer_birthday,
                                customer_since, transaction_count, customer_total_spend
FROM customer FULL OUTER JOIN transactions ON customer.customer_id =
transactions.customer_id
WHERE customer_address LIKE 'Dover%' OR customer_address LIKE '% Dover%';
```

--

C. Web Application

```
## To run the Flask application, go to your Terminal and go to the directory where the .py
and .html files are located. From there, type in the following commands:

## $ export FLASK_APP=market.py
## $ flask run



## market.py ##

from flask import Flask, render_template, request, flash, redirect
from flask_sqlalchemy import SQLAlchemy
from psycopg2 import connect

app = Flask(__name__)

app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False # silence the deprecation warning
app.config['SQLALCHEMY_DATABASE_URI'] =
'postgressql://postgres:password@localhost/MiniMart'

db = SQLAlchemy(app)

app.run()

# http://127.0.0.1:5000/

@app.route('/')
@app.route('/home')
def home_page():
    conn = connect(host="localhost", database="MiniMart", user='postgres',
password='password')
    cur = conn.cursor()
    cur.execute('SELECT * FROM Store;')
    stores = cur.fetchall()
```

```
    cur.close()
    conn.close()
    return render_template('home.html', stores=stores)


@app.route('/customers')
def customer_page():
    conn = connect(host="localhost", database="MiniMart", user='postgres',
password='password')
    cur = conn.cursor()
    cur.execute('SELECT * FROM customer;')
    customers = cur.fetchall()
    cur.close()
    conn.close()
    return render_template('customers.html', customers=customers)


@app.route("/customers/update/<customer_id>")
def update_customer(customer_id):
    return render_template('update.html', customer_id=customer_id)


@app.route("/customers/update/phh/<customer_id>", methods = ["POST"])
def update_phh(customer_id):
    customer_phone_number = request.form["phh"]
    conn = connect(host="localhost", database="MiniMart", user='postgres',
password='password')
    cur = conn.cursor()
    cur.execute(f'UPDATE customer SET customer_phone_number = {customer_phone_number} WHERE
customer_id = {customer_id};')
    conn.commit()
    cur.close()
    conn.close()
    return redirect("/customers")
```

```
<!-- base.html -->

<!doctype html>
<html lang="en">
  <head>
     <!-- Meta tags -->
     <meta charset="utf-8">
     <meta name="viewport" content="width=device-width, initial-scale=1,
shrink-to-fit=no">
     <!-- Bootstrap CSS -->
     <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@4.5.3/dist/css/bootstrap.min.css"
integrity="sha384-TX8t27EcRE3e/ihU7zmQxVncDAy5uIKz4rEkgIXeMed4M0jlfIDPvg6uqKI2xXr2"
crossorigin="anonymous">
     <title>
         {% block title %}

         {% endblock %}
     </title>
  </head>
  <body>
   <nav class="navbar navbar-expand-md navbar-dark bg-dark">
     <a class="navbar-brand" href="#">The Mini-Mart</a>
     <button class="navbar-toggler" type="button" data-toggle="collapse"
data-target="#navbarNav">
       <span class="navbar-toggler-icon"></span>
     </button>
     <div class="collapse navbar-collapse" id="navbarNav">
       <ul class="navbar-nav mr-auto">
           <li class="nav-item active">
               <a class="nav-link" href="{{ url_for('home_page') }}">Home <span
class="sr-only">(current)</span></a>
           </li>
           <li class="nav-item">
               <a class="nav-link" href="{{ url_for('customer_page') }}">Customer</a>
           </li>
       </ul>
```

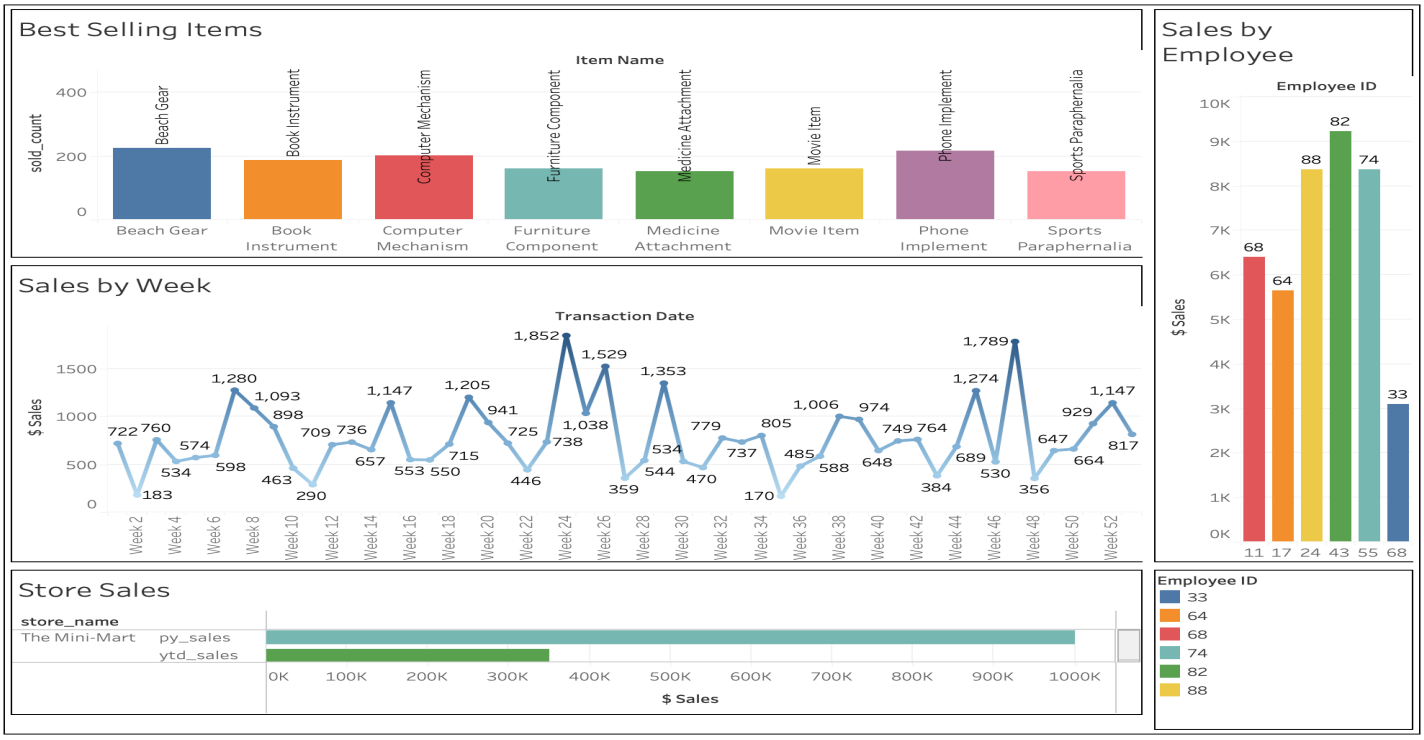
```
        <ul class="navbar-nav">
            <li class="nav-item">
                <a class="nav-link" href="#">Login</a>
            </li>
            <li class="nav-item">
                <a class="nav-link" href="#">Register</a>
            </li>
        </ul>
      </div>
    </nav>
      {% block content %}

      {% endblock %}

      <!-- Optional JavaScript -->
      <!-- jQuery first, then Popper.js, then Bootstrap JS -->
      <script src='https://kit.fontawesome.com/a076d05399.js'></script>
      <script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"
integrity="sha384-DfXdz2htPH0lsSSs5nCTpuj/zy4C+OGpamoFVy38MVBnE+IbbVYUew+OrCXaRkfj"
crossorigin="anonymous"></script>
      <script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.1/dist/umd/popper.min.js"
integrity="sha384-9/reFTGAW83EW2RDu2S0VKaIzap3H66lZH81PoYlFhbGU+6BZp6G7niu735Sk7lN"
crossorigin="anonymous"></script>
      <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"
integrity="sha384-B4gt1jrGC7Jh4AgTPSdUtOBvfO8shuf57BaghqFfPlYxofvL8/KUEfYiJOMMV+rV"
crossorigin="anonymous"></script>
  </body>
  <style>
      body {
      background-color: #212121;
      color: white
      }
  </style>
</html>
```

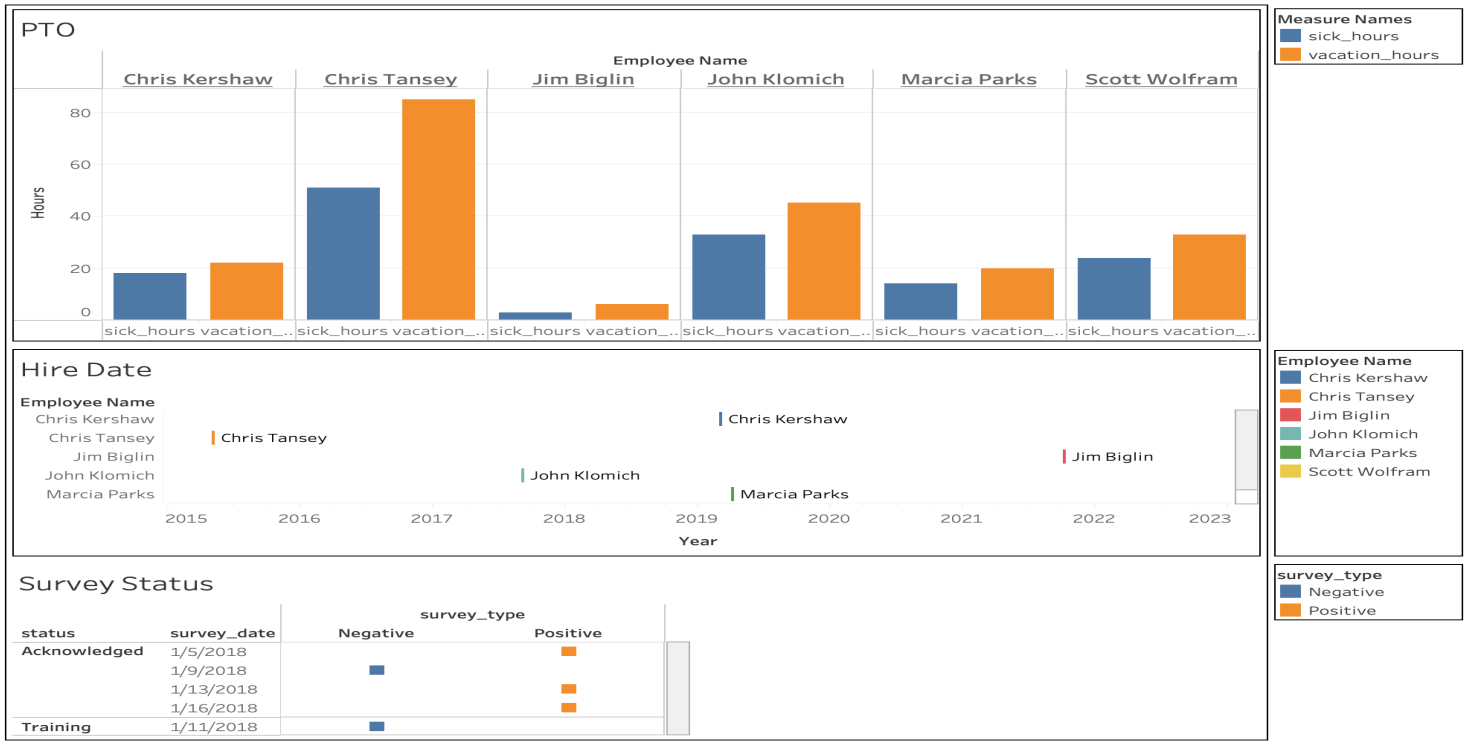
```
<!-- home.html -->

{% extends 'base.html' %}
{% block title %}
   Home Page
{% endblock %}

{% block content %}
   <h2>This is the Home Page for The Mini-Mart!</h2>
   <h3>Store Information:</h3>
   {% for store in stores %}
           <tr>
               <!-- Your Columns HERE -->
               <th scope="col">Store Name, Address, Phone Number</th>
           </tr>
       <p>{{ store[0] }}</p>
       <p>{{ store[1] }}</p>
       <p>{{ store[2] }}</p>
   {% endfor %}
{% endblock %}
```

```
<!-- customers.html -->

{% extends 'base.html' %}
{% block title %}
  Customer Page
{% endblock %}

{% block content %}
   <table class="table table-hover table-dark">
       <thead>
           <tr>
               <!-- Your Columns HERE -->
               <th scope="col">Customer Name</th>
               <th scope="col">Phone Number</th>
               <th scope="col">Update</th>
           </tr>
       </thead>
       <tbody>
           <!-- Your rows inside the table HERE: -->
           {% for customer in customers %}
               <tr>
                   <form action = '{{ "/customers/update/" ~ customer[0] }}' >
                       <td>{{ customer[1] }}</td>
                       <td>{{ customer[3] }}</td>
                       <td>
                           <button class="btn btn-outline btn-success">Update Now</button>
                       </td>
                   </form>
               </tr>
           {% endfor %}
       </tbody>
   </table>
{% endblock %}
```

```
<!-- update.html -->

{% extends 'base.html' %}
{% block title %}
   Update Page
{% endblock %}

{% block content %}
   <h2>This will update customer's phone number</h2>
   <form action = '{{ "/customers/update/phh/" ~ customer_id }}' method = "POST" >
       <input type = "tel" name = "phh" placeholder = "Enter Phone Number Here"
pattern="[1-9]{1}[0-9]{9}" required />
       <button class="btn btn-outline btn-success">Update Now</button>
   </form>
{% endblock %}




## To run the Flask application, go to your Terminal and go to the directory where the .py
and .html files are located. From there, type in the following commands:

## $ export FLASK_APP=market.py
## $ flask run
```

D. Tableau Dashboards





Presentation

- The recorded Zoom presentation can be found in the Brightspace Media Gallery.

- https://docs.google.com/document/d/1yIoffARQ1Esh4sXXfMVRuTfQj_kAWbb2OgzDkEnAL_4/edit?usp=sharing

References

- Silberschatz, A., Korth, H. F., & Sudarshan, S. (2020). *Database System Concepts* (7th ed.). McGraw-Hill Education.
- Holmes, N. (2020, August 31). *Why Excel is Not a Product Database*. Widen. Retrieved March 12, 2022, from https://www.widen.com/blog/excel-is-not-product-database
- JimShapedCoding, (2021, March 10). *Flask Course - Python Web Application Development*. YouTube. Retrieved April 2, 2022, from https://www.youtube.com/watch?v=Qr4QMBUPxWo
- *Data Generator*. Cobbl. (n.d.). Retrieved April 1, 2022, from https://cobbl.io/