Business Analysis

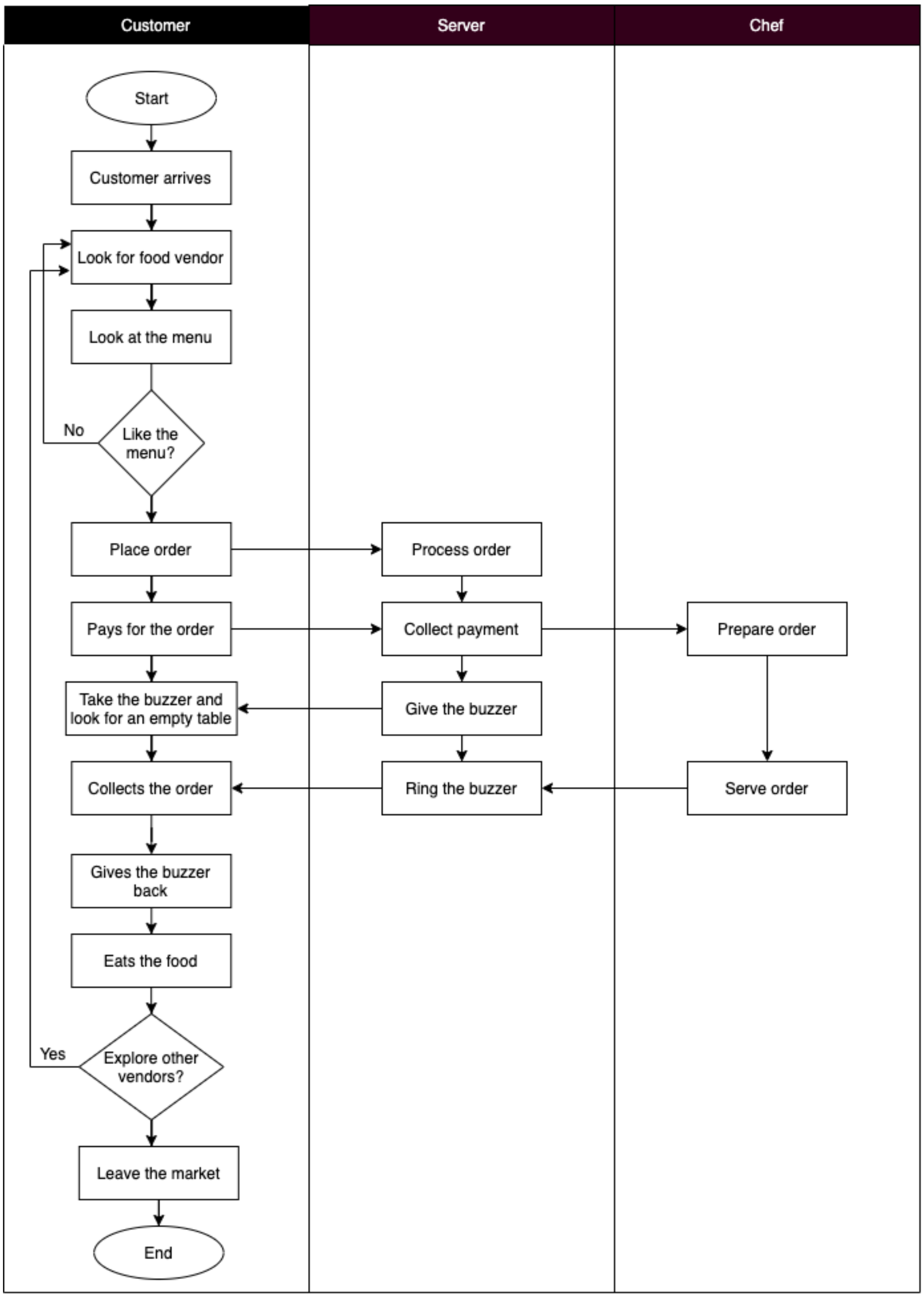
Name: Tanmayi Varanasi

Class: MSBA

Project Objective:

This project allows us to have deep understanding of how data can be generated, stored and utilized to solve business problems. In any business, data generation is a very important step which can help answer various business questions. To understand the data generation process, it is important to know the end to end business activity that happens on a daily basis. This assignment gives us an opportunity to build a business scenario based on our personal experience. As a part of the business analysis, we built a database to replicate 10 customers based on our 'requirements analysis'.

Swimlane Diagram of the scenario below:



Business Scenario:

On the 12th of Jan 2020, I went to meet some friends for lunch at a food market called "San Pedro Square", located in downtown San Jose. I parked my car in the garage opposite to the

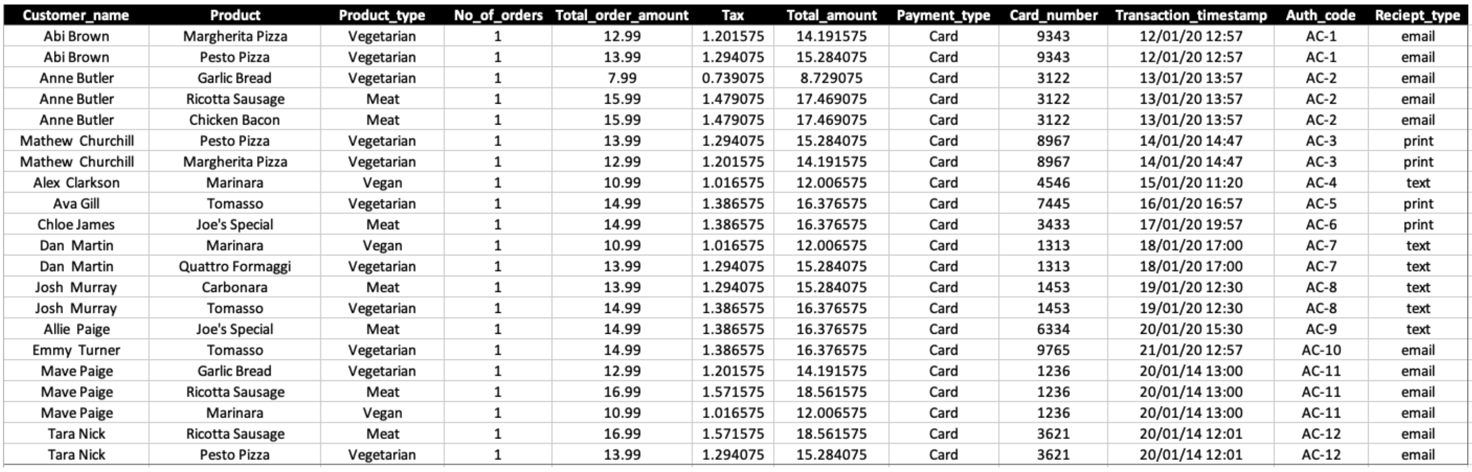
Square at 12 PM, met up with my friends and walked into the market. There were quite a few food options available. I found a pizza place called "Pizza Bocca Lupo" that looked interesting. I waited in line to order. When my turn came, I ordered one Margherita and one Pesto pizza. The cashier/server took my order and told me the amount to be paid is $32.43. After I paid for the order using my credit card, the cashier gave me a receipt and a buzzer. The server then gave the order to the chef. Meanwhile, I looked for an empty table and settled down. Once the order was prepared and assembled by the chef, the server rang the buzzer to notify that my food was ready. Once I returned the buzzer and collected my food, I sat my table and enjoyed my pizza with my friends. After I finished eating, I took all my belongings and left the market at 1:30 PM.

Step 1: Data creation in Excel

Denormalized table

The table below captures all the order information including the customer name, the items ordered and the payment details. This is all the information that can be ascertained from an order receipt at "**Pizza Bocca Lupo**".

This table is useful for doing a quick look up to get information such as- Which customer ordered which item or how much did one customer spend in the store etc.



Customer_name	Product	Product_type	No_of_orders	Total_order_amount	Tax	Total_amount	Payment_type	Card_number	Transaction_timestamp	Auth_code	Reciept_type
Abi Brown	Margherita Pizza	Vegetarian	1	12.99	1.201575	14.191575	Card	9343	12/01/20 12:57	AC-1	email
Abi Brown	Pesto Pizza	Vegetarian	1	13.99	1.294075	15.284075	Card	9343	12/01/20 12:57	AC-1	email
Anne Butler	Garlic Bread	Vegetarian	1	7.99	0.739075	8.729075	Card	3122	13/01/20 13:57	AC-2	email
Anne Butler	Ricotta Sausage	Meat	1	15.99	1.479075	17.469075	Card	3122	13/01/20 13:57	AC-2	email
Anne Butler	Chicken Bacon	Meat	1	15.99	1.479075	17.469075	Card	3122	13/01/20 13:57	AC-2	email
Mathew Churchill	Pesto Pizza	Vegetarian	1	13.99	1.294075	15.284075	Card	8967	14/01/20 14:47	AC-3	print
Mathew Churchill	Margherita Pizza	Vegetarian	1	12.99	1.201575	14.191575	Card	8967	14/01/20 14:47	AC-3	print
Alex Clarkson	Marinara	Vegan	1	10.99	1.016575	12.006575	Card	4546	15/01/20 11:20	AC-4	text
Ava Gill	Tomasso	Vegetarian	1	14.99	1.386575	16.376575	Card	7445	16/01/20 16:57	AC-5	print
Chloe James	Joe's Special	Meat	1	14.99	1.386575	16.376575	Card	3433	17/01/20 19:57	AC-6	print
Dan Martin	Marinara	Vegan	1	10.99	1.016575	12.006575	Card	1313	18/01/20 17:00	AC-7	text
Dan Martin	Quattro Formaggi	Vegetarian	1	13.99	1.294075	15.284075	Card	1313	18/01/20 17:00	AC-7	text
Josh Murray	Carbonara	Meat	1	13.99	1.294075	15.284075	Card	1453	19/01/20 12:30	AC-8	text
Josh Murray	Tomasso	Vegetarian	1	14.99	1.386575	16.376575	Card	1453	19/01/20 12:30	AC-8	text
Allie Paige	Joe's Special	Meat	1	14.99	1.386575	16.376575	Card	6334	20/01/20 15:30	AC-9	text
Emmy Turner	Tomasso	Vegetarian	1	14.99	1.386575	16.376575	Card	9765	21/01/20 12:57	AC-10	email
Mave Paige	Garlic Bread	Vegetarian	1	12.99	1.201575	14.191575	Card	1236	20/01/14 13:00	AC-11	email
Mave Paige	Ricotta Sausage	Meat	1	16.99	1.571575	18.561575	Card	1236	20/01/14 13:00	AC-11	email
Mave Paige	Marinara	Vegan	1	10.99	1.016575	12.006575	Card	1236	20/01/14 13:00	AC-11	email
Tara Nick	Ricotta Sausage	Meat	1	16.99	1.571575	18.561575	Card	3621	20/01/14 12:01	AC-12	email
Tara Nick	Pesto Pizza	Vegetarian	1	13.99	1.294075	15.284075	Card	3621	20/01/14 12:01	AC-12	email

But, the drawback with the denormalized table is that, there exists **Insertion**, **Deletion** and **Modification** anomalies.

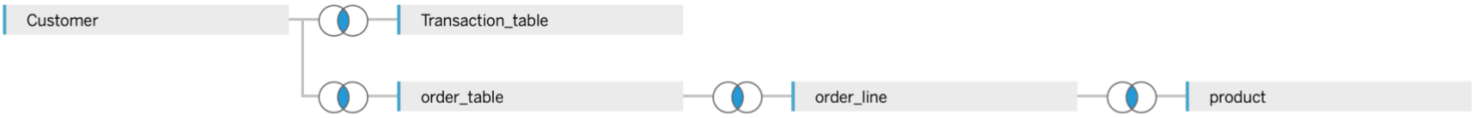
To overcome the above anomalies, Table 1 was normalized to the Third normal form (3NF).

Steps involved to get tables in 3NF:

- Converting each cell to be a single value and unique records (1NF).
- Remove all the partial dependencies
- Remove all the transitive dependencies

Based on the information in the table above, the data was split into multiple tables to better represent the relationships between different fields. This way, the information can be maintained better without any redundancies and deletion and updating is no more a hassle.

Tables in 3NF:

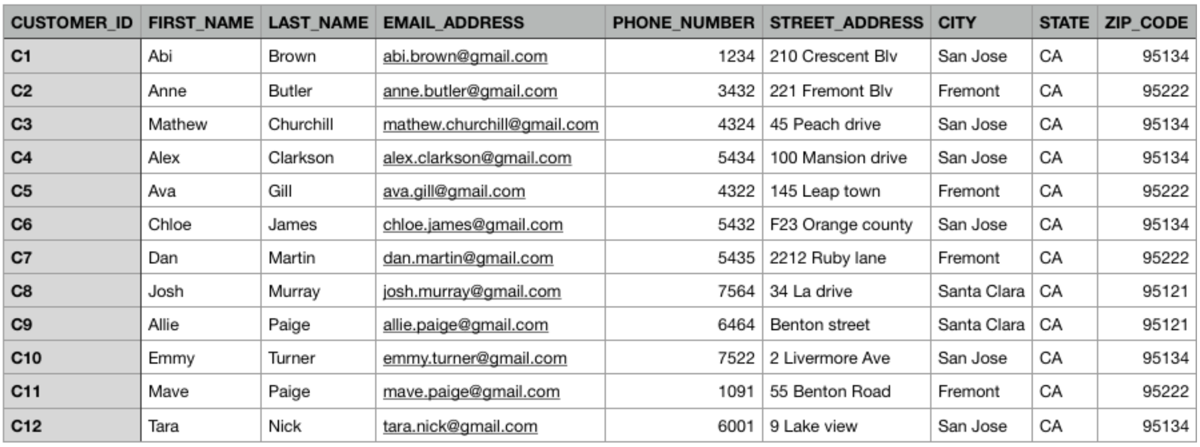


Let’s look at these tables in detail.

Customer Table

In the denormalized table, each row represented one item in the order for every customer. Due to that, there were multiple rows with the same customer name in the case where a customer ordered more than one item.

But the table below is a representation of customer information in 3NF. Here, each record in the table pertains to a single customer. This helps maintain a clean database without any duplication.

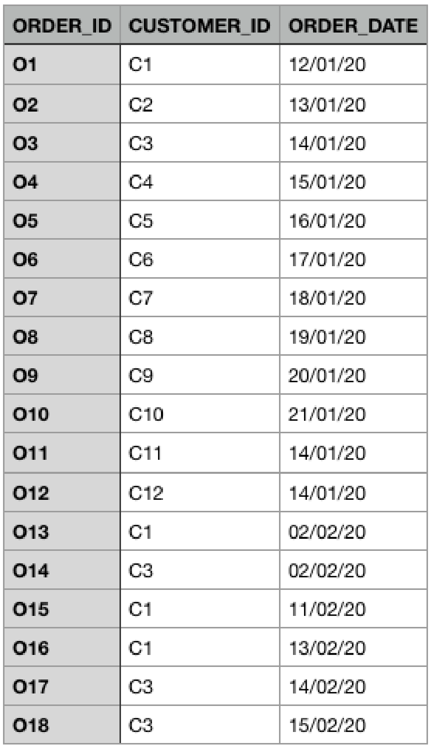


CUSTOMER_ID	FIRST_NAME	LAST_NAME	EMAIL_ADDRESS	PHONE_NUMBER	STREET_ADDRESS	CITY	STATE	ZIP_CODE
C1	Abi	Brown	abi.brown@gmail.com	1234	210 Crescent Blv	San Jose	CA	95134
C2	Anne	Butler	anne.butler@gmail.com	3432	221 Fremont Blv	Fremont	CA	95222
C3	Mathew	Churchill	mathew.churchill@gmail.com	4324	45 Peach drive	San Jose	CA	95134
C4	Alex	Clarkson	alex.clarkson@gmail.com	5434	100 Mansion drive	San Jose	CA	95134
C5	Ava	Gill	ava.gill@gmail.com	4322	145 Leap town	Fremont	CA	95222
C6	Chloe	James	chloe.james@gmail.com	5432	F23 Orange county	San Jose	CA	95134
C7	Dan	Martin	dan.martin@gmail.com	5435	2212 Ruby lane	Fremont	CA	95222
C8	Josh	Murray	josh.murray@gmail.com	7564	34 La drive	Santa Clara	CA	95121
C9	Allie	Paige	allie.paige@gmail.com	6464	Benton street	Santa Clara	CA	95121
C10	Emmy	Turner	emmy.turner@gmail.com	7522	2 Livermore Ave	San Jose	CA	95134
C11	Mave	Paige	mave.paige@gmail.com	1091	55 Benton Road	Fremont	CA	95222
C12	Tara	Nick	tara.nick@gmail.com	6001	9 Lake view	San Jose	CA	95134

Figure 1

Order Table

This table gives an order level view for each customer. Unlike the denormalized table, the order information here is much more organized. Each row represents the mapping from order ID to the customer ID.

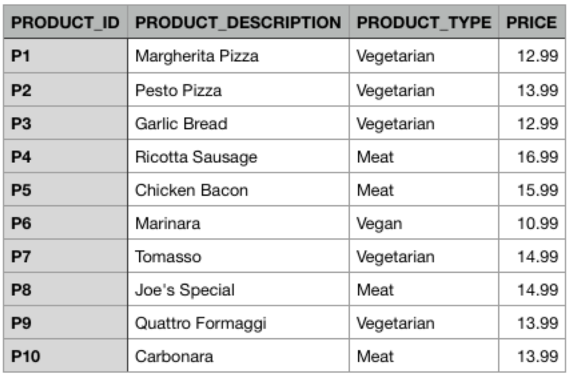


ORDER_ID	CUSTOMER_ID	ORDER_DATE
O1	C1	12/01/20
O2	C2	13/01/20
O3	C3	14/01/20
O4	C4	15/01/20
O5	C5	16/01/20
O6	C6	17/01/20
O7	C7	18/01/20
O8	C8	19/01/20
O9	C9	20/01/20
O10	C10	21/01/20
O11	C11	14/01/20
O12	C12	14/01/20
O13	C1	02/02/20
O14	C3	02/02/20
O15	C1	11/02/20
O16	C1	13/02/20
O17	C3	14/02/20
O18	C3	15/02/20

Figure 2.

Product Table

This table contains all the product information only. Each row represents each item on the menu. This table is volatile. The fields are subject to frequent changes. For example, depending on the demand, the price of an item can be changed, or the items can be added or removed from the menu, changing the number of rows. These kinds of alterations would not affect any other table.

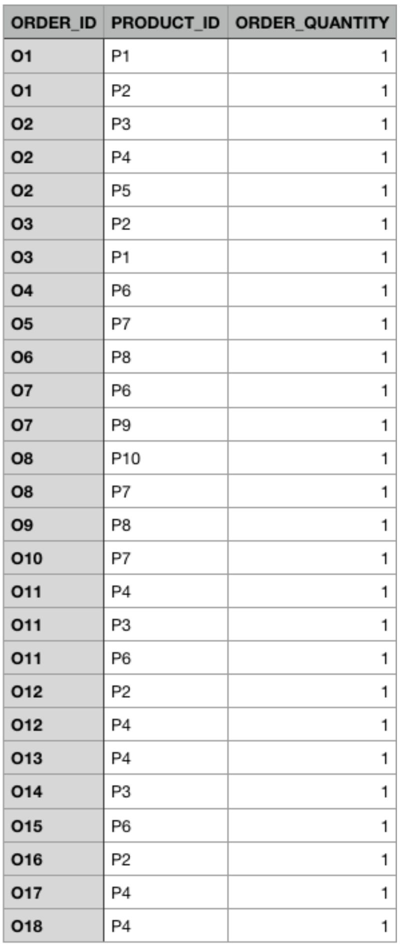


PRODUCT_ID	PRODUCT_DESCRIPTION	PRODUCT_TYPE	PRICE
P1	Margherita Pizza	Vegetarian	12.99
P2	Pesto Pizza	Vegetarian	13.99
P3	Garlic Bread	Vegetarian	12.99
P4	Ricotta Sausage	Meat	16.99
P5	Chicken Bacon	Meat	15.99
P6	Marinara	Vegan	10.99
P7	Tomasso	Vegetarian	14.99
P8	Joe's Special	Meat	14.99
P9	Quattro Formaggi	Vegetarian	13.99
P10	Carbonara	Meat	13.99

Figure 3.

Order Line table

This table contains all the product/item level purchase information for each order.

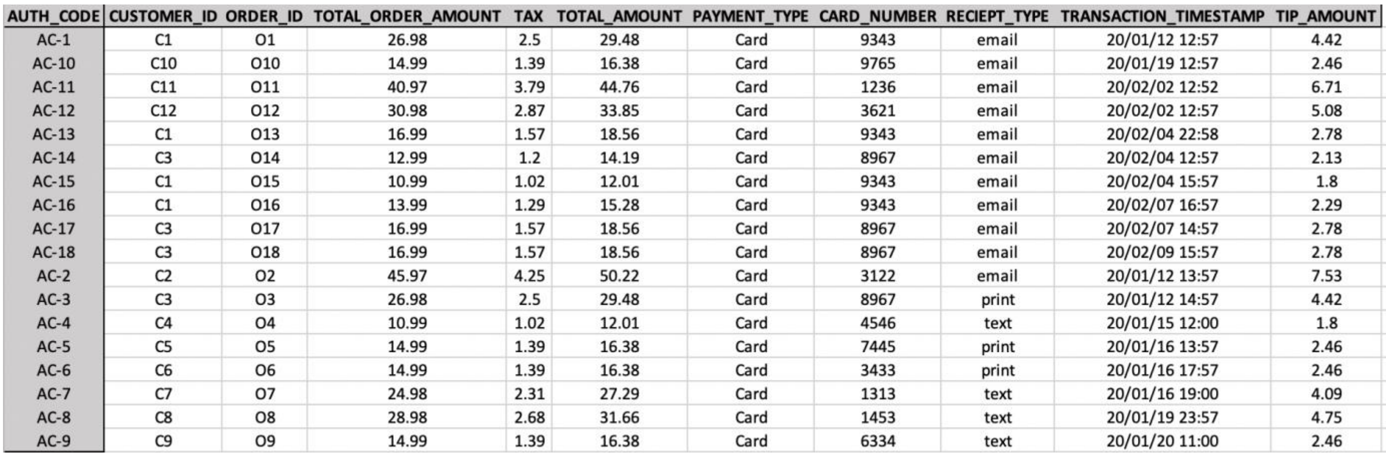


ORDER_ID	PRODUCT_ID	ORDER_QUANTITY
O1	P1	1
O1	P2	1
O2	P3	1
O2	P4	1
O2	P5	1
O3	P2	1
O3	P1	1
O4	P6	1
O5	P7	1
O6	P8	1
O7	P6	1
O7	P9	1
O8	P10	1
O8	P7	1
O9	P8	1
O10	P7	1
O11	P4	1
O11	P3	1
O11	P6	1
O12	P2	1
O12	P4	1
O13	P4	1
O14	P3	1
O15	P6	1
O16	P2	1
O17	P4	1
O18	P4	1

Figure 4.

Transaction Table

This table represents the details about the transactions processed by the cashier. Each record represents one order which could consists of multiple items. This tables also captures information such as total order amount, taxes, tips, receipt type etc.



AUTH_CODE	CUSTOMER_ID	ORDER_ID	TOTAL_ORDER_AMOUNT	TAX	TOTAL_AMOUNT	PAYMENT_TYPE	CARD_NUMBER	RECIEPT_TYPE	TRANSACTION_TIMESTAMP	TIP_AMOUNT
AC-1	C1	O1	26.98	2.5	29.48	Card	9343	email	20/01/12 12:57	4.42
AC-10	C10	O10	14.99	1.39	16.38	Card	9765	email	20/01/19 12:57	2.46
AC-11	C11	O11	40.97	3.79	44.76	Card	1236	email	20/02/02 12:52	6.71
AC-12	C12	O12	30.98	2.87	33.85	Card	3621	email	20/02/02 12:57	5.08
AC-13	C1	O13	16.99	1.57	18.56	Card	9343	email	20/02/04 22:58	2.78
AC-14	C3	O14	12.99	1.2	14.19	Card	8967	email	20/02/04 12:57	2.13
AC-15	C1	O15	10.99	1.02	12.01	Card	9343	email	20/02/04 15:57	1.8
AC-16	C1	O16	13.99	1.29	15.28	Card	9343	email	20/02/07 16:57	2.29
AC-17	C3	O17	16.99	1.57	18.56	Card	8967	email	20/02/07 14:57	2.78
AC-18	C3	O18	16.99	1.57	18.56	Card	8967	email	20/02/09 15:57	2.78
AC-2	C2	O2	45.97	4.25	50.22	Card	3122	email	20/01/12 13:57	7.53
AC-3	C3	O3	26.98	2.5	29.48	Card	8967	print	20/01/12 14:57	4.42
AC-4	C4	O4	10.99	1.02	12.01	Card	4546	text	20/01/15 12:00	1.8
AC-5	C5	O5	14.99	1.39	16.38	Card	7445	print	20/01/16 13:57	2.46
AC-6	C6	O6	14.99	1.39	16.38	Card	3433	print	20/01/16 17:57	2.46
AC-7	C7	O7	24.98	2.31	27.29	Card	1313	text	20/01/16 19:00	4.09
AC-8	C8	O8	28.98	2.68	31.66	Card	1453	text	20/01/19 23:57	4.75
AC-9	C9	O9	14.99	1.39	16.38	Card	6334	text	20/01/20 11:00	2.46

Figure 5.

The above tables eliminate redundant data and ensures that data dependencies make sense.

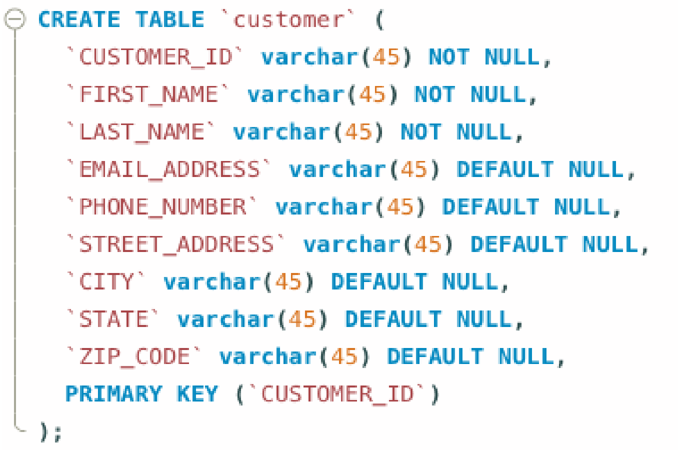
Step 2: Create tables in SQL

Once we create our tables in excel, we can import them into an SQL database.

To import them, we need to first create the corresponding tables in SQL. The basic syntax to create a table is:

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

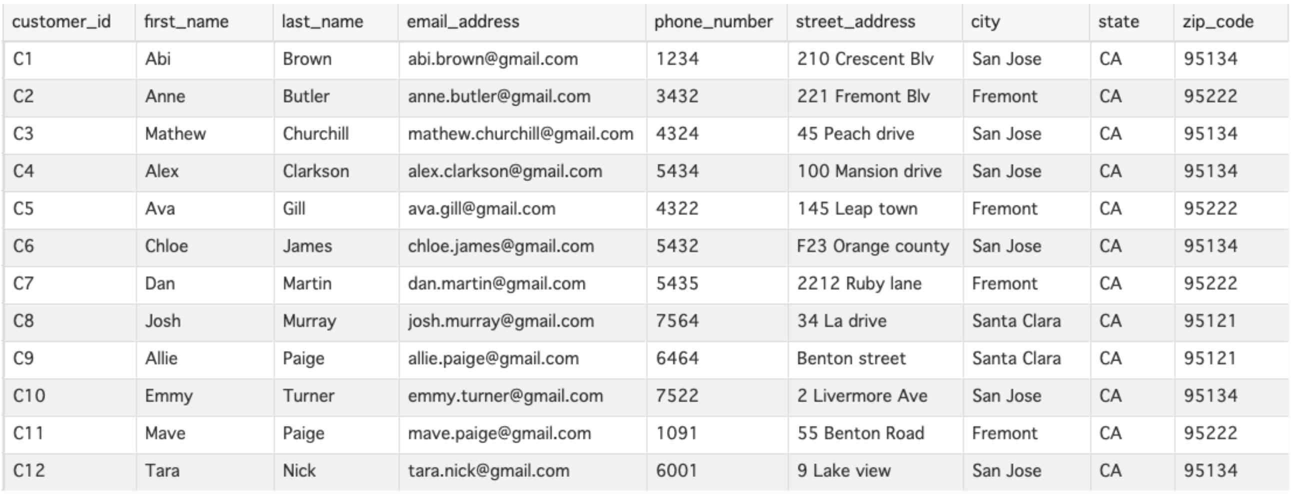
1. Script to create customer table:



```
CREATE TABLE `customer` (
  `CUSTOMER_ID` varchar(45) NOT NULL,
  `FIRST_NAME` varchar(45) NOT NULL,
  `LAST_NAME` varchar(45) NOT NULL,
  `EMAIL_ADDRESS` varchar(45) DEFAULT NULL,
  `PHONE_NUMBER` varchar(45) DEFAULT NULL,
  `STREET_ADDRESS` varchar(45) DEFAULT NULL,
  `CITY` varchar(45) DEFAULT NULL,
  `STATE` varchar(45) DEFAULT NULL,
  `ZIP_CODE` varchar(45) DEFAULT NULL,
  PRIMARY KEY (`CUSTOMER_ID`)
);
```

The above script creates a customer table with columns like Customer_ID, First name, Last name, email address, phone number and address.

Every table has a primary key column which uniquely identifies each record in that table.



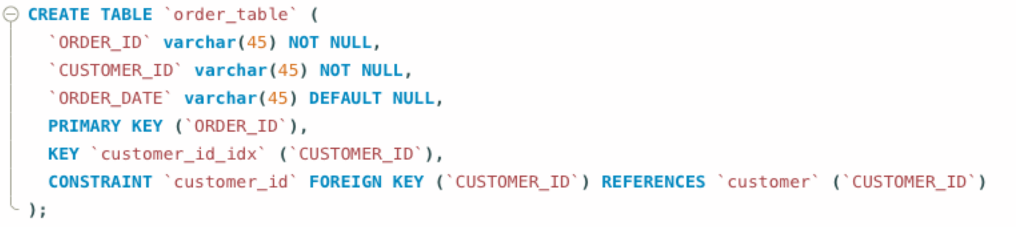
customer_id	first_name	last_name	email_address	phone_number	street_address	city	state	zip_code
C1	Abi	Brown	abi.brown@gmail.com	1234	210 Crescent Blv	San Jose	CA	95134
C2	Anne	Butler	anne.butler@gmail.com	3432	221 Fremont Blv	Fremont	CA	95222
C3	Mathew	Churchill	mathew.churchill@gmail.com	4324	45 Peach drive	San Jose	CA	95134
C4	Alex	Clarkson	alex.clarkson@gmail.com	5434	100 Mansion drive	San Jose	CA	95134
C5	Ava	Gill	ava.gill@gmail.com	4322	145 Leap town	Fremont	CA	95222
C6	Chloe	James	chloe.james@gmail.com	5432	F23 Orange county	San Jose	CA	95134
C7	Dan	Martin	dan.martin@gmail.com	5435	2212 Ruby lane	Fremont	CA	95222
C8	Josh	Murray	josh.murray@gmail.com	7564	34 La drive	Santa Clara	CA	95121
C9	Allie	Paige	allie.paige@gmail.com	6464	Benton street	Santa Clara	CA	95121
C10	Emmy	Turner	emmy.turner@gmail.com	7522	2 Livermore Ave	San Jose	CA	95134
C11	Mave	Paige	mave.paige@gmail.com	1091	55 Benton Road	Fremont	CA	95222
C12	Tara	Nick	tara.nick@gmail.com	6001	9 Lake view	San Jose	CA	95134

Output: Customer table

Customer_Id is the primary key in this table. It is an alpha numeric string that uniquely identifies each customer that has done business with “Pizza Bocca Lupo”.

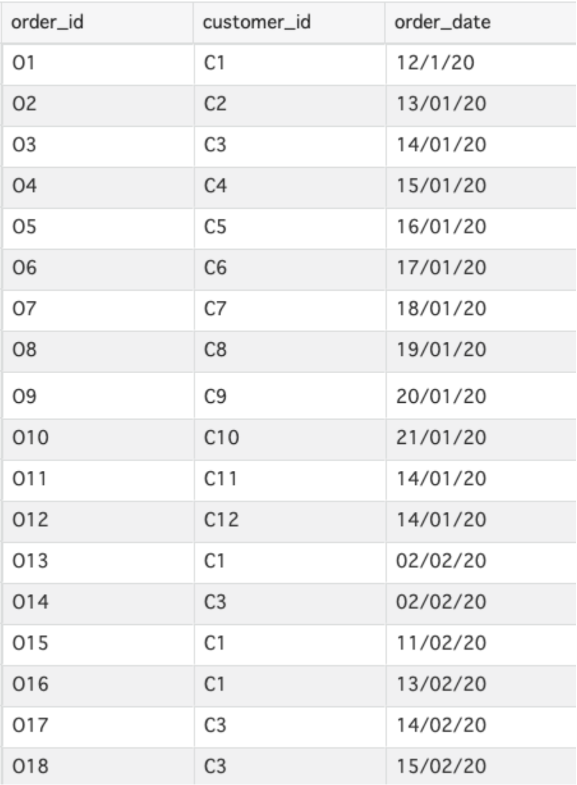
The purpose of this table is to have all the customer related information in one table. This table can give information such as- How many customers are there in the database? Or Which city do customers come from? Etc.

2. Script to create Order table:



```
CREATE TABLE `order_table` (
  `ORDER_ID` varchar(45) NOT NULL,
  `CUSTOMER_ID` varchar(45) NOT NULL,
  `ORDER_DATE` varchar(45) DEFAULT NULL,
  PRIMARY KEY (`ORDER_ID`),
  KEY `customer_id_idx` (`CUSTOMER_ID`),
  CONSTRAINT `customer_id` FOREIGN KEY (`CUSTOMER_ID`) REFERENCES `customer` (`CUSTOMER_ID`)
);
```

The above script creates an order table with columns like Order_ID, Customer_ID and Order_date. The primary key in this table is **Order_id** which uniquely identifies an order made each customer.

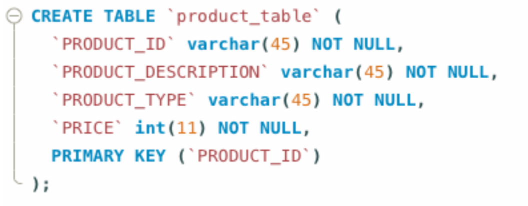


order_id	customer_id	order_date
O1	C1	12/1/20
O2	C2	13/01/20
O3	C3	14/01/20
O4	C4	15/01/20
O5	C5	16/01/20
O6	C6	17/01/20
O7	C7	18/01/20
O8	C8	19/01/20
O9	C9	20/01/20
O10	C10	21/01/20
O11	C11	14/01/20
O12	C12	14/01/20
O13	C1	02/02/20
O14	C3	02/02/20
O15	C1	11/02/20
O16	C1	13/02/20
O17	C3	14/02/20
O18	C3	15/02/20

Output: Order table

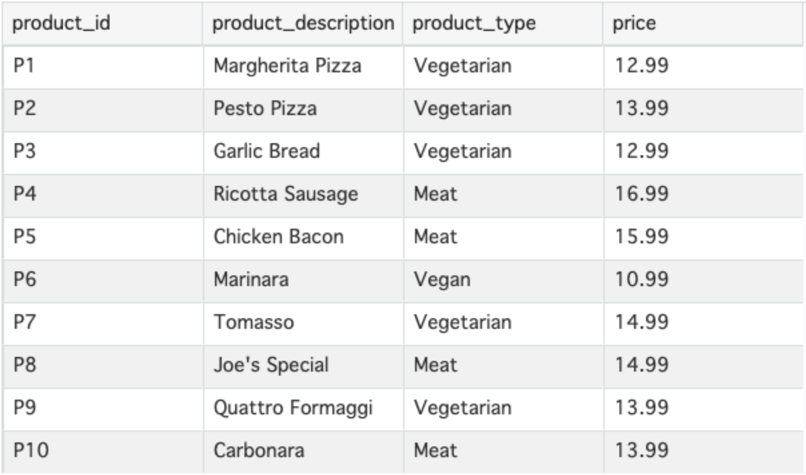
The purpose of this table is to keep track of the customer orders and the date of order. With the help of this table we can write queries for questions like- On an average, how many orders are placed by each customer? Or How many orders on an average does this place receive in day, week or month etc.

3. Script to create Product table:



```
CREATE TABLE `product_table` (
  `PRODUCT_ID` varchar(45) NOT NULL,
  `PRODUCT_DESCRIPTION` varchar(45) NOT NULL,
  `PRODUCT_TYPE` varchar(45) NOT NULL,
  `PRICE` int(11) NOT NULL,
  PRIMARY KEY (`PRODUCT_ID`)
);
```

This script creates a product table with fields like Product_id, Product description, Product type and Price. Here, the primary key is defined by the column **Product_Id** where each record represents an item on the menu.

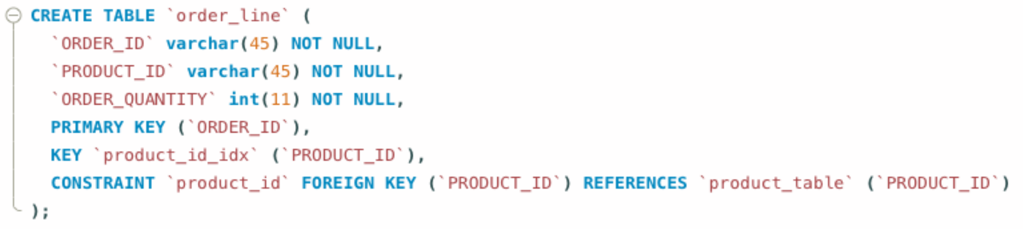


product_id	product_description	product_type	price
P1	Margherita Pizza	Vegetarian	12.99
P2	Pesto Pizza	Vegetarian	13.99
P3	Garlic Bread	Vegetarian	12.99
P4	Ricotta Sausage	Meat	16.99
P5	Chicken Bacon	Meat	15.99
P6	Marinara	Vegan	10.99
P7	Tomasso	Vegetarian	14.99
P8	Joe's Special	Meat	14.99
P9	Quattro Formaggi	Vegetarian	13.99
P10	Carbonara	Meat	13.99

Output: Product table

This table has all the product related fields.

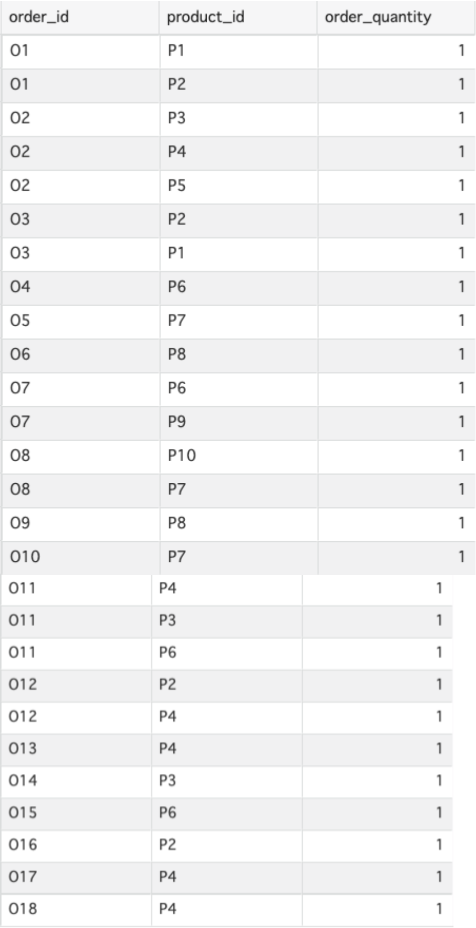
4. Script to create Order line table:



```
CREATE TABLE `order_line` (
  `ORDER_ID` varchar(45) NOT NULL,
  `PRODUCT_ID` varchar(45) NOT NULL,
  `ORDER_QUANTITY` int(11) NOT NULL,
  PRIMARY KEY (`ORDER_ID`),
  KEY `product_id_idx` (`PRODUCT_ID`),
  CONSTRAINT `product_id` FOREIGN KEY (`PRODUCT_ID`) REFERENCES `product_table` (`PRODUCT_ID`)
);
```

The above script creates an order line table with columns like Order_ID, Product_ID and Order_quantity. Unlike other tables, this tables uses two columns to uniquely identify each record. It is also known as a **Composite key**.

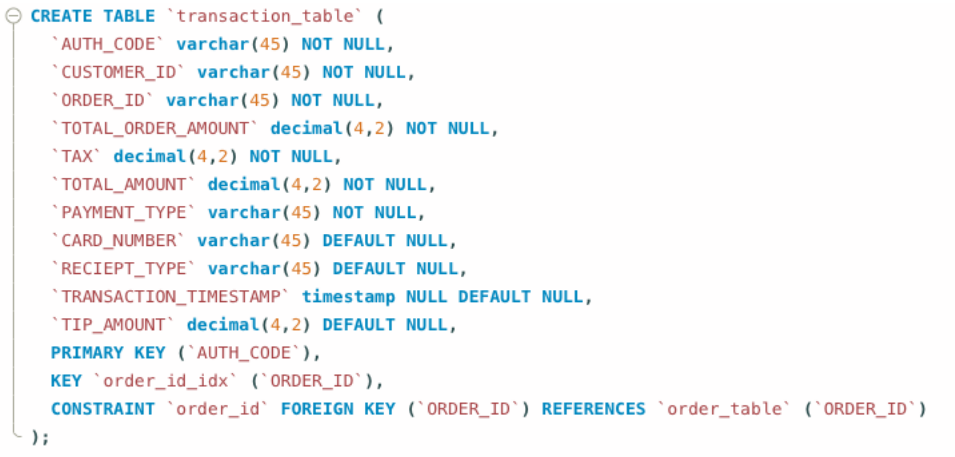
Output: Order line table



order_id	product_id	order_quantity
O1	P1	1
O1	P2	1
O2	P3	1
O2	P4	1
O2	P5	1
O3	P2	1
O3	P1	1
O4	P6	1
O5	P7	1
O6	P8	1
O7	P6	1
O7	P9	1
O8	P10	1
O8	P7	1
O9	P8	1
O10	P7	1
O11	P4	1
O11	P3	1
O11	P6	1
O12	P2	1
O12	P4	1
O13	P4	1
O14	P3	1
O15	P6	1
O16	P2	1
O17	P4	1
O18	P4	1

Each customer can have one order but multiple products in each order. This tables keeps track of all the products that are contained within each order.

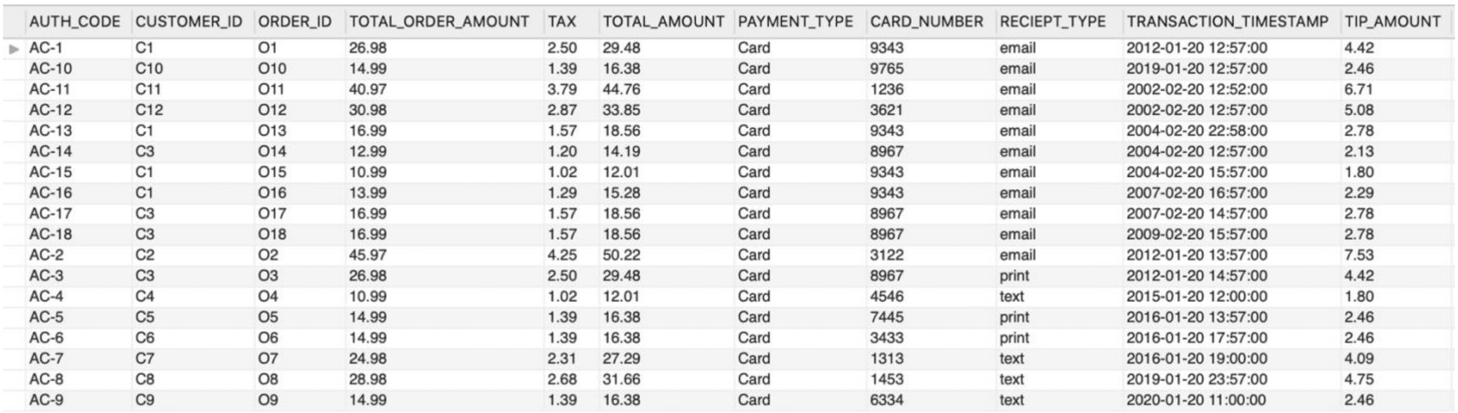
5. Script to create Transaction table:



```
CREATE TABLE `transaction_table` (
  `AUTH_CODE` varchar(45) NOT NULL,
  `CUSTOMER_ID` varchar(45) NOT NULL,
  `ORDER_ID` varchar(45) NOT NULL,
  `TOTAL_ORDER_AMOUNT` decimal(4,2) NOT NULL,
  `TAX` decimal(4,2) NOT NULL,
  `TOTAL_AMOUNT` decimal(4,2) NOT NULL,
  `PAYMENT_TYPE` varchar(45) NOT NULL,
  `CARD_NUMBER` varchar(45) DEFAULT NULL,
  `RECIEPT_TYPE` varchar(45) DEFAULT NULL,
  `TRANSACTION_TIMESTAMP` timestamp NULL DEFAULT NULL,
  `TIP_AMOUNT` decimal(4,2) DEFAULT NULL,
  PRIMARY KEY (`AUTH_CODE`),
  KEY `order_id_idx` (`ORDER_ID`),
  CONSTRAINT `order_id` FOREIGN KEY (`ORDER_ID`) REFERENCES `order_table` (`ORDER_ID`)
);
```

This script creates a Transaction table with fields like Auth_code, Customer_id, order_id and other payment details. Here, the primary key is defined by the column **Auth_code** where each record represents one transaction.

Output: Transaction table



AUTH_CODE	CUSTOMER_ID	ORDER_ID	TOTAL_ORDER_AMOUNT	TAX	TOTAL_AMOUNT	PAYMENT_TYPE	CARD_NUMBER	RECIEPT_TYPE	TRANSACTION_TIMESTAMP	TIP_AMOUNT
AC-1	C1	O1	26.98	2.50	29.48	Card	9343	email	2012-01-20 12:57:00	4.42
AC-10	C10	O10	14.99	1.39	16.38	Card	9765	email	2019-01-20 12:57:00	2.46
AC-11	C11	O11	40.97	3.79	44.76	Card	1236	email	2002-02-20 12:52:00	6.71
AC-12	C12	O12	30.98	2.87	33.85	Card	3621	email	2002-02-20 12:57:00	5.08
AC-13	C1	O13	16.99	1.57	18.56	Card	9343	email	2004-02-20 22:58:00	2.78
AC-14	C3	O14	12.99	1.20	14.19	Card	8967	email	2004-02-20 12:57:00	2.13
AC-15	C1	O15	10.99	1.02	12.01	Card	9343	email	2004-02-20 15:57:00	1.80
AC-16	C1	O16	13.99	1.29	15.28	Card	9343	email	2007-02-20 16:57:00	2.29
AC-17	C3	O17	16.99	1.57	18.56	Card	8967	email	2007-02-20 14:57:00	2.78
AC-18	C3	O18	16.99	1.57	18.56	Card	8967	email	2009-02-20 15:57:00	2.78
AC-2	C2	O2	45.97	4.25	50.22	Card	3122	email	2012-01-20 13:57:00	7.53
AC-3	C3	O3	26.98	2.50	29.48	Card	8967	print	2012-01-20 14:57:00	4.42
AC-4	C4	O4	10.99	1.02	12.01	Card	4546	text	2015-01-20 12:00:00	1.80
AC-5	C5	O5	14.99	1.39	16.38	Card	7445	print	2016-01-20 13:57:00	2.46
AC-6	C6	O6	14.99	1.39	16.38	Card	3433	print	2016-01-20 17:57:00	2.46
AC-7	C7	O7	24.98	2.31	27.29	Card	1313	text	2016-01-20 19:00:00	4.09
AC-8	C8	O8	28.98	2.68	31.66	Card	1453	text	2019-01-20 23:57:00	4.75
AC-9	C9	O9	14.99	1.39	16.38	Card	6334	text	2020-01-20 11:00:00	2.46

This table has the details about each transaction by the cashier. It links the customer Id to the order Id. Each entry here indicates that a customer has already paid for a corresponding order.

SQL Statement Creation

Now that we have our tables set up, let's write a few SQL select statements to retrieve some information. A basic SQL select statement without joins can be used to retrieve any combination of rows and columns from a single table.

Basic SQL SELECT syntax:

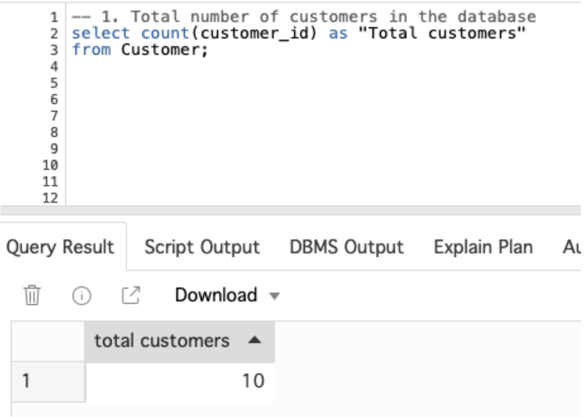
SELECT *column1*, *column2, ...*
FROM *table_name*;

The attributes in **SELECT** statement always represents the columns you want to display in your output.

Depending on the kind of data you want to get, you can modify the syntax with conditions accordingly. For example, If you want to get the total number of orders in a the database, you can use an aggregate function called **COUNT**.

Let's look at some simple SQL queries.

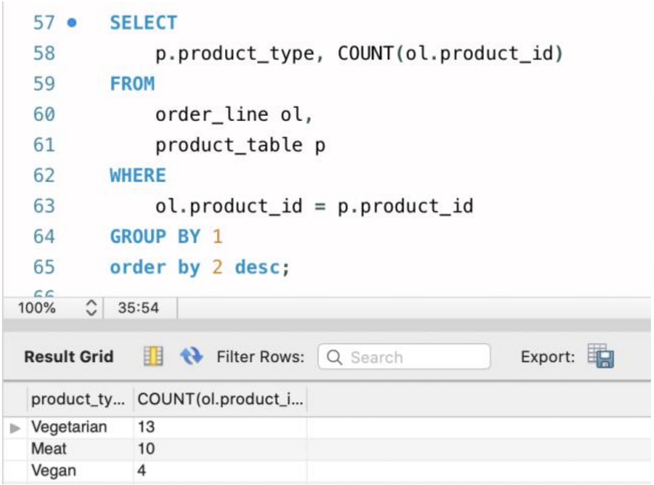
- If you want to know how many customers have visited this place till date, you can use a simple select query as below.



```
-- 1. Total number of customers in the database
select count(customer_id) as "Total customers"
from Customer;
```

	total customers ▲
1	10

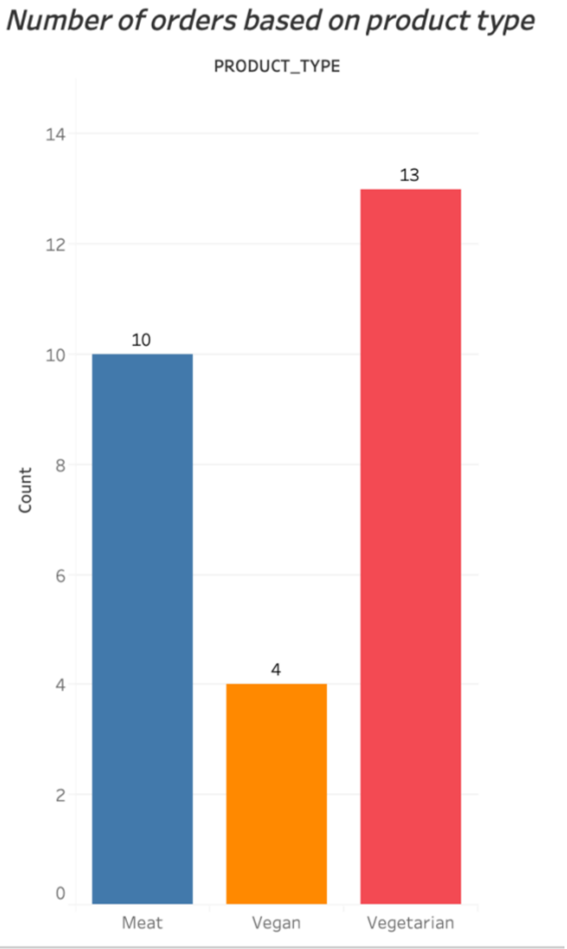
- The table below is to know the number of meat, vegetarian and vegan orders for this vendor.



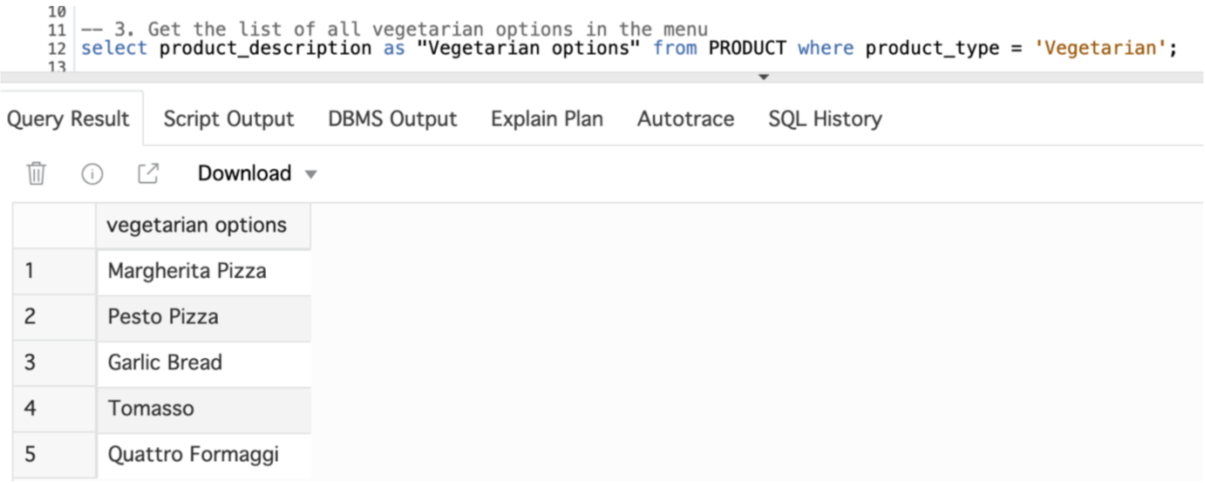
```
SELECT
    p.product_type, COUNT(ol.product_id)
FROM
    order_line ol,
    product_table p
WHERE
    ol.product_id = p.product_id
GROUP BY 1
order by 2 desc;
```

product_ty...	COUNT(ol.product_i...
Vegetarian	13
Meat	10
Vegan	4

Tableau view



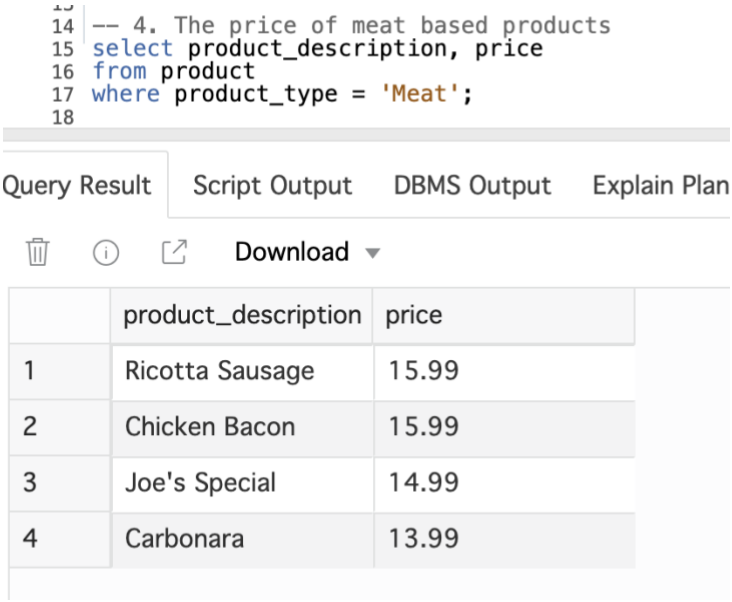
- You can run the query below to get all the vegetarian options in the menu.



```sql
-- 3. Get the list of all vegetarian options in the menu
select product_description as "Vegetarian options" from PRODUCT where product_type = 'Vegetarian';
```

	vegetarian options
1	Margherita Pizza
2	Pesto Pizza
3	Garlic Bread
4	Tomasso
5	Quattro Formaggi

- This query fetches the names of all the meat-based items on the menu along with their price.

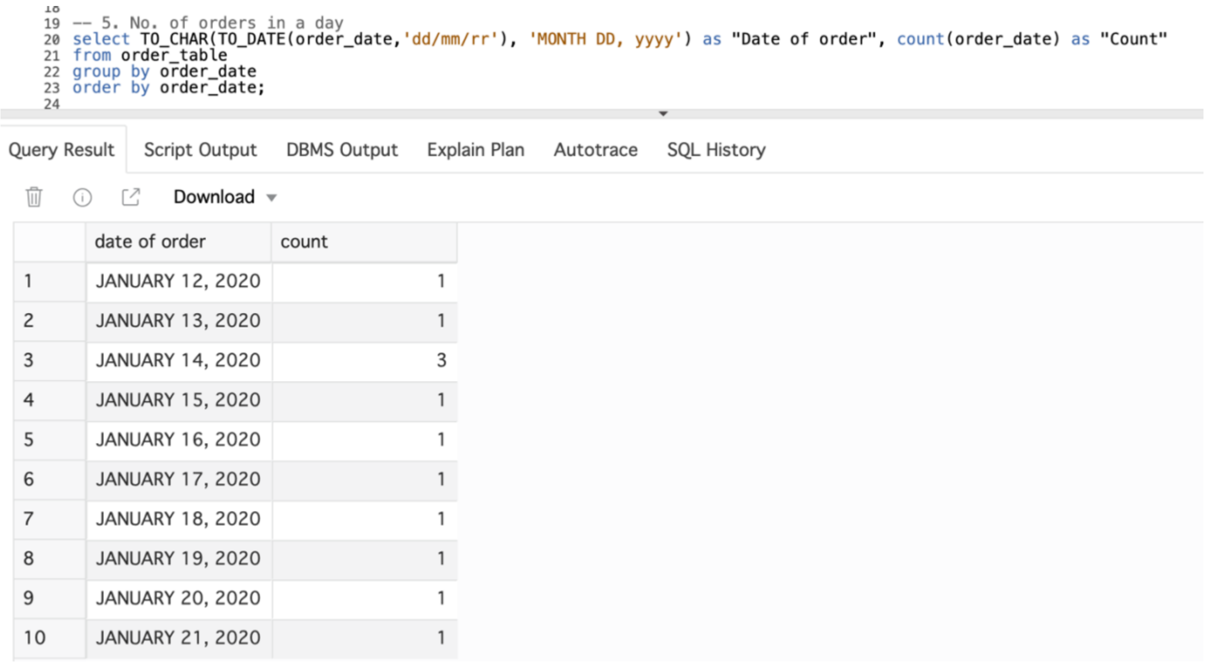


```sql
-- 4. The price of meat based products
select product_description, price
from product
where product_type = 'Meat';
```

	product_description	price
1	Ricotta Sausage	15.99
2	Chicken Bacon	15.99
3	Joe's Special	14.99
4	Carbonara	13.99

- With the help of this query, you can get a snapshot of the number of orders this vendor gets in the date range specified. Similarly, by tweaking the query slightly, we can get a snapshot of the order count per week, month or year.

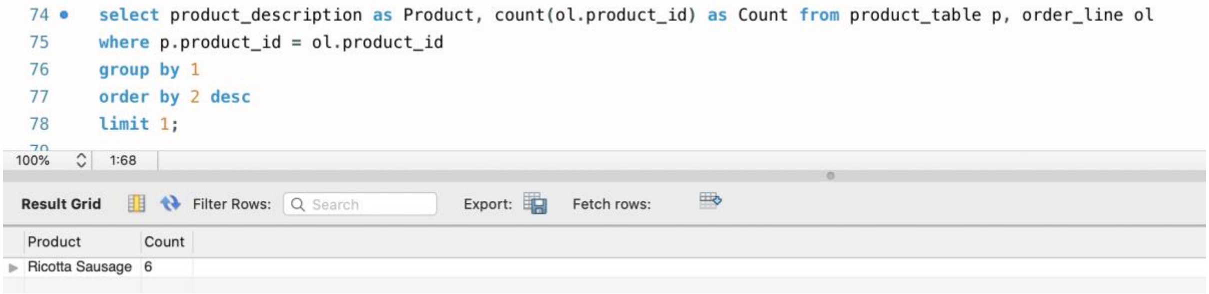
  This kind of information can be useful from a business point of view to get an idea of what are the busiest days/weeks/months etc.



```
-- 5. No. of orders in a day
select TO_CHAR(TO_DATE(order_date,'dd/mm/rr'), 'MONTH DD, yyyy') as "Date of order", count(order_date) as "Count"
from order_table
group by order_date
order by order_date;
```

	date of order	count
1	JANUARY 12, 2020	1
2	JANUARY 13, 2020	1
3	JANUARY 14, 2020	3
4	JANUARY 15, 2020	1
5	JANUARY 16, 2020	1
6	JANUARY 17, 2020	1
7	JANUARY 18, 2020	1
8	JANUARY 19, 2020	1
9	JANUARY 20, 2020	1
10	JANUARY 21, 2020	1

- This query shows the most selling item in the store. This information could indicate that “Ricotta Sausage” must really be this vendor’s specialty.

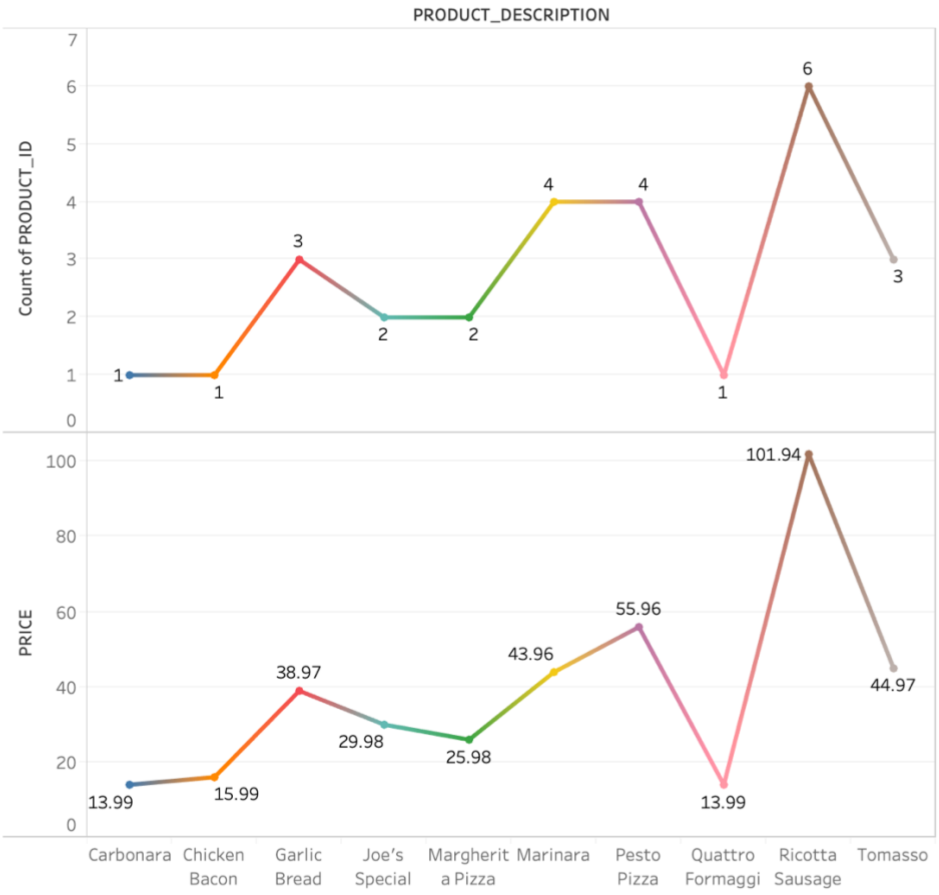


```
select product_description as Product, count(ol.product_id) as Count from product_table p, order_line ol
where p.product_id = ol.product_id
group by 1
order by 2 desc
limit 1;
```

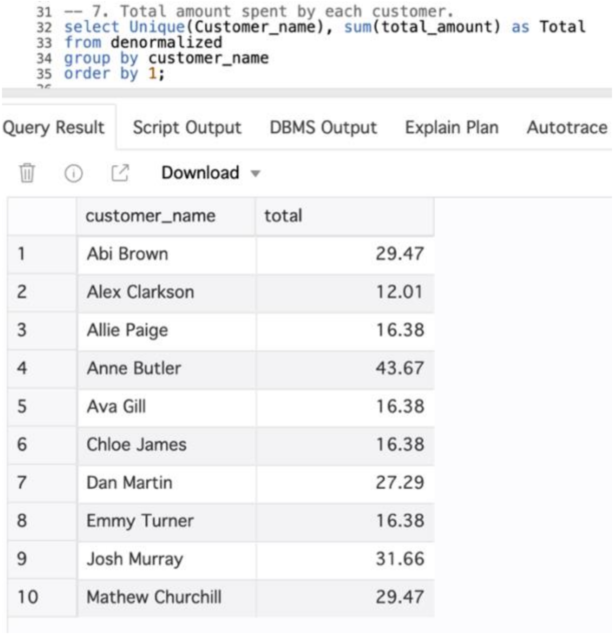
Product	Count
Ricotta Sausage	6

Tableau view:

What is the most ordered item on the menu?



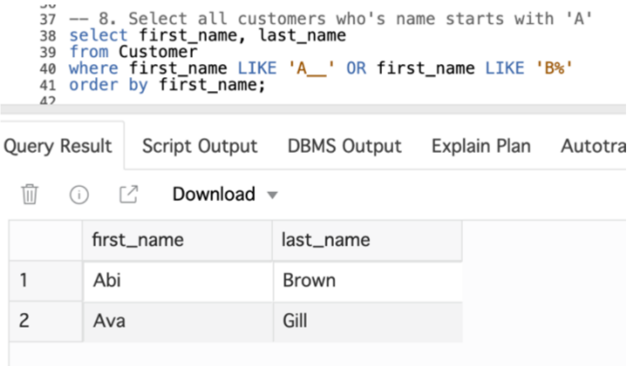
- What is the total amount spent by each customer?



```
-- 7. Total amount spent by each customer.
select Unique(Customer_name), sum(total_amount) as Total
from denormalized
group by customer_name
order by 1;
```

	customer_name	total
1	Abi Brown	29.47
2	Alex Clarkson	12.01
3	Allie Paige	16.38
4	Anne Butler	43.67
5	Ava Gill	16.38
6	Chloe James	16.38
7	Dan Martin	27.29
8	Emmy Turner	16.38
9	Josh Murray	31.66
10	Mathew Churchill	29.47

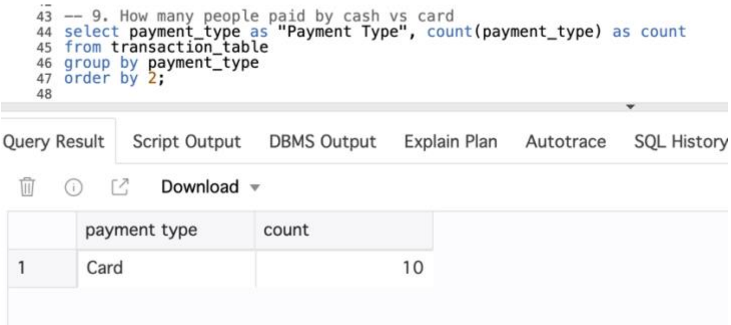
- Select all customers whose name starts with ‘A’



```
-- 8. Select all customers who's name starts with 'A'
select first_name, last_name
from Customer
where first_name LIKE 'A__' OR first_name LIKE 'B%'
order by first_name;
```

	first_name	last_name
1	Abi	Brown
2	Ava	Gill

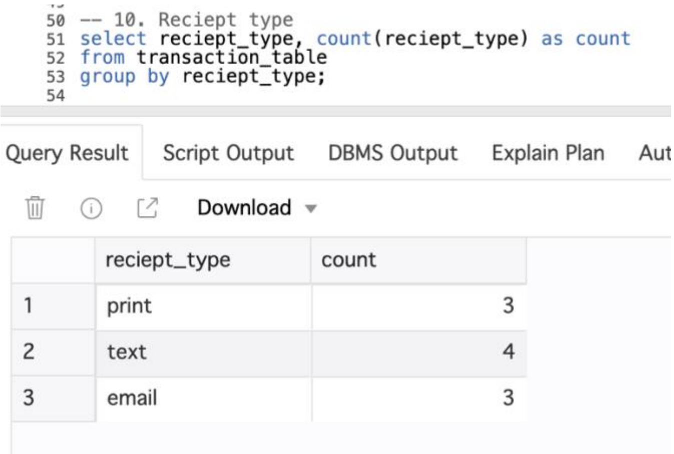
- How many people paid by card/cash?



```
-- 9. How many people paid by cash vs card
select payment_type as "Payment Type", count(payment_type) as count
from transaction_table
group by payment_type
order by 2;
```

	payment type	count
1	Card	10

- Display the receipt type



```
-- 10. Reciept type
select reciept_type, count(reciept_type) as count
from transaction_table
group by reciept_type;
```

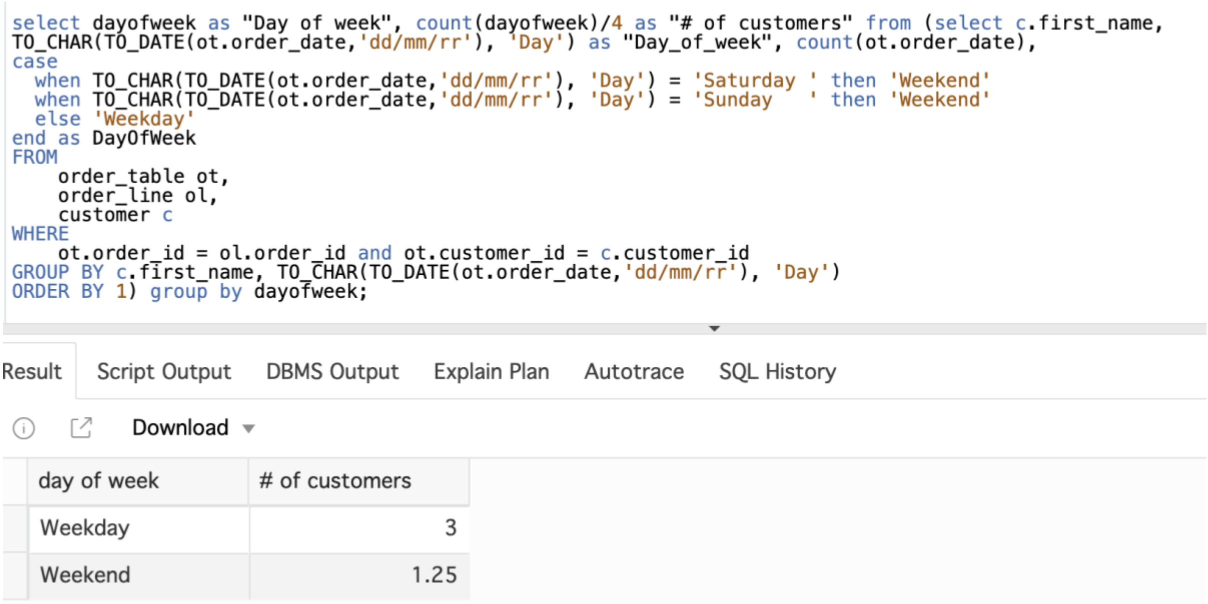
	reciept_type	count
1	print	3
2	text	4
3	email	3

Advanced SQL

Using SQL, we can write complex queries to retrieve meaningful information from multiple tables into a single output. This can be achieved with the help of joins and subqueries. Such information can be used to answer several business questions.

Let’s look at a few examples.

Example 1: Information like the number of customers visiting the establishment on weekdays vs weekends can be useful to the store manager since it can aid in planning for demand by procuring more material, hiring additional part time help etc.

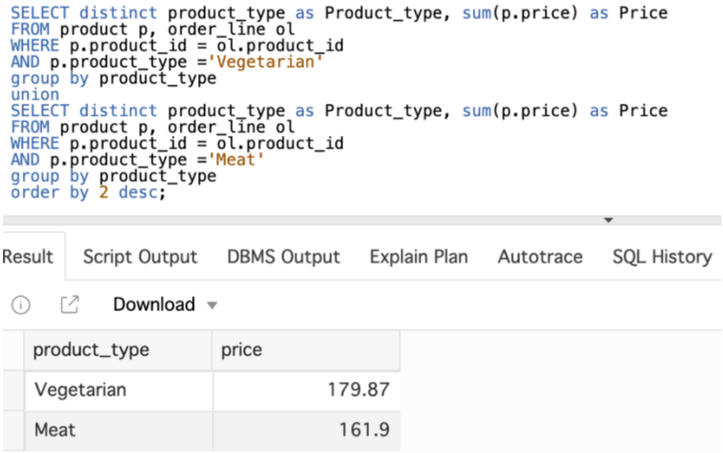


```
select dayofweek as "Day of week", count(dayofweek)/4 as "# of customers" from (select c.first_name,
TO_CHAR(TO_DATE(ot.order_date,'dd/mm/rr'), 'Day') as "Day_of_week", count(ot.order_date),
case
  when TO_CHAR(TO_DATE(ot.order_date,'dd/mm/rr'), 'Day') = 'Saturday ' then 'Weekend'
  when TO_CHAR(TO_DATE(ot.order_date,'dd/mm/rr'), 'Day') = 'Sunday   ' then 'Weekend'
  else 'Weekday'
end as DayOfWeek
FROM
    order_table ot,
    order_line ol,
    customer c
WHERE
    ot.order_id = ol.order_id and ot.customer_id = c.customer_id
GROUP BY c.first_name, TO_CHAR(TO_DATE(ot.order_date,'dd/mm/rr'), 'Day')
ORDER BY 1) group by dayofweek;
```

day of week	# of customers
Weekday	3
Weekend	1.25

From the results, we can see that on an average, Pizza Bocca Lupo gets comparatively more customers on weekdays compared to weekends. Such information can be useful for the store manager/CEO who is looking to maximize their sales. For example, depending on the demand, they can adjust their timings on those days when the inflow of customers is more.

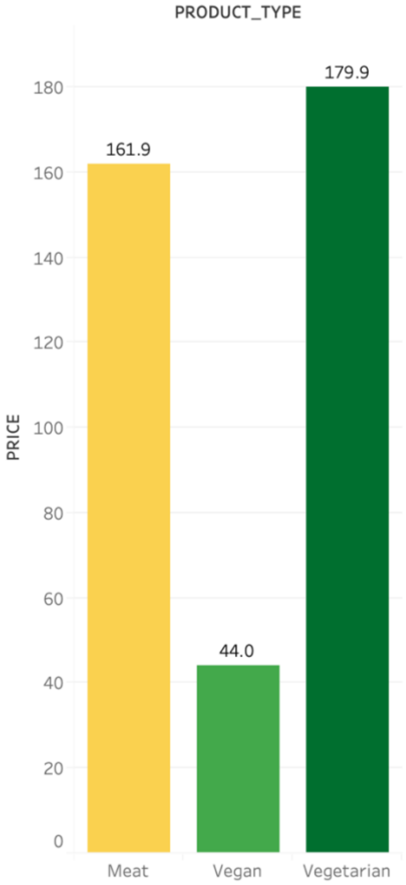
Example 2: Suppose, we want to see which category contributes majorly to their revenue, we can write a query like:



```
SELECT distinct product_type as Product_type, sum(p.price) as Price
FROM product p, order_line ol
WHERE p.product_id = ol.product_id
AND p.product_type ='Vegetarian'
group by product_type
union
SELECT distinct product_type as Product_type, sum(p.price) as Price
FROM product p, order_line ol
WHERE p.product_id = ol.product_id
AND p.product_type ='Meat'
group by product_type
order by 2 desc;
```

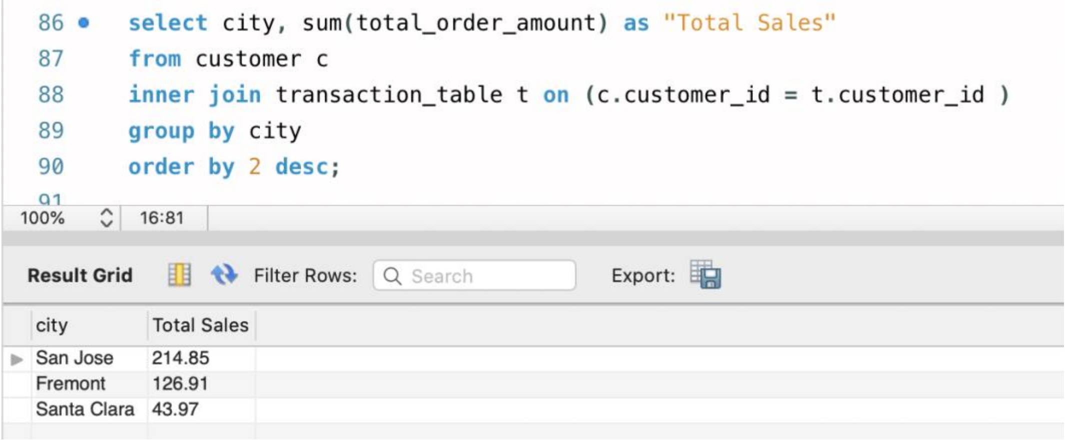
product_type	price
Vegetarian	179.87
Meat	161.9

Tableau view:



From the results we see that, the sales of vegetarian options are slightly more than the corresponding sales for the meat options. Such insights can be used when expanding the menu offerings based on the most popular categories.

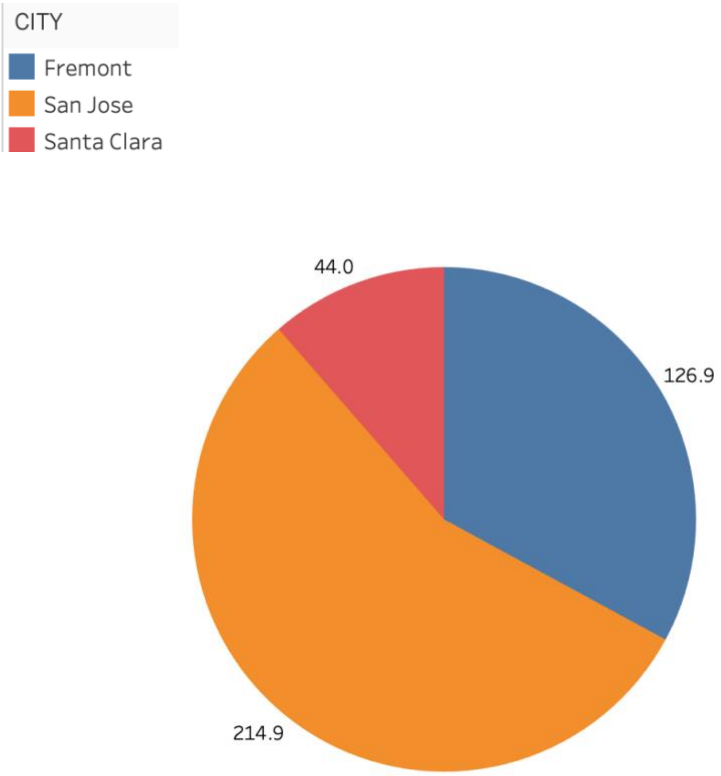
Example 3: Among the neighboring areas, to find the city from where most customers are residents, we can use the following query:



```
select city, sum(total_order_amount) as "Total Sales"
from customer c
inner join transaction_table t on (c.customer_id = t.customer_id )
group by city
order by 2 desc;
```

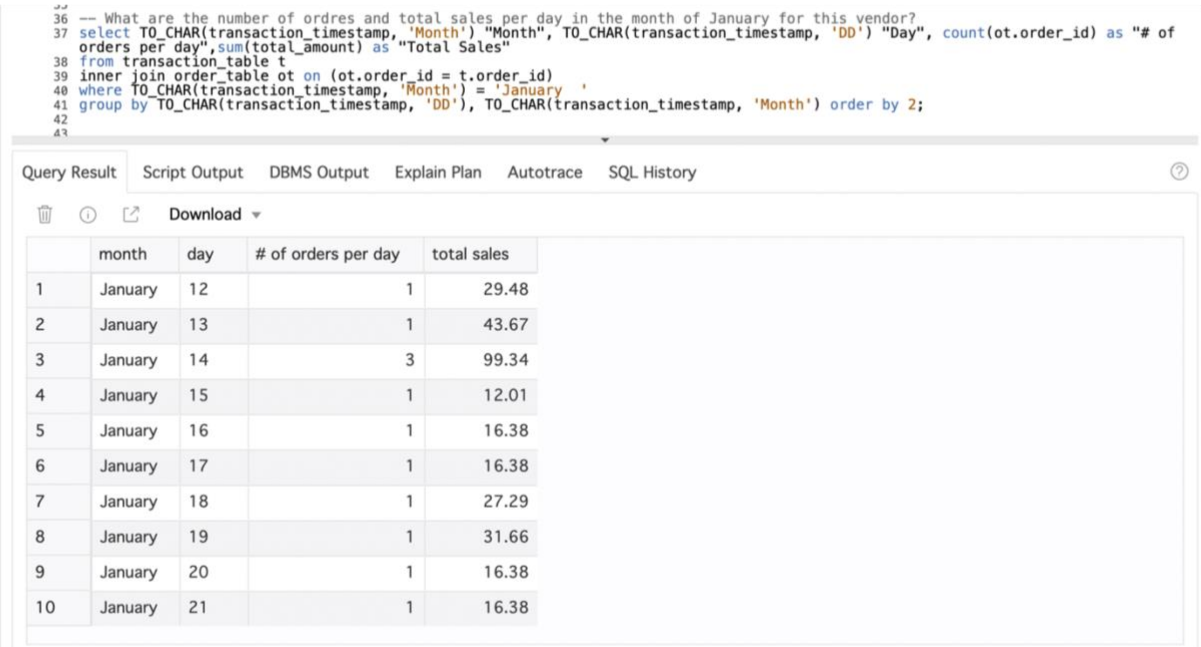
city	Total Sales
San Jose	214.85
Fremont	126.91
Santa Clara	43.97

Tableau view:



This output helps us understand the sales breakup per city. This kind of a breakup will give us an idea about where are the customers come from. For example, if there are a lot of customers/sales from Fremont, the vendor might consider opening up a branch in that area.

Example 4: This query is to find out the number of sales and the revenue in a given month.

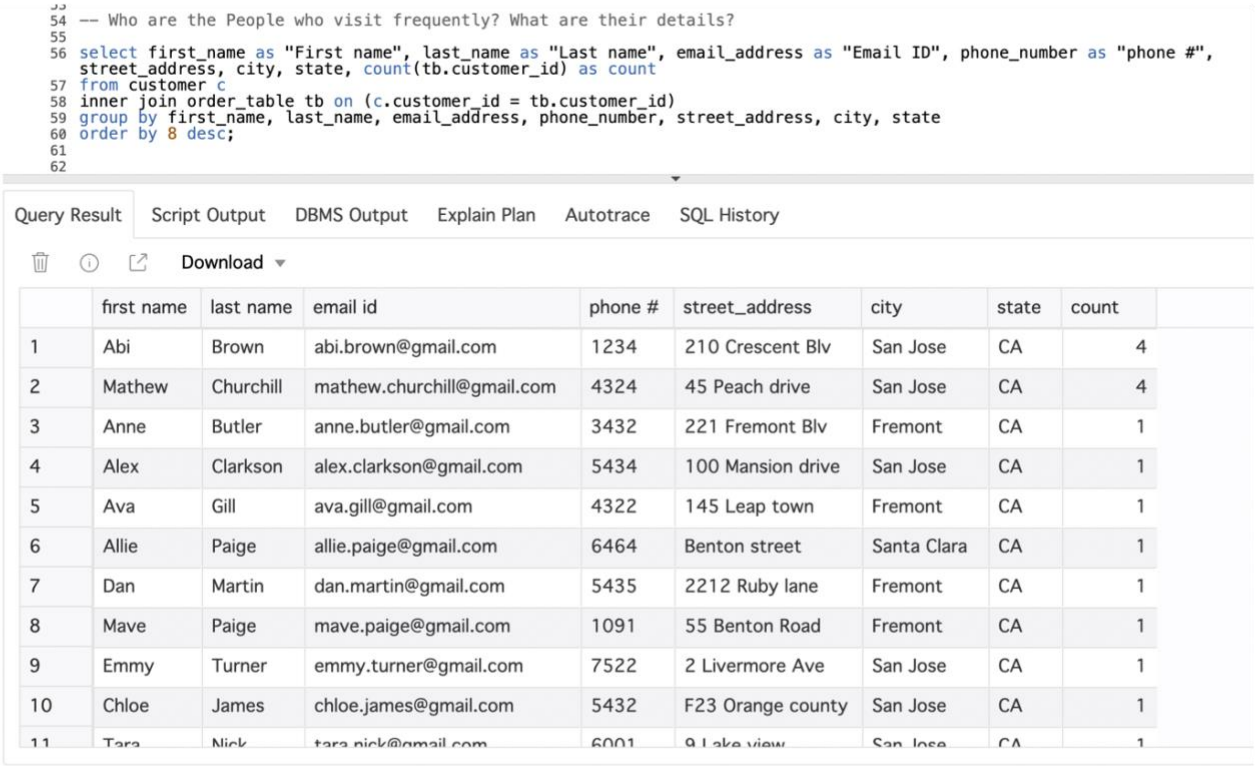


```
-- What are the number of ordres and total sales per day in the month of January for this vendor?
select TO_CHAR(transaction_timestamp, 'Month') "Month", TO_CHAR(transaction_timestamp, 'DD') "Day", count(ot.order_id) as "# of orders per day",sum(total_amount) as "Total Sales"
from transaction_table t
inner join order_table ot on (ot.order_id = t.order_id)
where TO_CHAR(transaction_timestamp, 'Month') = 'January  '
group by TO_CHAR(transaction_timestamp, 'DD'), TO_CHAR(transaction_timestamp, 'Month') order by 2;
```

	month	day	# of orders per day	total sales
1	January	12	1	29.48
2	January	13	1	43.67
3	January	14	3	99.34
4	January	15	1	12.01
5	January	16	1	16.38
6	January	17	1	16.38
7	January	18	1	27.29
8	January	19	1	31.66
9	January	20	1	16.38
10	January	21	1	16.38

From Jan 12 – Jan 21, there have been a total of 12 orders. We can also see the sales per day in the month of January. This helps us understand which days of the month had the highest sales.

Example 5: To get the list customers and how often they visit this place, we can run the following query:



```
-- Who are the People who visit frequently? What are their details?

select first_name as "First name", last_name as "Last name", email_address as "Email ID", phone_number as "phone #",
street_address, city, state, count(tb.customer_id) as count
from customer c
inner join order_table tb on (c.customer_id = tb.customer_id)
group by first_name, last_name, email_address, phone_number, street_address, city, state
order by 8 desc;
```

	first name	last name	email id	phone #	street_address	city	state	count
1	Abi	Brown	abi.brown@gmail.com	1234	210 Crescent Blv	San Jose	CA	4
2	Mathew	Churchill	mathew.churchill@gmail.com	4324	45 Peach drive	San Jose	CA	4
3	Anne	Butler	anne.butler@gmail.com	3432	221 Fremont Blv	Fremont	CA	1
4	Alex	Clarkson	alex.clarkson@gmail.com	5434	100 Mansion drive	San Jose	CA	1
5	Ava	Gill	ava.gill@gmail.com	4322	145 Leap town	Fremont	CA	1
6	Allie	Paige	allie.paige@gmail.com	6464	Benton street	Santa Clara	CA	1
7	Dan	Martin	dan.martin@gmail.com	5435	2212 Ruby lane	Fremont	CA	1
8	Mave	Paige	mave.paige@gmail.com	1091	55 Benton Road	Fremont	CA	1
9	Emmy	Turner	emmy.turner@gmail.com	7522	2 Livermore Ave	San Jose	CA	1
10	Chloe	James	chloe.james@gmail.com	5432	F23 Orange county	San Jose	CA	1
11	Tara	Nick	tara.nick@gmail.com	6001	9 Lake view	San Jose	CA	1

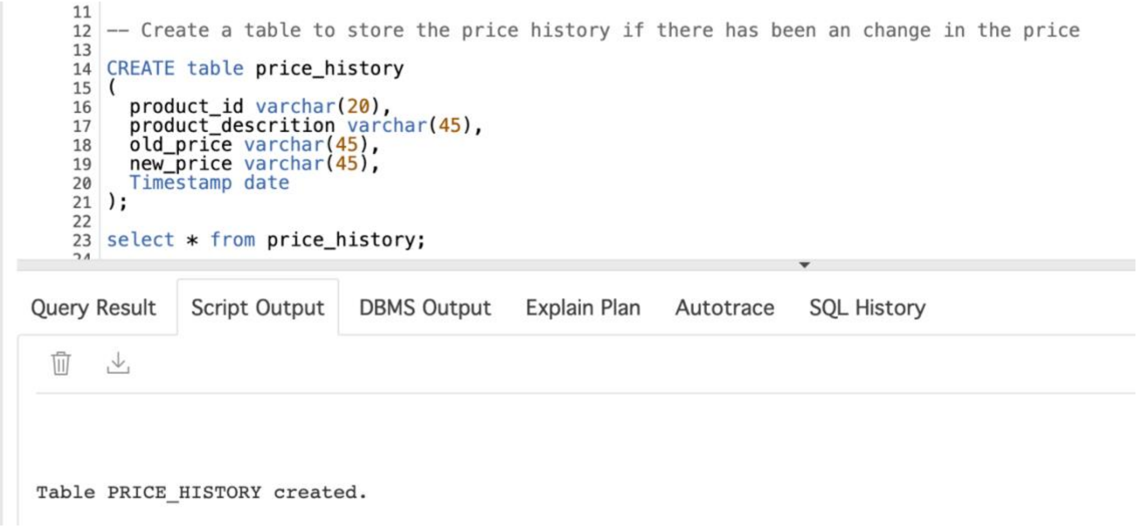
This helps us get an idea of the recurring customers. Such recurring customers can be targeted to send coupons and promotional emails/letters via their phone numbers, address and email Ids.

Triggers

Since prices on the menu always keep changing with time, it is a good practice to record those changes overtime. This trigger helps you keep track of the price history of the items on the menu. The `sysdate' column helps identify the date when the change happened.

This trigger updates the original table as well as stores the updated values along with old values in another table.

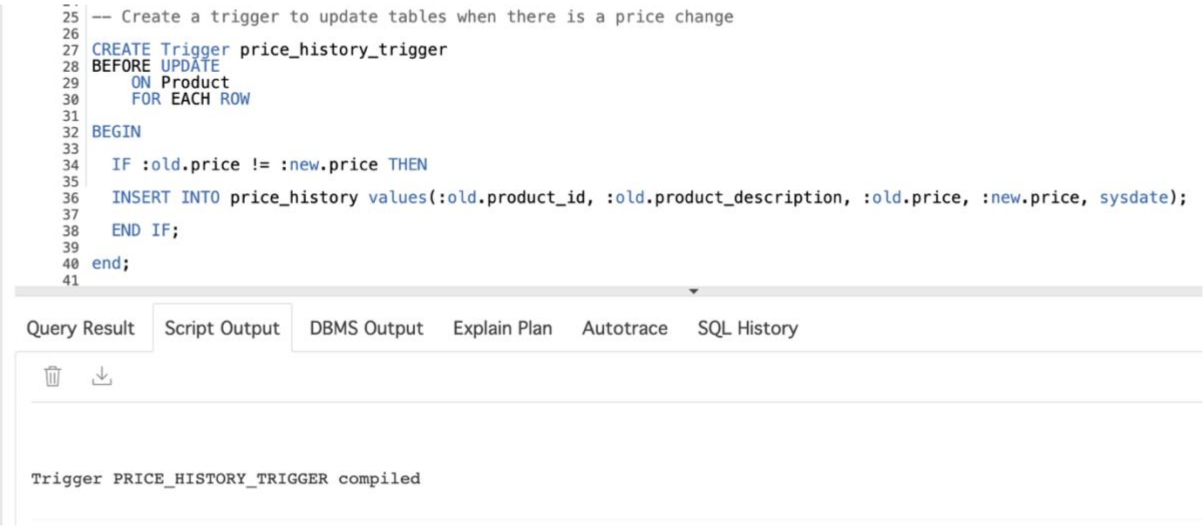
First, let's create a table to store all the old and new prices.



```
-- Create a table to store the price history if there has been an change in the price

CREATE table price_history
(
  product_id varchar(20),
  product_descrition varchar(45),
  old_price varchar(45),
  new_price varchar(45),
  Timestamp date
);

select * from price_history;
```

```
Table PRICE_HISTORY created.
```

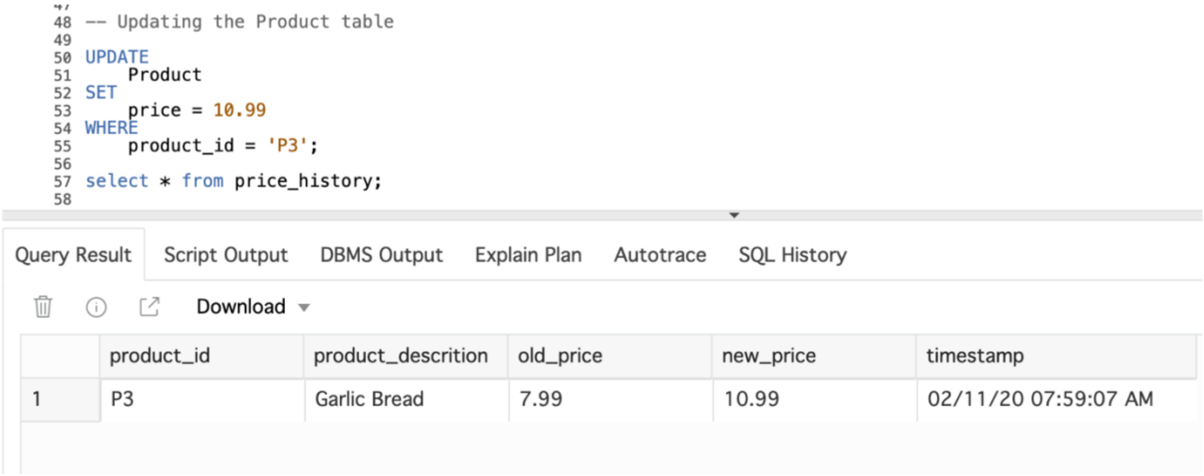
Now creating a trigger called **price_history_trigger**.



```
-- Create a trigger to update tables when there is a price change

CREATE Trigger price_history_trigger
BEFORE UPDATE
    ON Product
    FOR EACH ROW

BEGIN

  IF :old.price != :new.price THEN

  INSERT INTO price_history values(:old.product_id, :old.product_description, :old.price, :new.price, sysdate);

  END IF;

end;
```

```
Trigger PRICE_HISTORY_TRIGGER compiled
```

To test this trigger, let’s try updating the product table.



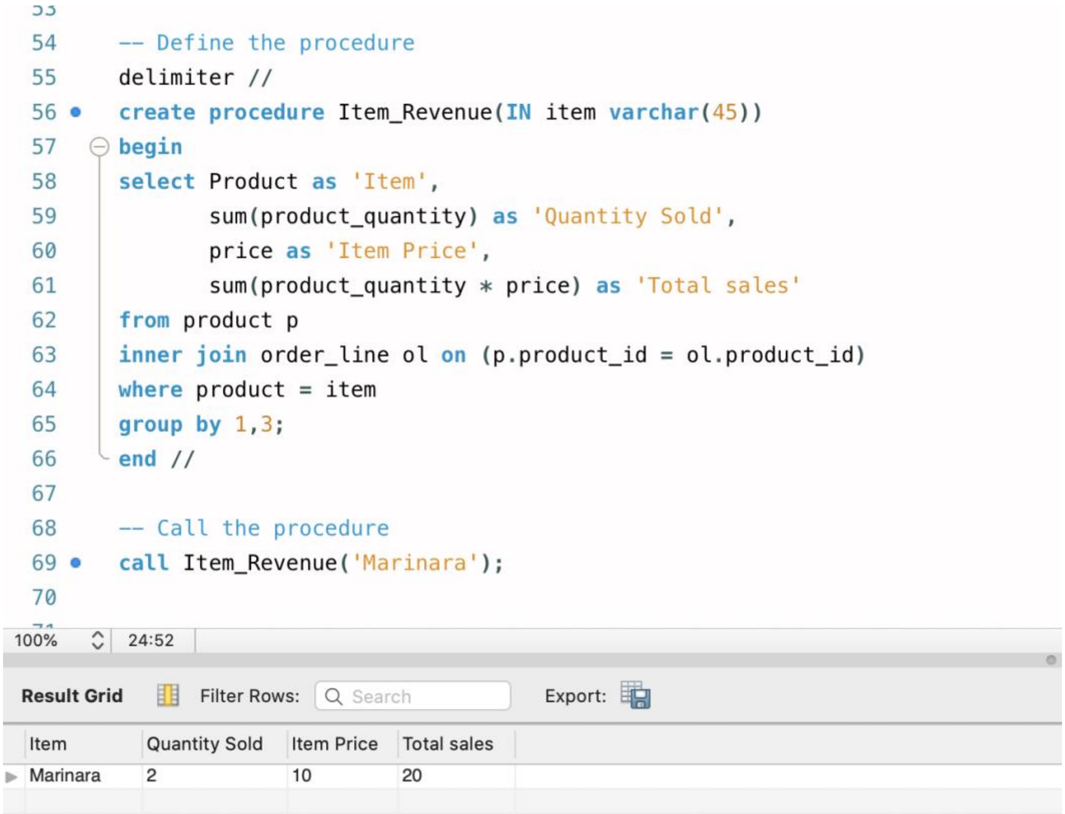
```
-- Updating the Product table

UPDATE
    Product
SET
    price = 10.99
WHERE
    product_id = 'P3';

select * from price_history;
```

	product_id	product_descrition	old_price	new_price	timestamp
1	P3	Garlic Bread	7.99	10.99	02/11/20 07:59:07 AM

Above, the old price and updated price is stored in the ‘price_history’ table.

Procedure:

If you quickly want to see how much revenue us being generated by an item on the menu, you can run the below procedure:



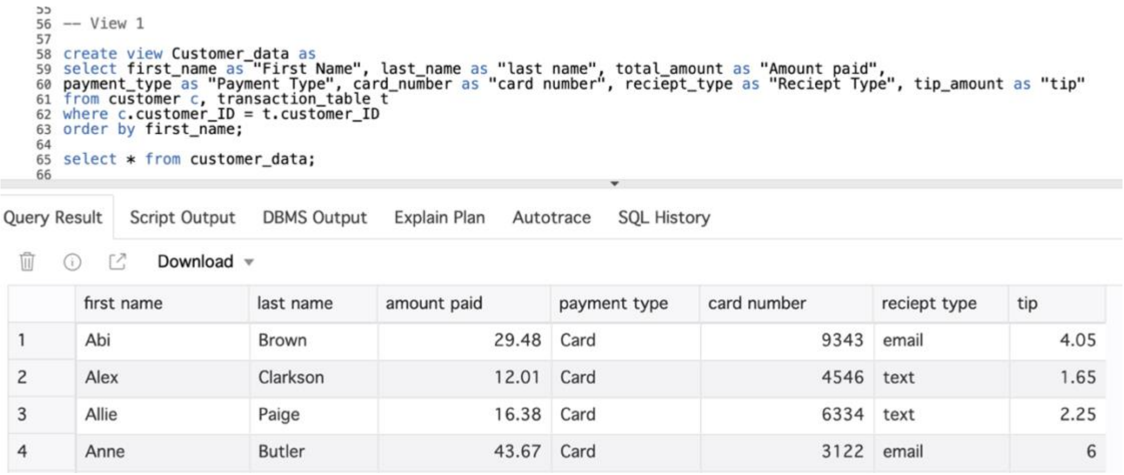
```
-- Define the procedure
delimiter //
create procedure Item_Revenue(IN item varchar(45))
begin
select Product as 'Item',
       sum(product_quantity) as 'Quantity Sold',
       price as 'Item Price',
       sum(product_quantity * price) as 'Total sales'
from product p
inner join order_line ol on (p.product_id = ol.product_id)
where product = item
group by 1,3;
end //

-- Call the procedure
call Item_Revenue('Marinara');
```

Item	Quantity Sold	Item Price	Total sales
Marinara	2	10	20

Views

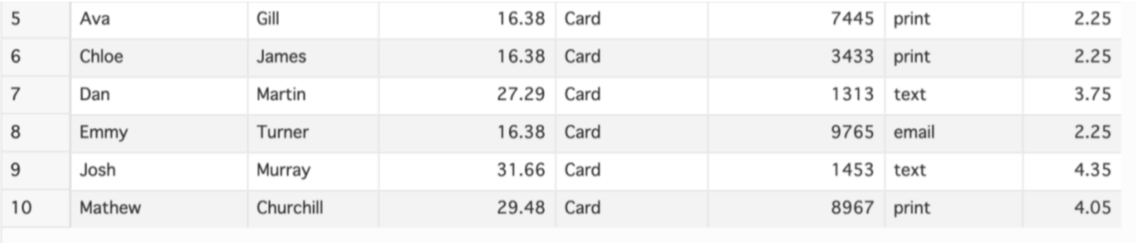
A view is a table whose contents are defined by a query. One if its advantages is that every user can be given access to the database only through a small set of views. This can only authorize the user to see the data, thus restricting the user's access to stored data.
Another advantage is that, a view can collate data from various tables and present it as a single table.

For example, in table below, a view was with the data from 'customer table' and 'transaction table'.



```
-- View 1

create view Customer_data as
select first_name as "First Name", last_name as "last name", total_amount as "Amount paid",
payment_type as "Payment Type", card_number as "card number", reciept_type as "Reciept Type", tip_amount as "tip"
from customer c, transaction_table t
where c.customer_ID = t.customer_ID
order by first_name;

select * from customer_data;
```

	first name	last name	amount paid	payment type	card number	reciept type	tip
1	Abi	Brown	29.48	Card	9343	email	4.05
2	Alex	Clarkson	12.01	Card	4546	text	1.65
3	Allie	Paige	16.38	Card	6334	text	2.25
4	Anne	Butler	43.67	Card	3122	email	6

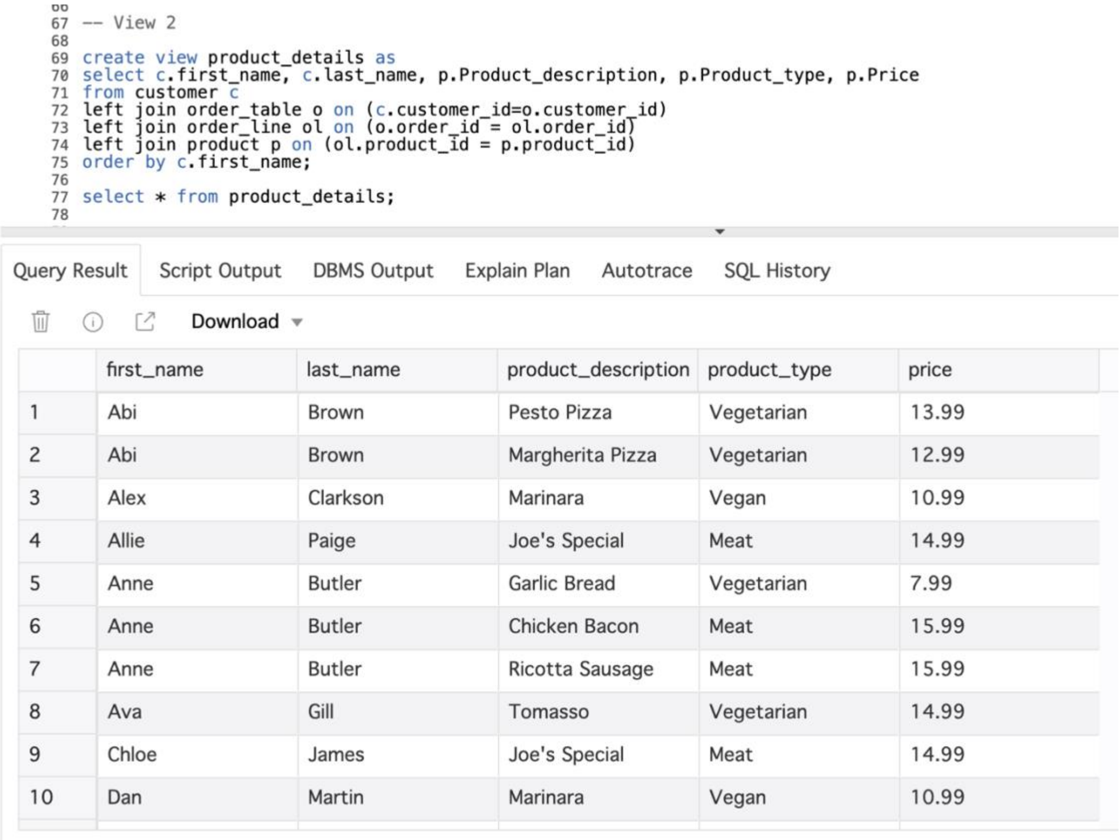


5	Ava	Gill	16.38	Card	7445	print	2.25
6	Chloe	James	16.38	Card	3433	print	2.25
7	Dan	Martin	27.29	Card	1313	text	3.75
8	Emmy	Turner	16.38	Card	9765	email	2.25
9	Josh	Murray	31.66	Card	1453	text	4.35
10	Mathew	Churchill	29.48	Card	8967	print	4.05

View 1.

Earlier in the Transaction table, a customer was identified through their customer ID only. But, creating a view like this allows you to see the names of the customers and their payment details without having write a complex query.

Similar to View 1, this view is created with data from customer and the product table.



```
-- View 2

create view product_details as
select c.first_name, c.last_name, p.Product_description, p.Product_type, p.Price
from customer c
left join order_table o on (c.customer_id=o.customer_id)
left join order_line ol on (o.order_id = ol.order_id)
left join product p on (ol.product_id = p.product_id)
order by c.first_name;

select * from product_details;
```

	first_name	last_name	product_description	product_type	price
1	Abi	Brown	Pesto Pizza	Vegetarian	13.99
2	Abi	Brown	Margherita Pizza	Vegetarian	12.99
3	Alex	Clarkson	Marinara	Vegan	10.99
4	Allie	Paige	Joe's Special	Meat	14.99
5	Anne	Butler	Garlic Bread	Vegetarian	7.99
6	Anne	Butler	Chicken Bacon	Meat	15.99
7	Anne	Butler	Ricotta Sausage	Meat	15.99
8	Ava	Gill	Tomasso	Vegetarian	14.99
9	Chloe	James	Joe's Special	Meat	14.99
10	Dan	Martin	Marinara	Vegan	10.99

View 2.

Conclusion

We can make this data more useful to the business operations if we had additional data related to employees.
For example:

- Information like "How many customers can the business serve at once?" can be derived with the help of the number of employees working at any given time.
- What is the average time a customer needs to wait for an order? – This can be answered by looking at the number of orders that are ahead of the current order and approximately how long each order takes.
- If we know what the cost of raw materials and other costs are, we will be able to derive how much profit they are making.

In conclusion, by analyzing the data of Pizza Bocca Lupo, we were able to get insight into their business practices.