2/16/2020

FINAL PROJECT REPORT

Submission By: Stuti Sanghavi

Student ID: 00001587699

Course: DBMS – SQL

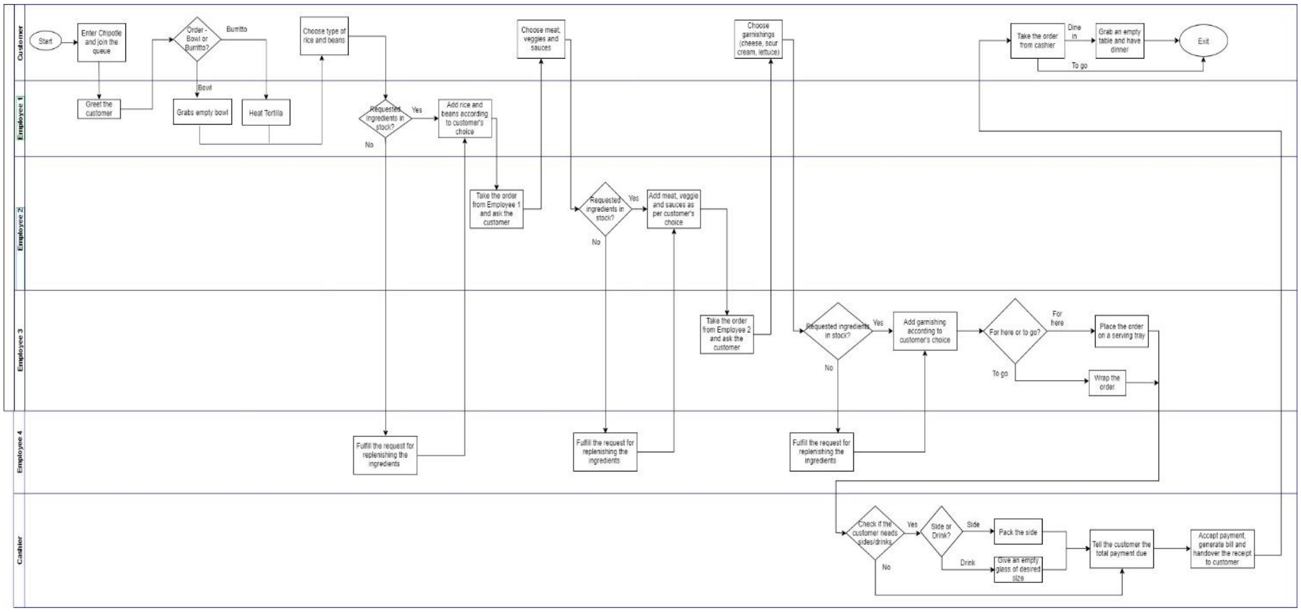
Table of Contents

I. Business Scenario Description

1. Restaurant Scenario:

Matt, the customer walks into the Chipotle outlet located in Fremont near Mowry Avenue at 7:25 PM. Matt waits in the queue for his turn. There are 4 employees working at the front counter for quick order processing. After waiting for about 10 minutes, employee 1 at the store greets Matt and asks what he would like to order from the menu. Matt orders a burrito bowl. Employee one grabbed an empty bowl without the lid and then asked Matt for his choice of rice and beans. Matt replied that he wanted brown rice with black beans. Employee 1 filled the bowl accordingly and passed it on to Employee 2. Employee 2 asked for the choice of meat, veggies and sauces. Matt replied no meat, just tomatoes and corn along with hot sauce. Employee 2 adds the veggies and sauces as per Matt's liking and passes the order to the third employee. Employee 3 then asks Matt for the final garnishing of sour cream, cheese, guacamole and lettuce, and adds them accordingly. Employee 3 asked Matt whether the order was for here or to go. Matt responded for here. Employee 3 then places the order on a serving tray and passes it on to the cashier. The cashier asks Matt if any sides or drink is needed. Matt politely refuses. Post which the cashier mentions that the order total is $8.69 with tax. Matt uses his credit card to make the payment and picks up his tray and chooses a table to dine. Post dinner, he disposes the trash, places the empty tray near the bin and then leaves the restaurant at 8:05 PM.

2. Swimlane Diagram (pls check separate attachment on camino for better visibility):

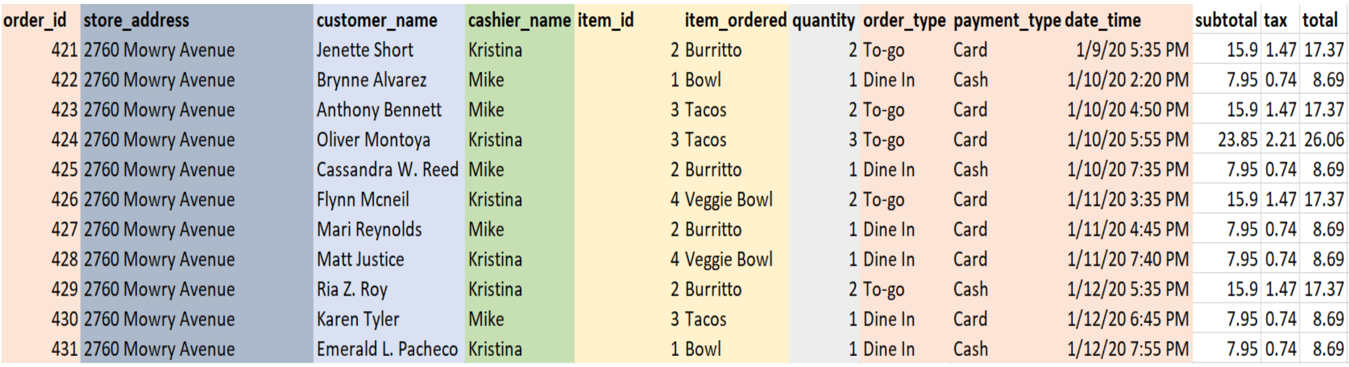


Customer
Employee 1
Employee 2
Employee 3
Employee 4
Cashier
Start
Enter Chipotle and join the queue
Greet the customer
Order - Bowl or Burritto?
Burritto
Bowl
Grabs empty bowl
Heat Tortilla
Choose type of rice and beans
Requested ingredients in stock?
Yes
No
Add rice and beans according to customer's choice
Take the order from Employee 1 and ask the customer
Choose meat, veggies and sauces
Requested ingredients in stock?
Yes
No
Add meat, veggie and sauces as per customer's choice
Take the order from Employee 2 and ask the customer
Choose garnishings (cheese, sour cream, lettuce)
Requested ingredients in stock?
Yes
No
Add garnishing according to customer's choice
For here or to go?
For here
To go
Place the order on a serving tray
Wrap the order
Fulfill the request for replenishing the ingredients
Fulfill the request for replenishing the ingredients
Fulfill the request for replenishing the ingredients
Check if the customer needs sides/drinks
Yes
No
Side or Drink?
Side
Drink
Pack the side
Give an empty glass of desired size
Tell the customer the total payment due
Accept payment, generate bill and handover the receipt to customer
Take the order from cashier
Dine in
To go
Grab an empty table and have dinner
Exit

II. Data Normalization

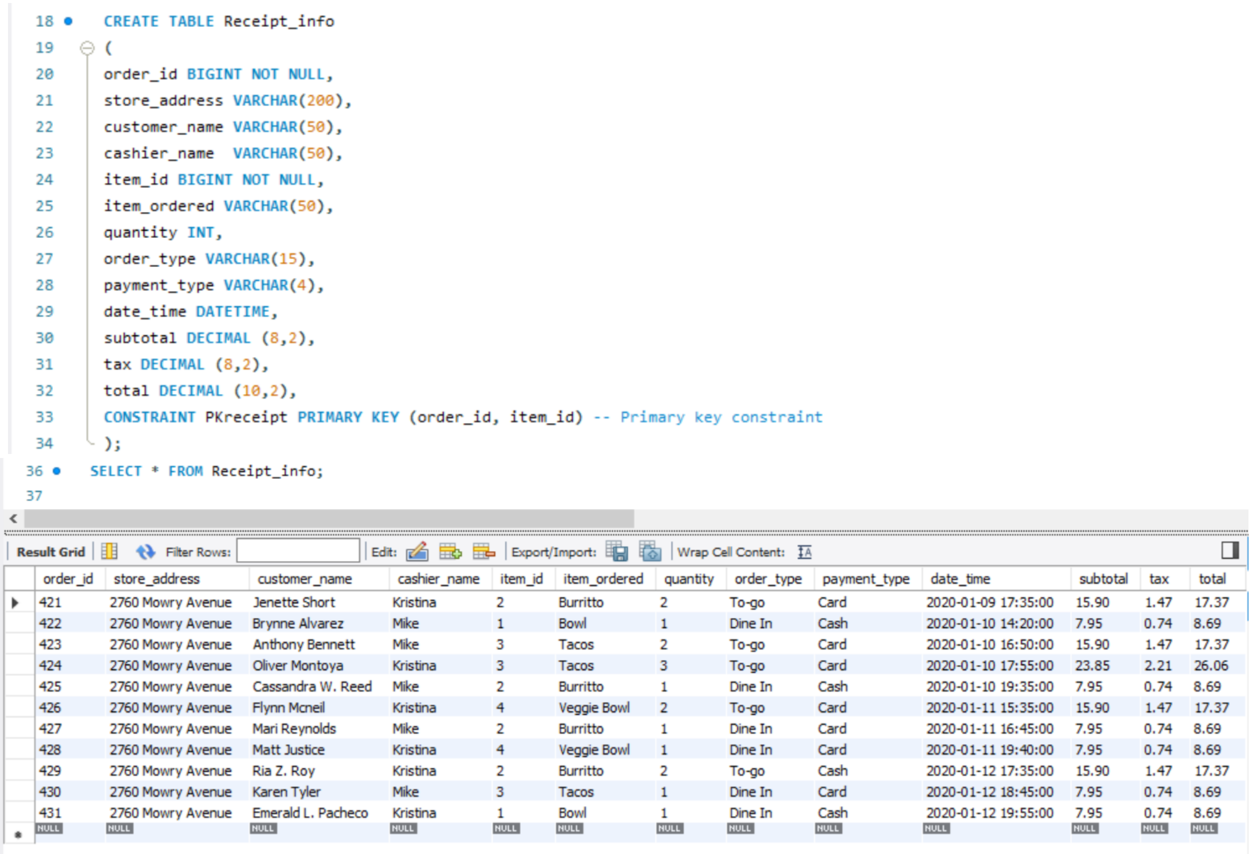
1. Denormalized Data (1st Normal Form)

First the data is collected in a denormalized form, where there are different entities in the same table, like the customer, employee, item, store etc. The denormalized data can be found as below:



order_id	store_address	customer_name	cashier_name	item_id	item_ordered	quantity	order_type	payment_type	date_time	subtotal	tax	total
421	2760 Mowry Avenue	Jenette Short	Kristina	2	Burritto	2	To-go	Card	1/9/20 5:35 PM	15.9	1.47	17.37
422	2760 Mowry Avenue	Brynne Alvarez	Mike	1	Bowl	1	Dine In	Cash	1/10/20 2:20 PM	7.95	0.74	8.69
423	2760 Mowry Avenue	Anthony Bennett	Mike	3	Tacos	2	To-go	Card	1/10/20 4:50 PM	15.9	1.47	17.37
424	2760 Mowry Avenue	Oliver Montoya	Kristina	3	Tacos	3	To-go	Card	1/10/20 5:55 PM	23.85	2.21	26.06
425	2760 Mowry Avenue	Cassandra W. Reed	Mike	2	Burritto	1	Dine In	Cash	1/10/20 7:35 PM	7.95	0.74	8.69
426	2760 Mowry Avenue	Flynn Mcneil	Kristina	4	Veggie Bowl	2	To-go	Card	1/11/20 3:35 PM	15.9	1.47	17.37
427	2760 Mowry Avenue	Mari Reynolds	Mike	2	Burritto	1	Dine In	Card	1/11/20 4:45 PM	7.95	0.74	8.69
428	2760 Mowry Avenue	Matt Justice	Kristina	4	Veggie Bowl	1	Dine In	Card	1/11/20 7:40 PM	7.95	0.74	8.69
429	2760 Mowry Avenue	Ria Z. Roy	Kristina	2	Burritto	2	To-go	Cash	1/12/20 5:35 PM	15.9	1.47	17.37
430	2760 Mowry Avenue	Karen Tyler	Mike	3	Tacos	1	Dine In	Card	1/12/20 6:45 PM	7.95	0.74	8.69
431	2760 Mowry Avenue	Emerald L. Pacheco	Kristina	1	Bowl	1	Dine In	Cash	1/12/20 7:55 PM	7.95	0.74	8.69

Each of the column above that have the same colors are the ones that should be grouped together for converting the data in normalized form. The table creation script for the denormalized table is



```sql
CREATE TABLE Receipt_info
(
order_id BIGINT NOT NULL,
store_address VARCHAR(200),
customer_name VARCHAR(50),
cashier_name  VARCHAR(50),
item_id BIGINT NOT NULL,
item_ordered VARCHAR(50),
quantity INT,
order_type VARCHAR(15),
payment_type VARCHAR(4),
date_time DATETIME,
subtotal DECIMAL (8,2),
tax DECIMAL (8,2),
total DECIMAL (10,2),
CONSTRAINT PKreceipt PRIMARY KEY (order_id, item_id) -- Primary key constraint
);

SELECT * FROM Receipt_info;
```

order_id	store_address	customer_name	cashier_name	item_id	item_ordered	quantity	order_type	payment_type	date_time	subtotal	tax	total
421	2760 Mowry Avenue	Jenette Short	Kristina	2	Burritto	2	To-go	Card	2020-01-09 17:35:00	15.90	1.47	17.37
422	2760 Mowry Avenue	Brynne Alvarez	Mike	1	Bowl	1	Dine In	Cash	2020-01-10 14:20:00	7.95	0.74	8.69
423	2760 Mowry Avenue	Anthony Bennett	Mike	3	Tacos	2	To-go	Card	2020-01-10 16:50:00	15.90	1.47	17.37
424	2760 Mowry Avenue	Oliver Montoya	Kristina	3	Tacos	3	To-go	Card	2020-01-10 17:55:00	23.85	2.21	26.06
425	2760 Mowry Avenue	Cassandra W. Reed	Mike	2	Burritto	1	Dine In	Cash	2020-01-10 19:35:00	7.95	0.74	8.69
426	2760 Mowry Avenue	Flynn Mcneil	Kristina	4	Veggie Bowl	2	To-go	Card	2020-01-11 15:35:00	15.90	1.47	17.37
427	2760 Mowry Avenue	Mari Reynolds	Mike	2	Burritto	1	Dine In	Card	2020-01-11 16:45:00	7.95	0.74	8.69
428	2760 Mowry Avenue	Matt Justice	Kristina	4	Veggie Bowl	1	Dine In	Card	2020-01-11 19:40:00	7.95	0.74	8.69
429	2760 Mowry Avenue	Ria Z. Roy	Kristina	2	Burritto	2	To-go	Cash	2020-01-12 17:35:00	15.90	1.47	17.37
430	2760 Mowry Avenue	Karen Tyler	Mike	3	Tacos	1	Dine In	Card	2020-01-12 18:45:00	7.95	0.74	8.69
431	2760 Mowry Avenue	Emerald L. Pacheco	Kristina	1	Bowl	1	Dine In	Cash	2020-01-12 19:55:00	7.95	0.74	8.69
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

2. Normalized Data (3rd Normal form)

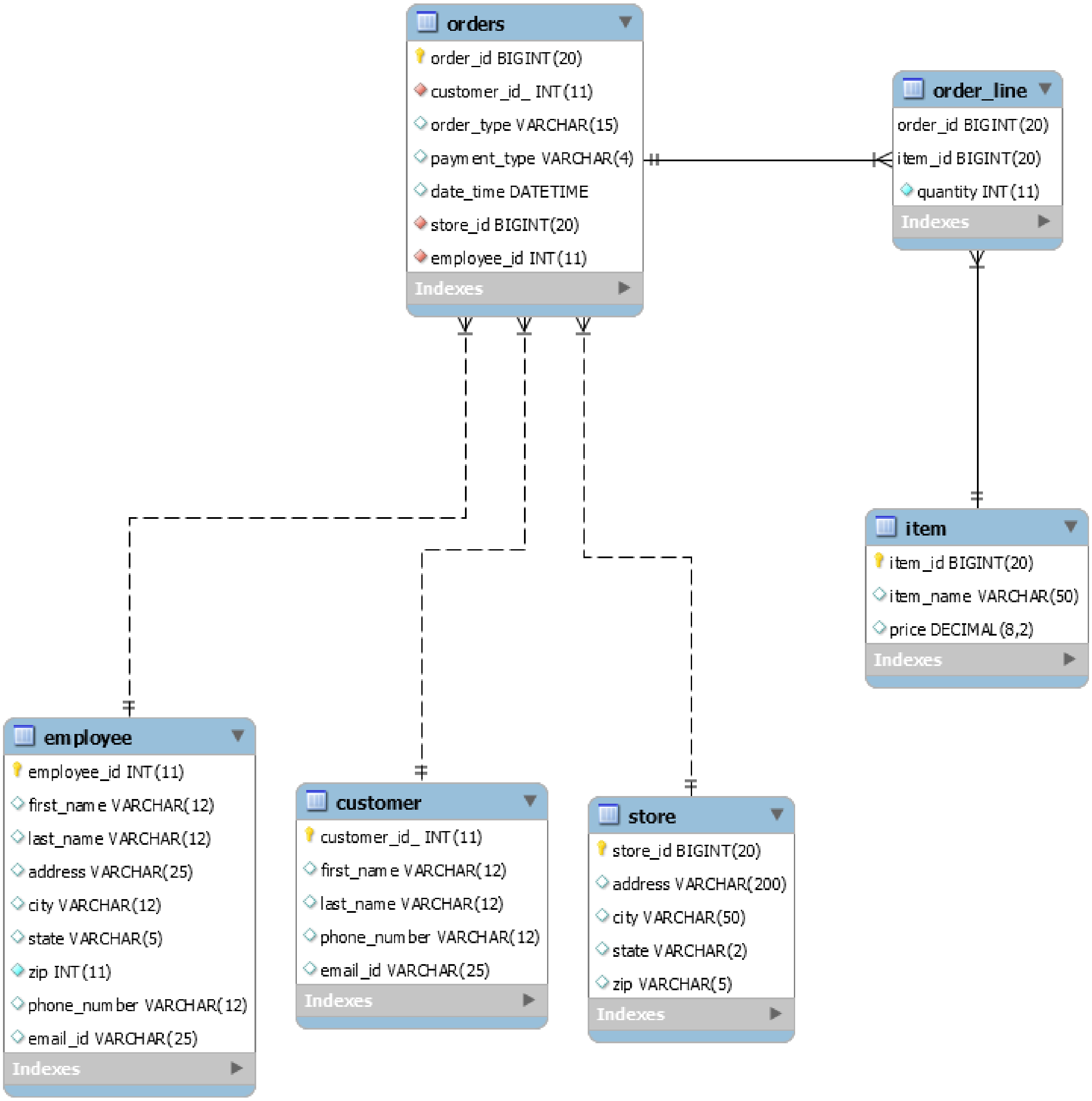
For converting the data into third normal form, we need to divide the denormalized table in a way that each of the entities are separated into different tables without having any kind of dependencies.

To do that, we divide the denormalized table into a total of 6 normalized tables as follows:

Sr.No	Table Name	Table Description
1	Store	➢ Contains information about the store, store_id, its location etc. ➢ It helps in knowing the various stores that we are looking at.
2	Customer	➢ Contains details about customers, including the name, number etc. ➢ It helps us in keeping a tab of customers who visited the store at least once
3	Item	➢ Details about all the items in the menu and their respective price
4	Employee	➢ Contains details about employee, including the name, number etc.
5	Orders	➢ Contains information about each order ➢ It tells us the customer ordered at which store, which employee helped the customer, the date, time and place of the order, etc.
6	Order Line	➢ Contains information about the item and quantity for each order

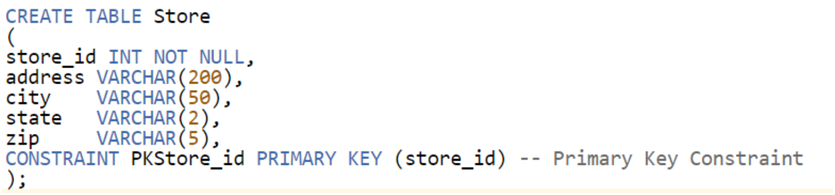
3. EER Diagram

Each of the table and its relationship with other tables can be better understood in the form of a data model as follows:



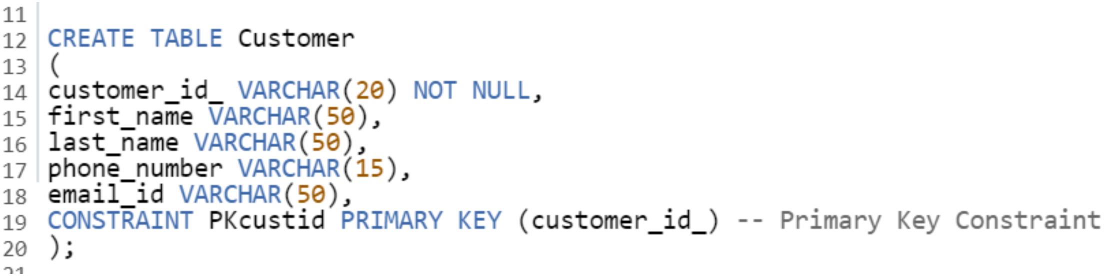
III. Table Creation Scripts for Normalized data

1. Store Table



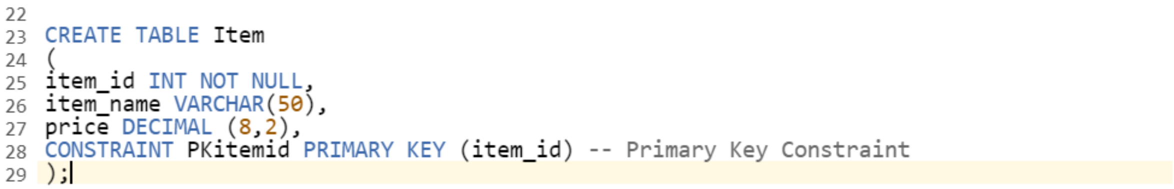
```sql
CREATE TABLE Store
(
store_id INT NOT NULL,
address VARCHAR(200),
city    VARCHAR(50),
state   VARCHAR(2),
zip     VARCHAR(5),
CONSTRAINT PKStore_id PRIMARY KEY (store_id) -- Primary Key Constraint
);
```

2. Customer Table



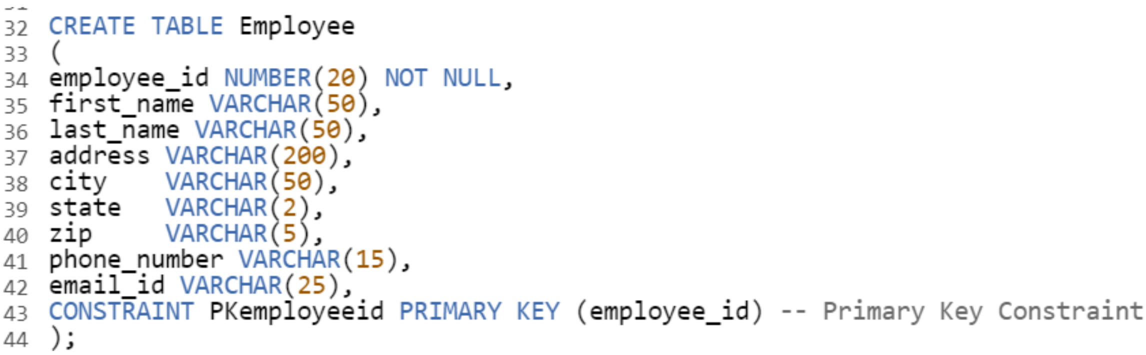
```sql
CREATE TABLE Customer
(
customer_id_ VARCHAR(20) NOT NULL,
first_name VARCHAR(50),
last_name VARCHAR(50),
phone_number VARCHAR(15),
email_id VARCHAR(50),
CONSTRAINT PKcustid PRIMARY KEY (customer_id_) -- Primary Key Constraint
);
```

3. Item Table



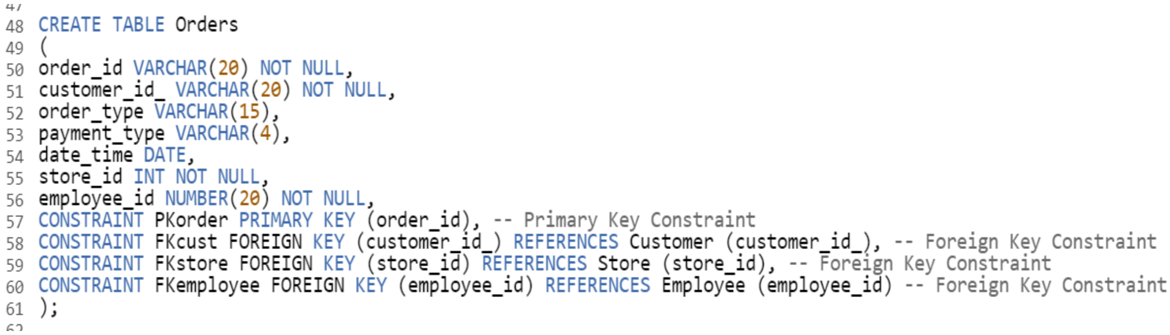
```sql
CREATE TABLE Item
(
item_id INT NOT NULL,
item_name VARCHAR(50),
price DECIMAL (8,2),
CONSTRAINT PKitemid PRIMARY KEY (item_id) -- Primary Key Constraint
);
```

4. Employee Table



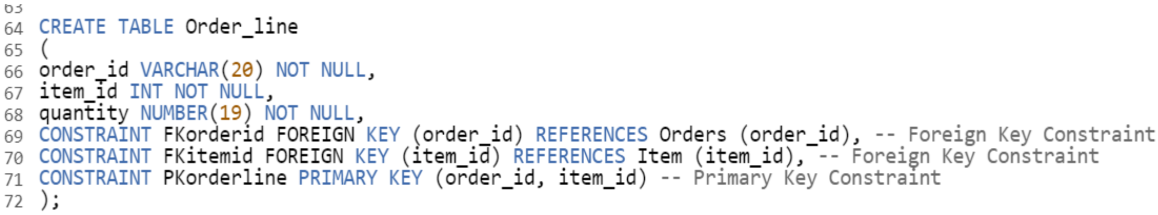
```sql
CREATE TABLE Employee
(
employee_id NUMBER(20) NOT NULL,
first_name VARCHAR(50),
last_name VARCHAR(50),
address VARCHAR(200),
city    VARCHAR(50),
state   VARCHAR(2),
zip     VARCHAR(5),
phone_number VARCHAR(15),
email_id VARCHAR(25),
CONSTRAINT PKemployeeid PRIMARY KEY (employee_id) -- Primary Key Constraint
);
```

5. Orders Table



```
CREATE TABLE Orders
(
order_id VARCHAR(20) NOT NULL,
customer_id_ VARCHAR(20) NOT NULL,
order_type VARCHAR(15),
payment_type VARCHAR(4),
date_time DATE,
store_id INT NOT NULL,
employee_id NUMBER(20) NOT NULL,
CONSTRAINT PKorder PRIMARY KEY (order_id), -- Primary Key Constraint
CONSTRAINT FKcust FOREIGN KEY (customer_id_) REFERENCES Customer (customer_id_), -- Foreign Key Constraint
CONSTRAINT FKstore FOREIGN KEY (store_id) REFERENCES Store (store_id), -- Foreign Key Constraint
CONSTRAINT FKemployee FOREIGN KEY (employee_id) REFERENCES Employee (employee_id) -- Foreign Key Constraint
);
```

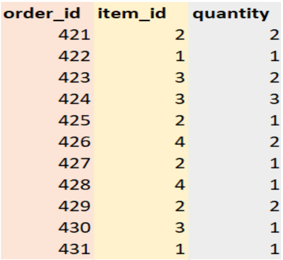
6. Order_Line Table



```
CREATE TABLE Order_line
(
order_id VARCHAR(20) NOT NULL,
item_id INT NOT NULL,
quantity NUMBER(19) NOT NULL,
CONSTRAINT FKorderid FOREIGN KEY (order_id) REFERENCES Orders (order_id), -- Foreign Key Constraint
CONSTRAINT FKitemid FOREIGN KEY (item_id) REFERENCES Item (item_id), -- Foreign Key Constraint
CONSTRAINT PKorderline PRIMARY KEY (order_id, item_id) -- Primary Key Constraint
);
```

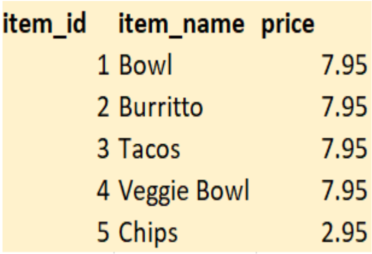
Normalized Data Tables

Order_Line Table



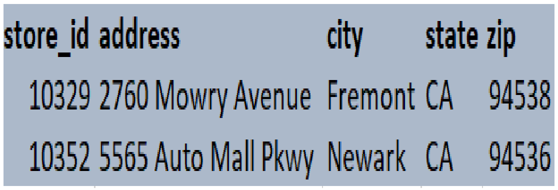
order_id	item_id	quantity
421	2	2
422	1	1
423	3	2
424	3	3
425	2	1
426	4	2
427	2	1
428	4	1
429	2	2
430	3	1
431	1	1

Item Table



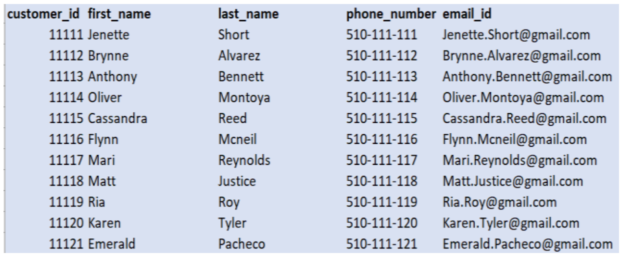
item_id	item_name	price
1	Bowl	7.95
2	Burritto	7.95
3	Tacos	7.95
4	Veggie Bowl	7.95
5	Chips	2.95

Store Table



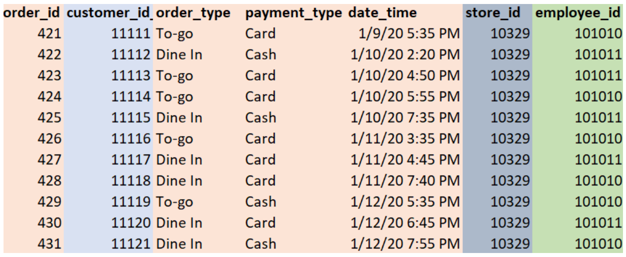
store_id	address	city	state	zip
10329	2760 Mowry Avenue	Fremont	CA	94538
10352	5565 Auto Mall Pkwy	Newark	CA	94536

Customer Table



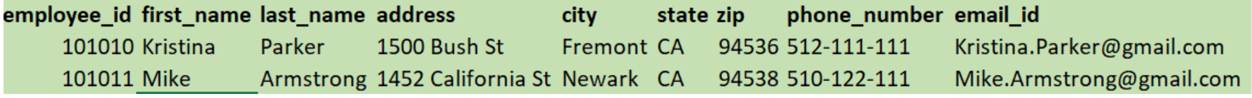
customer_id	first_name	last_name	phone_number	email_id
11111	Jenette	Short	510-111-111	Jenette.Short@gmail.com
11112	Brynne	Alvarez	510-111-112	Brynne.Alvarez@gmail.com
11113	Anthony	Bennett	510-111-113	Anthony.Bennett@gmail.com
11114	Oliver	Montoya	510-111-114	Oliver.Montoya@gmail.com
11115	Cassandra	Reed	510-111-115	Cassandra.Reed@gmail.com
11116	Flynn	Mcneil	510-111-116	Flynn.Mcneil@gmail.com
11117	Mari	Reynolds	510-111-117	Mari.Reynolds@gmail.com
11118	Matt	Justice	510-111-118	Matt.Justice@gmail.com
11119	Ria	Roy	510-111-119	Ria.Roy@gmail.com
11120	Karen	Tyler	510-111-120	Karen.Tyler@gmail.com
11121	Emerald	Pacheco	510-111-121	Emerald.Pacheco@gmail.com

Order Table



order_id	customer_id_	order_type	payment_type	date_time	store_id	employee_id
421	11111	To-go	Card	1/9/20 5:35 PM	10329	101010
422	11112	Dine In	Cash	1/10/20 2:20 PM	10329	101011
423	11113	To-go	Card	1/10/20 4:50 PM	10329	101011
424	11114	To-go	Card	1/10/20 5:55 PM	10329	101010
425	11115	Dine In	Cash	1/10/20 7:35 PM	10329	101011
426	11116	To-go	Card	1/11/20 3:35 PM	10329	101010
427	11117	Dine In	Card	1/11/20 4:45 PM	10329	101011
428	11118	Dine In	Card	1/11/20 7:40 PM	10329	101010
429	11119	To-go	Cash	1/12/20 5:35 PM	10329	101010
430	11120	Dine In	Card	1/12/20 6:45 PM	10329	101011
431	11121	Dine In	Cash	1/12/20 7:55 PM	10329	101010

Employee Table



employee_id	first_name	last_name	address	city	state	zip	phone_number	email_id
101010	Kristina	Parker	1500 Bush St	Fremont	CA	94536	512-111-111	Kristina.Parker@gmail.com
101011	Mike	Armstrong	1452 California St	Newark	CA	94538	510-122-111	Mike.Armstrong@gmail.com

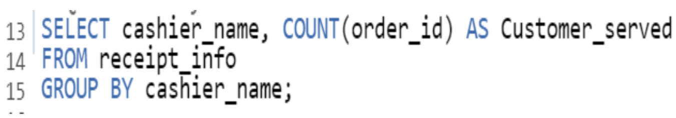
IV. SQL Queries to answer Business Questions

1. Use of Basic SQL

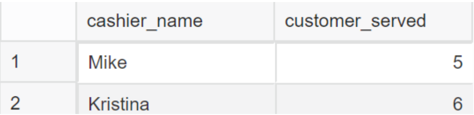
<u>The following business questions are answered without using Joins and Subqueries</u>

1. How many customers did each Kristina and Mike serve?

- *With this question we can answer, which employee(Kristina and Mike) served how many customers and also which employee served the most customers.*
- *For answering that, we take the cashier name and the count of the order_id (representing each order) and group by cashier name to get desired ouput as below.*



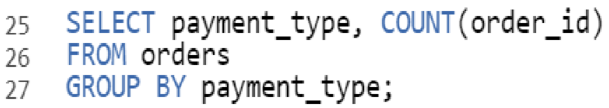
```
SELECT cashier_name, COUNT(order_id) AS Customer_served
FROM receipt_info
GROUP BY cashier_name;
```



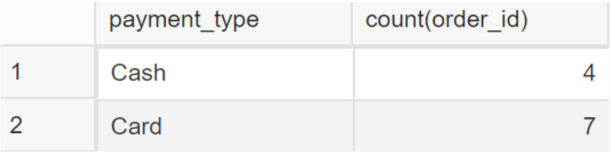
	cashier_name	customer_served
1	Mike	5
2	Kristina	6

2. Payments made by cash vs card?

- *This question helps us answer, the payment method that customers prefer and hence can help in better management of those kind of payments. (If more cash payments, how to deal with it)*
- *For this, we write a query to get the payment type and count of order_id indicating each order from order table and group by the type of payment to get following output.*



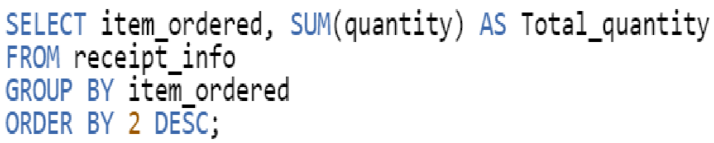
```
SELECT payment_type, COUNT(order_id)
FROM orders
GROUP BY payment_type;
```



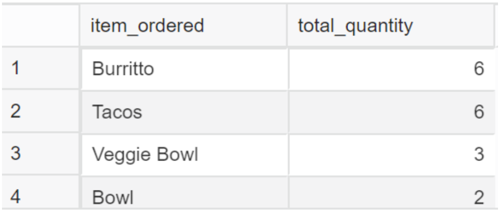
	payment_type	count(order_id)
1	Cash	4
2	Card	7

3. Which item sells the most?

- *This question helps us identify the most popular items from the menu. (Hence also helps in better stock management of each item)*
- *To answer this, we get the item ordered and sum of quantity as the total quantity, from receipt_info table and group by the item ordered arranged in descending order to get desired results.*



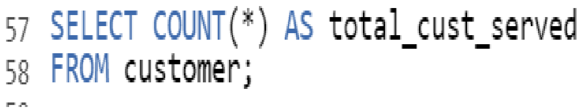
```
SELECT item_ordered, SUM(quantity) AS Total_quantity
FROM receipt_info
GROUP BY item_ordered
ORDER BY 2 DESC;
```



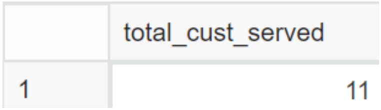
	item_ordered	total_quantity
1	Burritto	6
2	Tacos	6
3	Veggie Bowl	3
4	Bowl	2

4. Find the total number of customers served

- *This question helps us guage the number of customers visiting the store*
- *Hence, we take count of all rows as number of customers served as below*



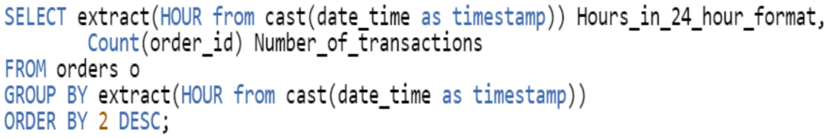
```
SELECT COUNT(*) AS total_cust_served
FROM customer;
```



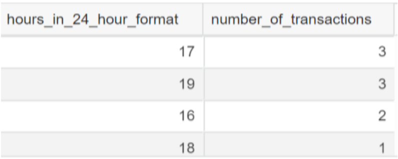
	total_cust_served
1	11

5. What is the business hour of the day by number of transactions?

- *This question helps us answer which is the most busy hour of the day by giving the number of transactions for each hour*
- *For anwering this, we extract the hour from the date_time and take the count of order_id as number of transactions from order table and group by hour and arrange the transactions by descending order to get output as follows.*



```
SELECT extract(HOUR from cast(date_time as timestamp)) Hours_in_24_hour_format,
       Count(order_id) Number_of_transactions
FROM orders o
GROUP BY extract(HOUR from cast(date_time as timestamp))
ORDER BY 2 DESC;
```



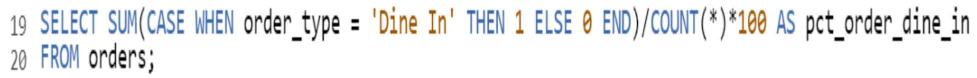
hours_in_24_hour_format	number_of_transactions
17	3
19	3
16	2
18	1

2. Use of Advanced SQL Queries

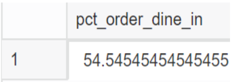
<u>The following business questions are answered using Case-When Statements, Views, Joins and Subqueries:</u>

6. What percentage of orders out of the total orders are dine-in? (Case-When Statement)

- *This question helps us answer the percentage of total orders that were dine in (as opposed to to-go).*
- *Answering this question can help Chipotle guage whether their existing seating capacity is enough or not.*
- *For this, we use Case-When statement, where we pick only the orders that were dine-in, by assigning them a value of 1, and 0 to other orders. After that we divide the number of dine-in with the total orders * 100 to get the percentage of orders that are dine-in.*



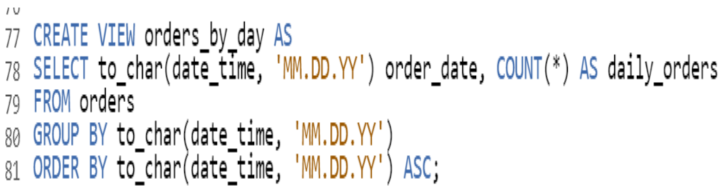
```
SELECT SUM(CASE WHEN order_type = 'Dine In' THEN 1 ELSE 0 END)/COUNT(*)*100 AS pct_order_dine_in
FROM orders;
```



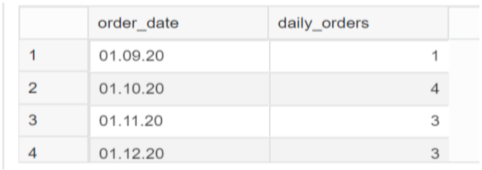
	pct_order_dine_in
1	54.54545454545455

7. The number of orders by day (Using View)

- *This question helps us answer the orders by day and we can see whether there are more sales on certain days as compared to the other days*
- *For that, we created a view (using create view statement), and inside of that wrote a SQL query to get total orders for each date as follows. Calling the view then gives the following result.*



```
CREATE VIEW orders_by_day AS
SELECT to_char(date_time, 'MM.DD.YY') order_date, COUNT(*) AS daily_orders
FROM orders
GROUP BY to_char(date_time, 'MM.DD.YY')
ORDER BY to_char(date_time, 'MM.DD.YY') ASC;
```

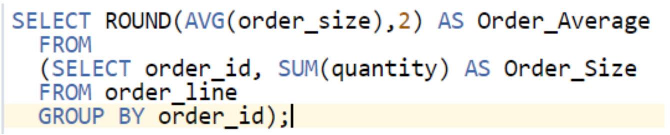


	order_date	daily_orders
1	01.09.20	1
2	01.10.20	4
3	01.11.20	3
4	01.12.20	3

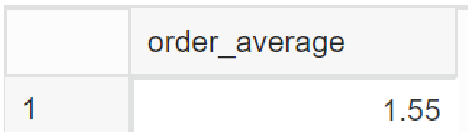
8. What is the average order size per order? (Subquery)

- *This question helps us get an average item quantity ordered for each order.*
- *For this, we write a subquery to first extract the order_size (quantity ordered) for each order and then take the average of all the order_size to get the average order size per order as below.*

.



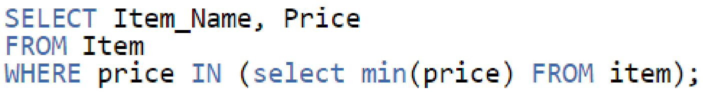
```
SELECT ROUND(AVG(order_size),2) AS Order_Average
  FROM
  (SELECT order_id, SUM(quantity) AS Order_Size
  FROM order_line
  GROUP BY order_id);
```



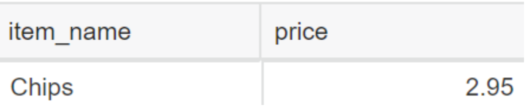
	order_average
1	1.55

9. Least priced item in the menu?

- *This question gives us the most economical priced item in the menu.*
- *For this we write a subquery to first get the minimum priced item from the menu and then display the item name and price as the end result, just as below.*



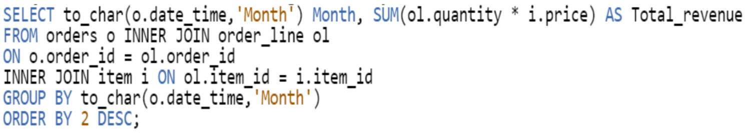
```
SELECT Item_Name, Price
FROM Item
WHERE price IN (select min(price) FROM item);
```



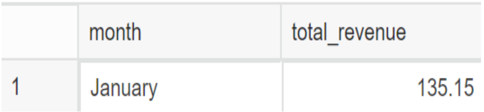
item_name	price
Chips	2.95

10. What is the total revenue generated by month?

- This question helps answer the total revenue generated for each month. For larger data, it can help find revenue generation across each month.
- For this, we join order table with order_line table and then that with the item table to get the month from date_time and (quantity*price) for revenue and sum of that for finding total revenue.
- Grouping it by month gives us the total revenue for each month as follows.



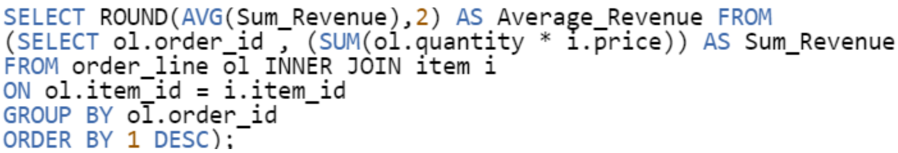
```
SELECT to_char(o.date_time,'Month') Month, SUM(ol.quantity * i.price) AS Total_revenue
FROM orders o INNER JOIN order_line ol
ON o.order_id = ol.order_id
INNER JOIN item i ON ol.item_id = i.item_id
GROUP BY to_char(o.date_time,'Month')
ORDER BY 2 DESC;
```



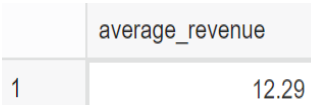
	month	total_revenue
1	January	135.15

11. What is the average revenue generated per order?

- *This question gives us an idea of the average revenue per order/customer*
- *For this, we write a subquery to get the total revenue for order, and then take the average of the total revenue received from the subquery to get the output as follows.*



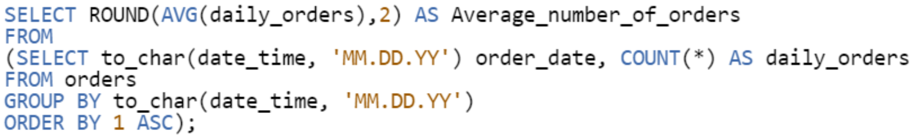
```
SELECT ROUND(AVG(Sum_Revenue),2) AS Average_Revenue FROM
(SELECT ol.order_id , (SUM(ol.quantity * i.price)) AS Sum_Revenue
FROM order_line ol INNER JOIN item i
ON ol.item_id = i.item_id
GROUP BY ol.order_id
ORDER BY 1 DESC);
```



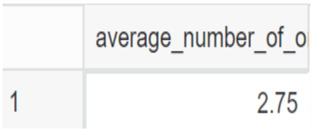
	average_revenue
1	12.29

12. What is the average number of orders in a day?

- *This question gives us an idea of the average number of orders in a day and can help us gauge how well the store is doing on each day*
- *For this we write a subquery inside of a 'FROM' to get total orders for each date and then take the average of the daily orders received from the subquery to get the output as follows.*



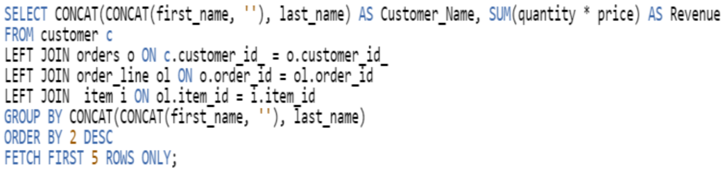
```
SELECT ROUND(AVG(daily_orders),2) AS Average_number_of_orders
FROM
(SELECT to_char(date_time, 'MM.DD.YY') order_date, COUNT(*) AS daily_orders
FROM orders
GROUP BY to_char(date_time, 'MM.DD.YY')
ORDER BY 1 ASC);
```



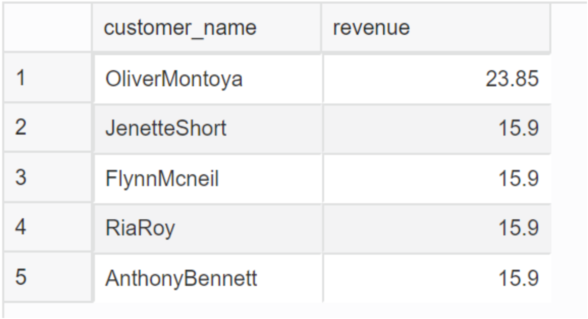
	average_number_of_o
1	2.75

13. Top 5 customers by revenue

- *This question gives us the high revenue generating customers*
- *For getting the customers and revenue, we join customer table with order table, order_line table and item table. We group by the customer and take sum of revenue and sort by summed revenue to get the top 5 customers.*



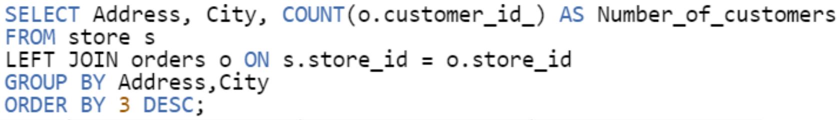
```
SELECT CONCAT(CONCAT(first_name, ''), last_name) AS Customer_Name, SUM(quantity * price) AS Revenue
FROM customer c
LEFT JOIN orders o ON c.customer_id_ = o.customer_id_
LEFT JOIN order_line ol ON o.order_id = ol.order_id
LEFT JOIN  item i ON ol.item_id = i.item_id
GROUP BY CONCAT(CONCAT(first_name, ''), last_name)
ORDER BY 2 DESC
FETCH FIRST 5 ROWS ONLY;
```



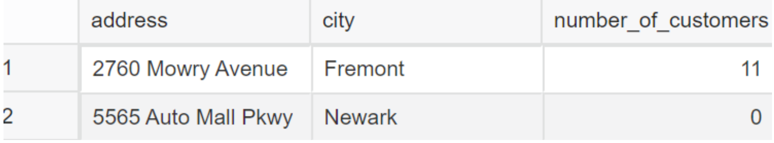
	customer_name	revenue
1	OliverMontoya	23.85
2	JenetteShort	15.9
3	FlynnMcneil	15.9
4	RiaRoy	15.9
5	AnthonyBennett	15.9

14. Number of customers by each store?

- *The above question answers, how many customers does each store cater to and hence we can identify which stores attracts more customers as compared to the other.*
- *To do this, we use 'Left Join' on store with orders. Left join helps get all the stores even if there are no orders. From that we select address, city, count of customer_id as number of customers and group by address and city (to group by each store) and order the count of customer id in descending to get the following results.*



```
SELECT Address, City, COUNT(o.customer_id_) AS Number_of_customers
FROM store s
LEFT JOIN orders o ON s.store_id = o.store_id
GROUP BY Address,City
ORDER BY 3 DESC;
```

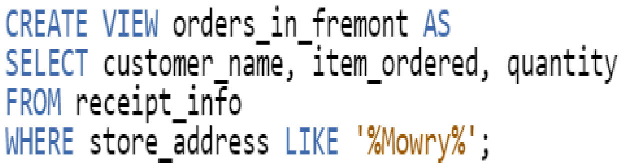


	address	city	number_of_customers
1	2760 Mowry Avenue	Fremont	11
2	5565 Auto Mall Pkwy	Newark	0

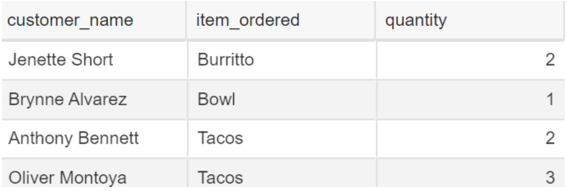
V. Views, Triggers and Stored Procedures

1. Views

a) *Created a view for displaying the order details in Fremont branch*

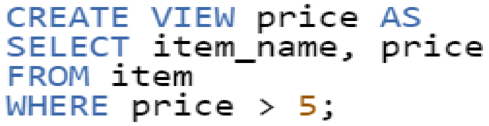


```
CREATE VIEW orders_in_fremont AS
SELECT customer_name, item_ordered, quantity
FROM receipt_info
WHERE store_address LIKE '%Mowry%';
```

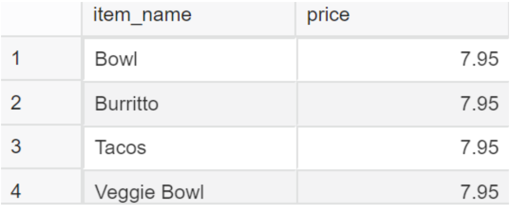


customer_name	item_ordered	quantity
Jenette Short	Burritto	2
Brynne Alvarez	Bowl	1
Anthony Bennett	Tacos	2
Oliver Montoya	Tacos	3

b) *Created a view for displaying all the items having price more than $5*



```
CREATE VIEW price AS
SELECT item_name, price
FROM item
WHERE price > 5;
```



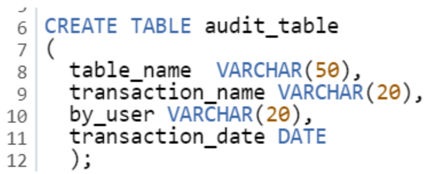
	item_name	price
1	Bowl	7.95
2	Burritto	7.95
3	Tacos	7.95
4	Veggie Bowl	7.95

2. Triggers

Creating an audit trigger when there is an insert, update or delete on an item table.

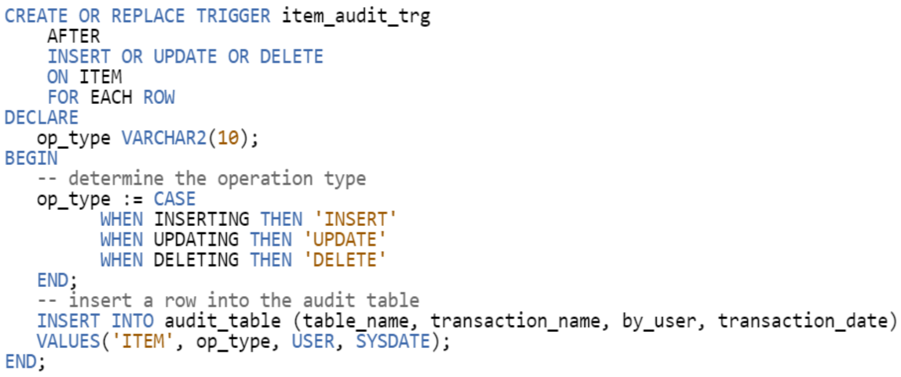
When there is either insert, update or delete action on the item table, we want to keep a history of the table name on which the action was performed, the user who performed the action and the date of action. For storing this information, we create a new table called audit_table. And then we create a trigger as follows:

Step 1: Creating a table called 'audit_table' where the details of changes will be stored



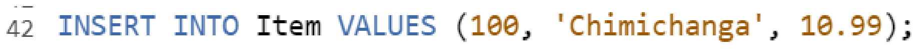
```
CREATE TABLE audit_table
(
  table_name  VARCHAR(50),
  transaction_name VARCHAR(20),
  by_user VARCHAR(20),
  transaction_date DATE
  );
```

Step 2: Create a trigger which records an insert/update/delete operation



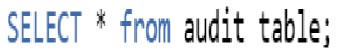
```
CREATE OR REPLACE TRIGGER item_audit_trg
    AFTER
    INSERT OR UPDATE OR DELETE
    ON ITEM
    FOR EACH ROW
DECLARE
   op_type VARCHAR2(10);
BEGIN
   -- determine the operation type
   op_type := CASE
         WHEN INSERTING THEN 'INSERT'
         WHEN UPDATING THEN 'UPDATE'
         WHEN DELETING THEN 'DELETE'
   END;
   -- insert a row into the audit table
   INSERT INTO audit_table (table_name, transaction_name, by_user, transaction_date)
   VALUES('ITEM', op_type, USER, SYSDATE);
END;
```

Step 3: We try to insert a new record on the insert table

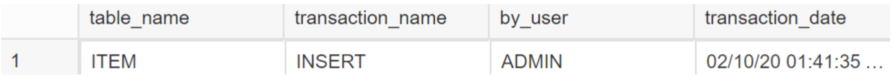


```sql
INSERT INTO Item VALUES (100, 'Chimichanga', 10.99);
```

Step 4: Checking if the trigger worked

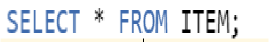


```sql
SELECT * from audit table;
```

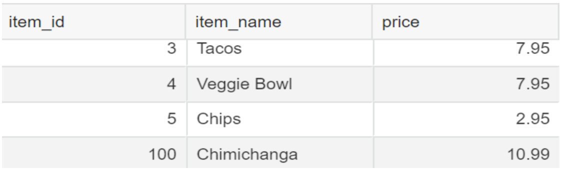


	table_name	transaction_name	by_user	transaction_date
1	ITEM	INSERT	ADMIN	02/10/20 01:41:35 ...

Step 5: We also check if the record got inserted into the table

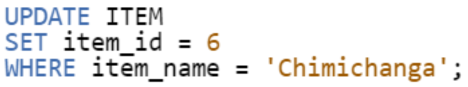


```sql
SELECT * FROM ITEM;
```



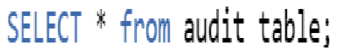
item_id	item_name	price
3	Tacos	7.95
4	Veggie Bowl	7.95
5	Chips	2.95
100	Chimichanga	10.99

Step 6: Updating a value on item table (To check if Update trigger works too)

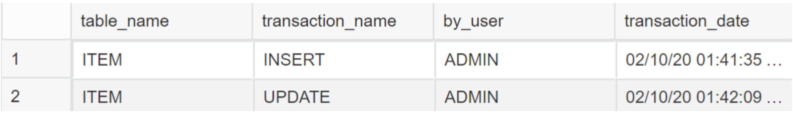


```sql
UPDATE ITEM
SET item_id = 6
WHERE item_name = 'Chimichanga';
```

Step 7: Checking the Trigger again to see if the trigger worked on update command

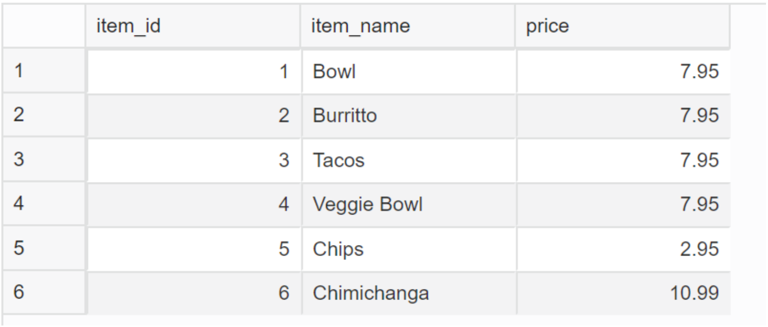


```sql
SELECT * from audit table;
```

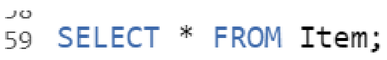


	table_name	transaction_name	by_user	transaction_date
1	ITEM	INSERT	ADMIN	02/10/20 01:41:35 ...
2	ITEM	UPDATE	ADMIN	02/10/20 01:42:09 ...

Step 8: Checking the item table to check if update worked correctly



	item_id	item_name	price
1	1	Bowl	7.95
2	2	Burritto	7.95
3	3	Tacos	7.95
4	4	Veggie Bowl	7.95
5	5	Chips	2.95
6	6	Chimichanga	10.99



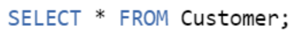
```sql
SELECT * FROM Item;
```

3. Stored Procedures

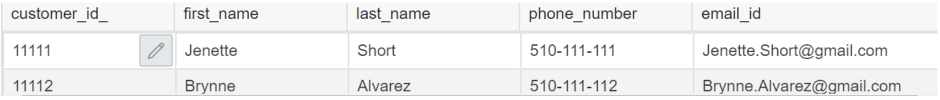
Create a procedure to update customer email id by passing the customer id.

We create a Stored procedure, to update a customer email id for a given customer id. For this the procedure takes both email-id and customer id as input. The email id provided at the input will be the new email-id that you want. The Stored procedure is thus executed as follows:

Step 1: Checking the existing table contents, before defining the procedure

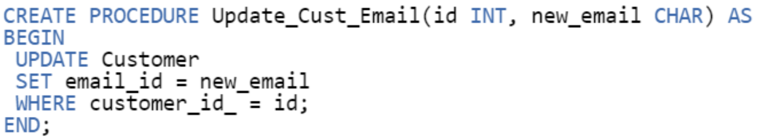


```
SELECT * FROM Customer;
```



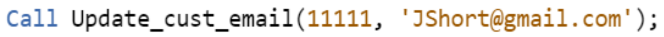
customer_id_	first_name	last_name	phone_number	email_id
11111	Jenette	Short	510-111-111	Jenette.Short@gmail.com
11112	Brynne	Alvarez	510-111-112	Brynne.Alvarez@gmail.com

Step 2: Defining a Procedure, to update customer email id for a given customer id



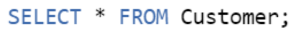
```
CREATE PROCEDURE Update_Cust_Email(id INT, new_email CHAR) AS
BEGIN
 UPDATE Customer
 SET email_id = new_email
 WHERE customer_id_ = id;
END;
```

Step 3: Calling the Procedure by giving the new value to be changed as the parameter

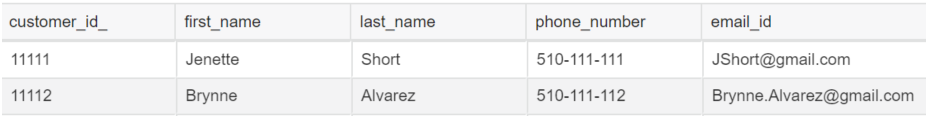


```
Call Update_cust_email(11111, 'JShort@gmail.com');
```

Step 4: Checking the Customer table to see whether Procedure worked and email-id got updated



```
SELECT * FROM Customer;
```

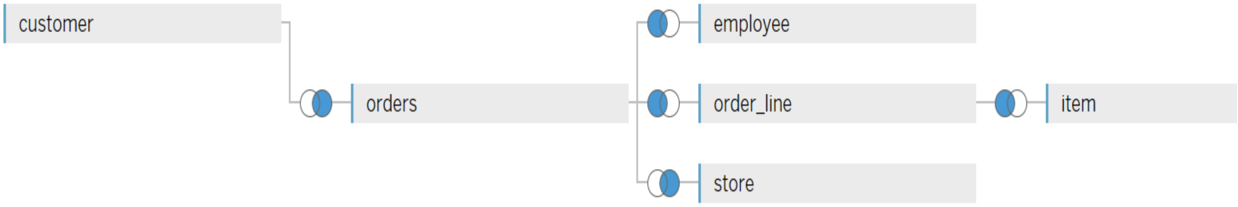


customer_id_	first_name	last_name	phone_number	email_id
11111	Jenette	Short	510-111-111	JShort@gmail.com
11112	Brynne	Alvarez	510-111-112	Brynne.Alvarez@gmail.com

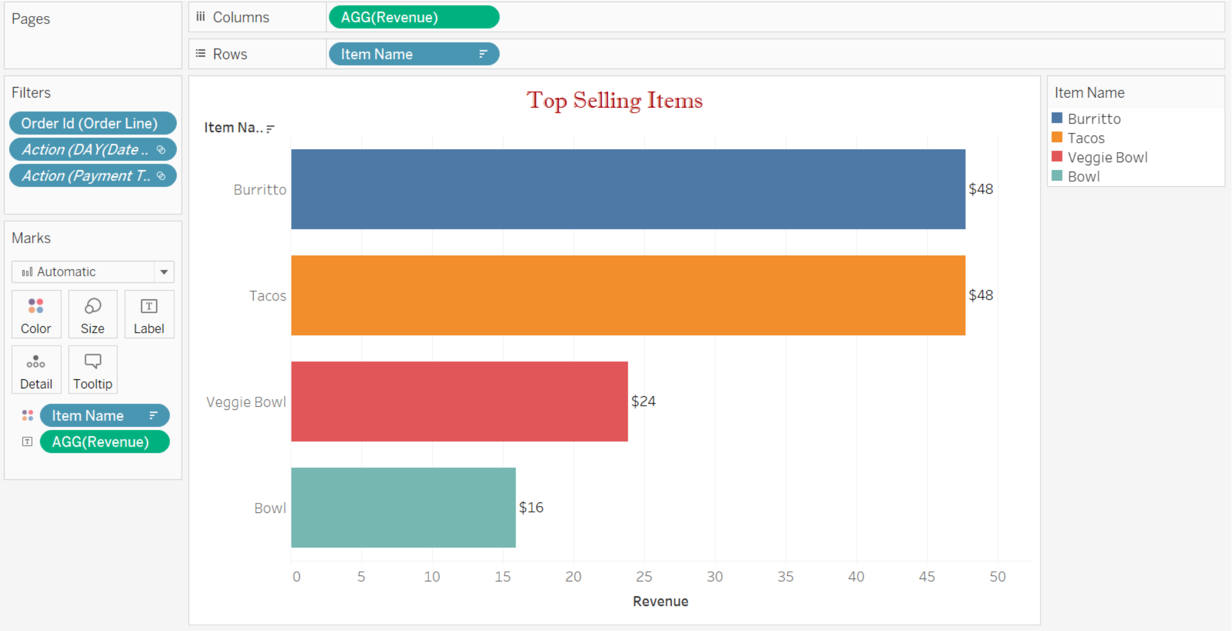
VI. Tableau Report

After having used SQL queries to manipulate data and answer business questions, it is always helpful to be able to visualize the results. Tableau is a tool which helps us do just that.

Before we can create a Dashboard, we need to join the normalized tables that we have, and only then will we be able to extract the desired information (since many of the business questions have been answered using Joins). For this dataset, the joins were performed as below:



After having done that, we can start answering business questions (or any other questions). One example of how that can be done is as follows:


Columns: AGG(Revenue)

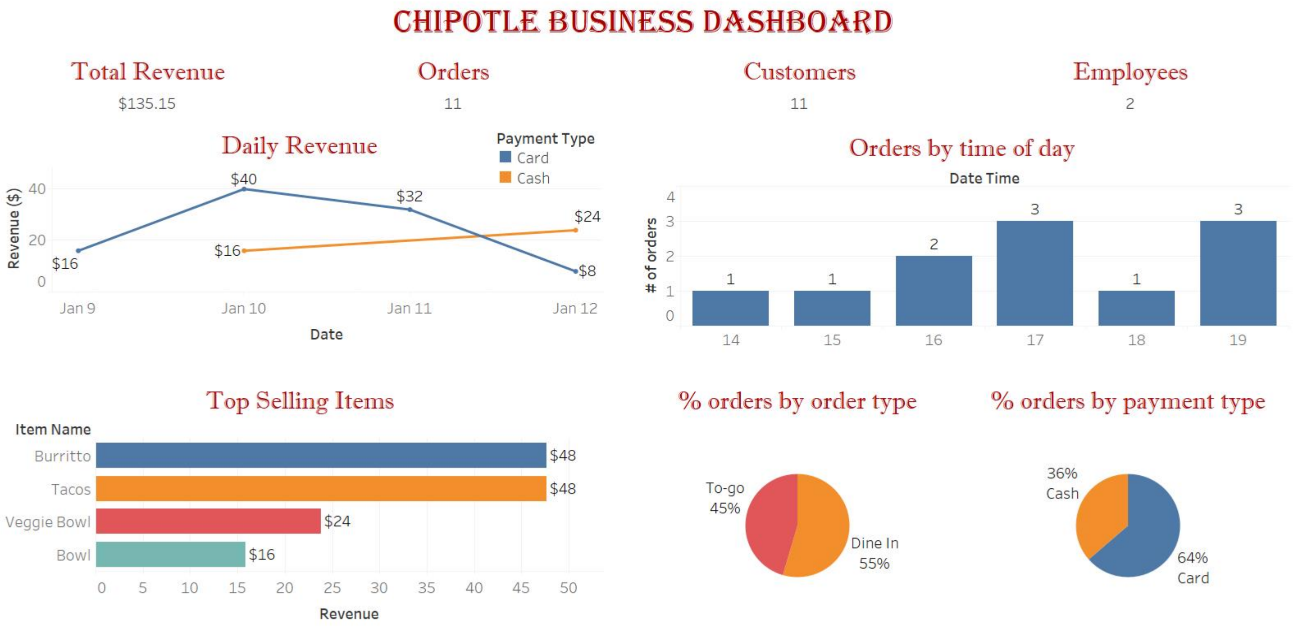
Rows: Item Name

Filters: Order Id (Order Line), Action (DAY(Date .., Action (Payment T..

Marks: Automatic; Item Name (Color), AGG(Revenue) (Label)

We can create different sheets, each answering the questions that we want and at the end, all of them can be merged to create a Dashboard, which displays various charts and details all in a single place.

1. Dashboard (pls check separate attachment on camino for better visibility)



The Dashboard created displays the following:

1. Key Metrics to date:

- Total Revenue of all the orders in the dataset
- The total orders in our data
- The total customers in our data
- The total employees who catered the customers

2. Daily Revenue

- This graph gives us the revenue generated on each day. We can try and identify patterns with this graph to see if on certain days where higher revenue is generated as compared to others.
- We see from the graph that for payments made by card there is higher revenue on Jan 10 ($40) and Jan 11 ($32) as compared to the other days. These two days are Friday and Saturday. For a larger data if this continues then we can conclude, that is higher revenue generation on weekends as compared to the weekdays.
- The graph also shows the revenue by payment type – the blue line depicting payment by card and orange line depicting payment by cash.

3. Orders by time of day

- This chart shows us the number of orders by hour.
- It helps us identify during which hour of the day there are maximum orders, thus telling us the peak hours.
- Here we see that at 17:00 hrs and 19:00 hrs there are maximum orders (3 each as compared to other hours of the day)
- If this continues for bigger dataset, we can probably conclude, that there are more orders closer to dinner time.

4. Top Selling Items

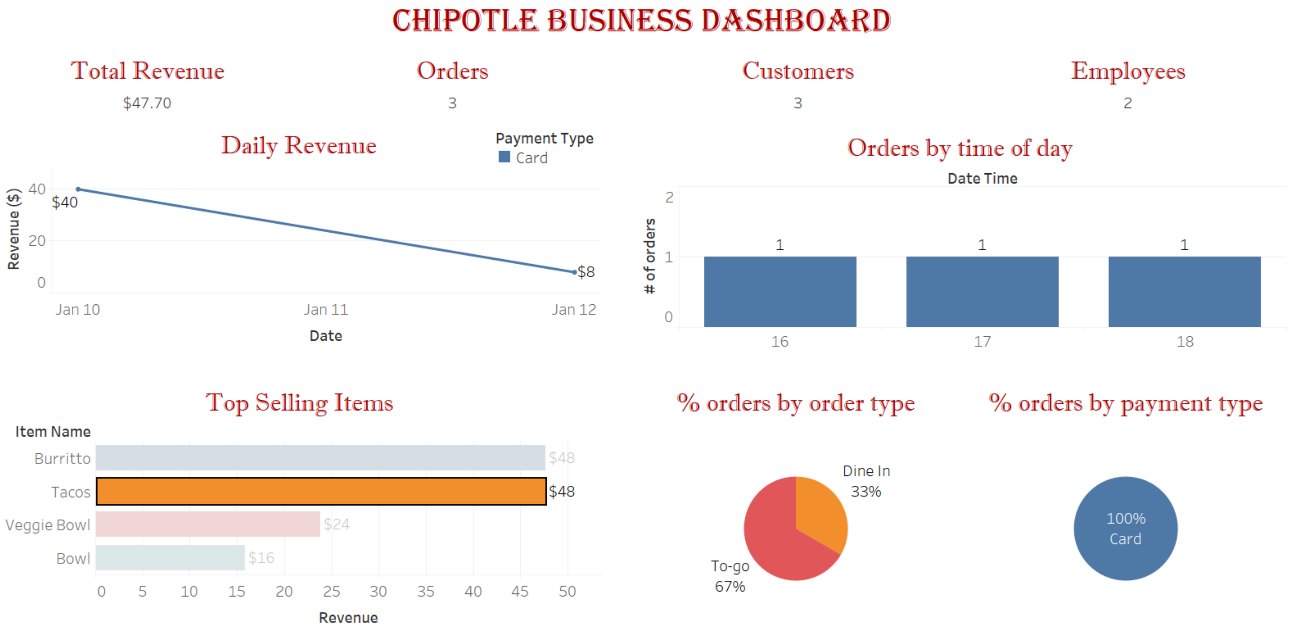
- In this chart we see the item that generates the most revenue. It tells us the most popular items.
- From this graph we see that burritos and tacos are the most popular items as compared to others since they generate the highest revenue ($48 each) as compared to others.
- This can help the store plan their inventory better

5. Percentage orders by order type and payment type

- The first pie chart tells us that out of all the orders, 55% were dine-in orders and remaining 45% were to-go. We see that customers prefer dining in rather than take out. This can be used by the store to check if the seating that they have is enough to cater to customers during peak hours
- The second pie chart tells us that out of all the orders, 64% are paid by card and 36% are paid by cash. Customers prefer paying by card, and this can help them manage the cash at the store accordingly.

2. Dashboard with Interaction in effect (pls check separate attachment on camino for better visibility)

We can make the dashboard interactive. For example, here if we select only Tacos from Top selling items, we can see all the information about Tacos as below.



From the above interaction when only Tacos is selected, we observe the following:

1. The Total Revenue generated from Tacos is $47.70
2. There is a total of three orders of Tacos
3. And three different customers ordered the Tacos.
4. Two employees helped the customers with their orders
5. The Revenue generated from Tacos was $40 on Jan 10th and approx. $8 on Jan 12
6. 1 of the Taco was ordered at 4:00 pm, other and 5 pm and other at 6 pm
7. Out of the total orders of Tacos, 67% (2 orders) were to-go and 33% were dine-in
8. And only card was used to make the payment for Taco orders.

Many such interactions can be seen and observed as per how we like.

VII. Conclusion

For the purpose of this project we used both SQL and Tableau.

SQL:

- After having used SQL, we see that using SQL helps facilitate data creation and manipulation to extract results.
- SQL is much faster and easier as compared to Excel. Getting data from multiple columns/ tables is also quite convenient

Tableau:

- After manipulation and extraction of required information from SQL, it is very helpful to visualize the questions that we are trying to answer.
- Visualization also helps in making the data easier to interpret.

Having implemented the above, and worked on the whole model, some of the limitations and **improvements** to overcome those limitations in the model according to me are:

- We see that in the Item table, if the item price is changed due to inflation it will create problems, since only current price is used to calculate the revenue. We do not have a way to keep a track of historical price for revenue calculations. To avoid that, we can first add a date column to the item table which will represent price of an item on each day. Then we can join the item table to the order table on order date and item id to get the accurate revenue.

- Currently my denormalized data contains information about taxes for each order. However, for the normalized table the taxes are not taken into consideration. For revenue calculation, we have multiplied price and quantity. The price here is without taxes. If we take taxes into consideration, then we get more accurate and realistic results for revenue.

- Currently, even though for processing/making an order there are more employees required who make the order and also the ones at the backend, taking care of things like stock replenishment, etc. In the current model however, there are only two employees (cashiers) whose data is entered into the system. It is always useful to include details about all the employees. One way this can be incorporated is in the order table, we can mention which employee takes care of a particular order by adding columns like Employee1, Employee2, etc. Other way we can include employee information is by creating a new table where we can capture the information about how many employees worked on each order. All this is useful to get an idea of how many orders are processed by how many employees.