PRODUCT SALES – DATABASE DESIGN

PROJECT PURPOSE: *To design a full database from the Product Sales Data of 2019.*

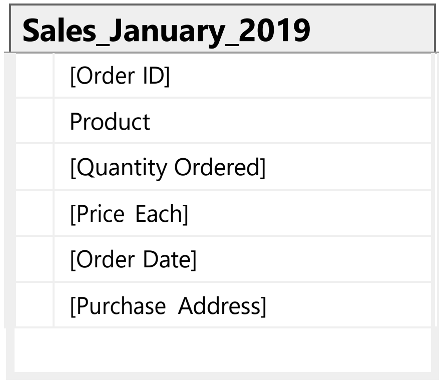
MISSION STATEMENT:

The purpose of the database is to maintain the data needed to support and analyze the company's retail sales and customer service operations.

Mission Objectives:

- Maintain record of all sales
- Maintain record of all orders
- Maintain record of all products
- Maintain record of all customers

Database Diagram of Sales_January_2019:



Sales_January_2019

[Order ID]
Product
[Quantity Ordered]
[Price Each]
[Order Date]
[Purchase Address]

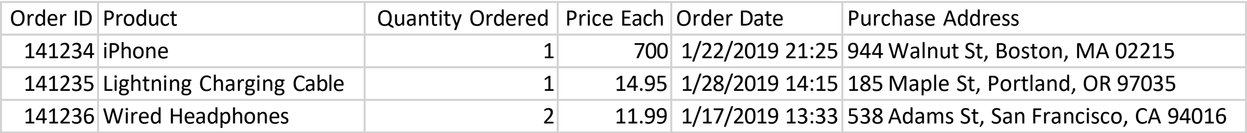
- *Above database diagram is the same for all other tables (imported csv's) (Sales_February_2019, Sales_March_2019, etc.).*

ANALYSIS OF CURRENT DATABASE/FIRST IMPRESSIONS:

Questions that need to be answered/addressed –

1. What types of data does the organization use?
2. How does the organization use that data?
3. How does the organization manage and maintain that data?

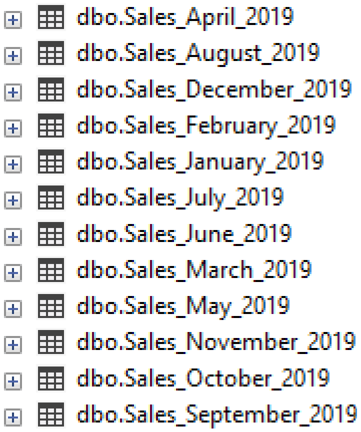
FIRST 3 ROWS OF COMPANY DATA FOR JANUARY SALES (2019):



Order ID	Product	Quantity Ordered	Price Each	Order Date	Purchase Address
141234	iPhone	1	700	1/22/2019 21:25	944 Walnut St, Boston, MA 02215
141235	Lightning Charging Cable	1	14.95	1/28/2019 14:15	185 Maple St, Portland, OR 97035
141236	Wired Headphones	2	11.99	1/17/2019 13:33	538 Adams St, San Francisco, CA 94016

- *The data maintained within these spreadsheets will need to be manipulated to create tables (sales, customers, products) in the new database.*

- *All data is separated into individual monthly csv's (Sales_January_2019, Sales_February_2019, etc.). All csv's will need to be combined into one file to properly manipulate and create tables. Possible paper-based or legacy database. Data entry through spreadsheet program.*



⊞ dbo.Sales_April_2019
⊞ dbo.Sales_August_2019
⊞ dbo.Sales_December_2019
⊞ dbo.Sales_February_2019
⊞ dbo.Sales_January_2019
⊞ dbo.Sales_July_2019
⊞ dbo.Sales_June_2019
⊞ dbo.Sales_March_2019
⊞ dbo.Sales_May_2019
⊞ dbo.Sales_November_2019
⊞ dbo.Sales_October_2019
⊞ dbo.Sales_September_2019

DATATYPE OF EACH OF THE COLUMNS:

```sql
SELECT COLUMN_NAME, DATA_TYPE FROM INFORMATION_SCHEMA.COLUMNS
WHERE TABLE_NAME = 'Sales_January_2019';
```

COLUMN_NAME	DATA_TYPE
Order ID	varchar
Product	varchar
Quantity Ordered	varchar
Price Each	varchar
Order Date	varchar
Purchase Address	varchar

Compiling a List of Fields -

List of Subjects (Table Names):

1. Sales
2. Customers
3. Products

Preliminary Field List (Core set of fields that will be defined in the database):

1. Sales ID
2. Customer ID
3. Product ID
4. Order ID
5. Purchase Date
6. Order Month
7. Order Day
8. Order Year
9. Quantity Ordered
10. Price Each
11. Address Name
12. City
13. State
14. Zip Code
15. Product
16. Unit Price

TABLE STRUCTURES

Sales	Customers	Products
Sales ID (PK)	Customer ID (PK)	Product (PK)
Order ID	Purchase Address	Product ID
Product (FK)	Address Name	Unit Price **
Quantity Ordered	City	
Price Each**	State	
Order Date*	Zip Code	
Purchase Address		
Order Month		
Order Day		
Order Year		

* - Order Date was kept as it contains the timestamp on when the order was made (this is information that could be used for more in-depth analysis). It is for this reason that this field was maintained.

** - The Price Each field is present in both the Sales and Products **not** to relate them together as the Product field already accomplishes this. Price Each is present to maintain the integrity of both tables. E.g., should the Price of a Product increase or decrease and require updating in the Products table it should not be updated for past Sales transactions in the Sales table (only for new Sales records entered after the update to price in the Products table).

```
USE SalesproductAnalysis;

/*      -------------------------------------------------------*/--
/*      -----------------DATABASE DESIGN      -----------------*/--
/*      -------------------------------------------------------*/--

/* CREATING A TABLE THAT INCLUDES ALL MONTHS DATA THAT WILL BE RUN INTO THE SALES TABLE
*/
/* USED UNION TO REMOVE ANY DUPLICATES ROWS THAT MAY EXIST BETWEEN THE TABLES */

DROP TABLE IF EXISTS sales2019;

SELECT * INTO sales2019 FROM
(SELECT * FROM Sales_January_2019 UNION
SELECT * FROM Sales_February_2019 UNION
SELECT * FROM Sales_March_2019 UNION
SELECT * FROM Sales_April_2019 UNION
SELECT * FROM Sales_May_2019 UNION
SELECT * FROM Sales_June_2019 UNION
SELECT * FROM Sales_July_2019 UNION
SELECT * FROM Sales_August_2019 UNION
SELECT * FROM Sales_September_2019 UNION
SELECT * FROM Sales_October_2019 UNION
SELECT * FROM Sales_November_2019 UNION
SELECT * FROM Sales_December_2019) AS sales2019;

SELECT TOP (5) * FROM sales2019;
```

Order ID	Product	Quantity Ordered	Price Each	Order Date	Purchase Address
254046	Google Phone	1	600	9/8/19 13:53	“875 Wilson St, Boston, MA 02215”
254046	Wired Headphones	1	11.99	9/8/19 13:53	“875 Wilson St, Boston, MA 02215”
254047	20in Monitor	1	109.99	9/7/19 21:47	“739 Sunset St, New York City, NY 10001”
254048	Apple Airpods Headphones	1	150	9/12/19 16:49	“377 6th St, Portland, OR 97035”
254049	AA Batteries (4-pack)	1	3.84	9/29/19 11:53	“858 Sunset St, Portland, OR 97035”

Cleaning the Data:

Since I will be inserting the data from the sales2019 into the final sales table, I will first clean the sales2019 table beforehand.

```
/* RENAME COLUMNS IN sales2019 TABLE */

EXEC sp_RENAME 'sales2019.Order ID','order_id','COLUMN';
EXEC sp_RENAME 'sales2019.product','product','COLUMN';
EXEC sp_RENAME 'sales2019.Quantity Ordered','qty_ordered','COLUMN';
EXEC sp_RENAME 'sales2019.Price Each','price_each','COLUMN';
EXEC sp_RENAME 'sales2019.Order Date','order_date','COLUMN';
EXEC sp_RENAME 'sales2019.Purchase Address','purchase_address','COLUMN';


/* CHECKING THE DATA TYPES FOR THE COLUMNS IN THE sales2019 TABLE */


SELECT COLUMN_NAME, DATA_TYPE FROM INFORMATION_SCHEMA.COLUMNS
WHERE TABLE_NAME = 'sales2019';
```

COLUMN_NAME	DATA_TYPE
order_id	varchar
product	varchar
qty_ordered	varchar
price_each	varchar
order_date	varchar
purchase_address	varchar

Upon trying to change the data types for the columns the below error message appears as expected upon looking at data types earlier.

```
Msg 245, Level 16, State 1, Line 56

Conversion failed when converting the varchar value 'Order ID' to data type int. The statement has been terminated.
```

Now checking to find rows in the sales2019 table where the order id is not an integer.

```
-- FINDING ROWS IN sales2019 COLUMN WHERE order_id IS NOT AN INTEGER –

SELECT * FROM sales2019 WHERE order_id NOT LIKE '%[0-9]%';
```

order_id	product	qty_ordered	price_each	order_date	purchase_address
Order ID	Product	Quantity Ordered	Price Each	Order Date	Purchase Address

```
-- DELETING ROWS IN sales2019 COLUMN WHERE order_id IS NOT AN INTEGER –

DELETE FROM sales2019 WHERE order_id NOT LIKE '%[0-9]%';

(2 rows affected)

Completion time: 2022-08-24T19:30:26.8122653-04:00


/* CHANGE THE DATA TYPES FOR THE COLUMNS ACCORDINGLY */

ALTER TABLE sales2019
ALTER COLUMN order_id INT;

ALTER TABLE sales2019
ALTER COLUMN product VARCHAR(255);

ALTER TABLE sales2019
ALTER COLUMN qty_ordered INT;

ALTER TABLE sales2019
ALTER COLUMN price_each FLOAT;

ALTER TABLE sales2019
ALTER COLUMN order_date DATETIME;

ALTER TABLE sales2019
ALTER COLUMN purchase_address VARCHAR(255);
```

```
/* CHECKING THE DATA TYPES FOR THE COLUMNS IN THE sales2019 TABLE */

SELECT COLUMN_NAME, DATA_TYPE FROM INFORMATION_SCHEMA.COLUMNS
WHERE TABLE_NAME = 'sales2019';
```

COLUMN_NAME	DATA_TYPE
order_id	int
product	varchar
qty_ordered	int
price_each	float
order_date	datetime
purchase_address	varchar

```
/* FINDING DUPLICATE ROWS IN THE sales2019 TABLE */

WITH duplicates AS
     (SELECT order_id,
             product,
             price_each,
             order_date,
             purchase_address,
             ROW_NUMBER() OVER (PARTITION BY order_id, product, price_each, order_date, purchase_address
             ORDER BY order_id, product,    price_each, order_date,   purchase_address)  AS ROW_NUM
      FROM sales2019)
SELECT * FROM duplicates WHERE ROW_NUM > 1;
```

Upon trying to find the duplicate rows, the query returns 47 rows (only 8 shown here):

order_id	product	price_each	order_date	purchase_address	ROW_NUM
154215	AAA Batteries (4-pack)	2.99	55:00.0	“600 South St, San Francisco, CA 94016”	2
155031	Lightning Charging Cable	14.95	52:00.0	“805 Ridge St, Austin, TX 73301”	2
159804	Lightning Charging Cable	14.95	09:00.0	“674 14th St, New York City, NY 10001”	2
160329	AAA Batteries (4-pack)	2.99	12:00.0	“166 Hill St, Seattle, WA 98101”	2
163578	AA Batteries (4-pack)	3.84	58:00.0	“57 Adams St, Boston, MA 02215”	2

165732	AAA Batteries (4-pack)	2.99	05:00.0	“408 Park St, Boston, MA 02215”	2
173803	USB-C Charging Cable	11.95	29:00.0	“791 4th St, San Francisco, CA 94016”	2
186331	AAA Batteries (4-pack)	2.99	35:00.0	“553 2nd St, Los Angeles, CA 90001”	2

Further inspection of the first 3 order_ids in the above table show that the order_id's are showing the same product but different quantities for each row. This redundancy will be addressed upon creating the final sales table later.

order_id	product	qty_ordered	price_each	order_date	purchase_address
154215	AAA Batteries (4-pack)	1	2.99	55:00.0	"600 South St, San Francisco, CA 94016"
154215	AAA Batteries (4-pack)	3	2.99	55:00.0	"600 South St, San Francisco, CA 94016"
155031	Lightning Charging Cable	1	14.95	52:00.0	"805 Ridge St, Austin, TX 73301"
155031	Lightning Charging Cable	2	14.95	52:00.0	"805 Ridge St, Austin, TX 73301"
159804	Lightning Charging Cable	1	14.95	09:00.0	"674 14th St, New York City, NY 10001"
159804	Lightning Charging Cable	2	14.95	09:00.0	"674 14th St, New York City, NY 10001"

Removing Quotations in purchase_address column:

```
-- REMOVING QUOTATIONS THAT APPEAR AROUND THE ADDRESS IN THE Purchase Address COLUMN --

UPDATE sales2019
SET [purchase_address] = REPLACE([purchase_address], '"','');
```

```
(185686 rows affected)


Completion time: 2022-08-24T19:49:19.0287081-04:00
```

Creating the Sales Table:

```
/* CREATING THE SALES TABLE FOR THE DATABASE */

DROP TABLE IF EXISTS sales
CREATE TABLE sales (
sales_id UNIQUEIDENTIFIER DEFAULT NEWSEQUENTIALID() NOT NULL PRIMARY KEY,
order_id INT,
product VARCHAR(255),
qty_ordered INT,
price_each FLOAT,
order_date DATETIME,
purchase_address VARCHAR(255))

INSERT INTO sales (order_id, product, qty_ordered, price_each, order_date, purchase_address)
SELECT order_id,
       product,
       SUM(qty_ordered),
       price_each,
       order_date,
       purchase_address
FROM sales2019
GROUP BY order_id, product, price_each, order_date, purchase_address;
```

** Using the `SUM(qty_ordered)` when inserting into the Sales Table resolved the duplicate row issue earlier (reducing the redundancy).

Before:

order_id	product	qty_ordered	price_each	order_date	purchase_address
154215	AAA Batteries (4-pack)	1	2.99	55:00.0	“600 South St, San Francisco, CA 94016”
154215	AAA Batteries (4-pack)	3	2.99	55:00.0	“600 South St, San Francisco, CA 94016”
155031	Lightning Charging Cable	1	14.95	52:00.0	“805 Ridge St, Austin, TX 73301”
155031	Lightning Charging Cable	2	14.95	52:00.0	“805 Ridge St, Austin, TX 73301”
159804	Lightning Charging Cable	1	14.95	09:00.0	“674 14th St, New York City, NY 10001”
159804	Lightning Charging Cable	2	14.95	09:00.0	“674 14th St, New York City, NY 10001”

After:

order_id	product	qty_ordered	price_each
154215	AAA Batteries (4-pack)	4	2.99
155031	Lightning Charging Cable	3	14.95
159804	Lightning Charging Cable	3	14.95

```
SELECT TOP (5) * FROM sales;
```

sales_id	order_id	product	qty_ordered	price_each	order_date	purchase_address
B682DE57-7C23-ED11-9FDE-94E6F7BD022B	141234	iPhone	1	700	25:00.0	944 Walnut St, Boston, MA 02215
B782DE57-7C23-ED11-9FDE-94E6F7BD022B	141235	Lightning Charging Cable	1	14.95	15:00.0	185 Maple St, Portland, OR 97035
B882DE57-7C23-ED11-9FDE-94E6F7BD022B	141236	Wired Headphones	2	11.99	33:00.0	538 Adams St, San Francisco, CA 94016
B982DE57-7C23-ED11-9FDE-94E6F7BD022B	141237	27in FHD Monitor	1	149.99	33:00.0	738 10th St, Los Angeles, CA 90001
BA82DE57-7C23-ED11-9FDE-94E6F7BD022B	141238	Wired Headphones	1	11.99	59:00.0	387 10th St, Austin, TX 73301

Creating the Customer Table:

```sql
/* CREATE THE CUSTOMER TABLE */


DROP TABLE IF EXISTS customers
CREATE TABLE customers (
customer_id INT IDENTITY(100000, 1) PRIMARY KEY,
purchase_address VARCHAR(255) NOT NULL);

SELECT * FROM customers

-- STORING UNIQUE VALUES IN purchase_address COLUMN FROM THE SALES TABLE --
-- THEN PARSING RESULTS TO INSERT INTO THE OTHER CUSTOMER TABLE FIELDS --

INSERT INTO customers (purchase_address)
SELECT DISTINCT purchase_address
FROM sales;

ALTER TABLE customers
ADD address_name VARCHAR(255);

ALTER TABLE customers ADD city VARCHAR(255);

ALTER TABLE customers
ADD address_state VARCHAR(10);

ALTER TABLE customers
ADD zip_code VARCHAR(10);

UPDATE customers
SET address_name = PARSENAME(REPLACE(purchase_address,',','.'),3);

UPDATE customers
SET city = PARSENAME(REPLACE(purchase_address,',','.'),2);
```

```sql
UPDATE customers
SET address_state = PARSENAME(REPLACE(purchase_address,',','.'),1);

-- 2ND UPDATE TO address_state COLUMN TO REMOVE ZIP CODE --
UPDATE customers
SET address_state = PARSENAME(REPLACE(address_state,' ','.'),2);

UPDATE customers
SET zip_code = REVERSE(PARSENAME(REPLACE(REVERSE(purchase_address),',','.'),3));

-- 2ND UPDATE TO zip_code COLUMN TO REMOVE address_state –
UPDATE customers
SET zip_code = PARSENAME(REPLACE(zip_code,' ','.'),1);

SELECT * FROM cus
```

customer_id	purchase_address	address_name	city	address_state	zip_code
100000	1 11th St, Atlanta, GA 30301	1 11th St	Atlanta	GA	30301
100001	1 11th St, Los Angeles, CA 90001	1 11th St	Los Angeles	CA	90001
100002	1 11th St, San Francisco, CA 94016	1 11th St	San Francisco	CA	94016
100003	1 12th St, Los Angeles, CA 90001	1 12th St	Los Angeles	CA	90001
100004	1 12th St, New York City, NY 10001	1 12th St	New York City	NY	10001

Creating the Product Table:

```
/* CREATE THE PRODUCT TABLE */

-- STORING UNIQUE VALUES IN THE product COLUMN –

DROP TABLE IF EXISTS product
CREATE TABLE product(
product_id INT IDENTITY(1, 1),
product VARCHAR(255) NOT NULL PRIMARY KEY,
unit_price FLOAT);

-- STORING UNIQUE VALUES IN product COLUMN FROM SALES TABLE WITHIN CTE NAMED item --
-- THEN INSERT RESULTS INTO PRODUCT TABLE --

WITH item AS (
     SELECT DISTINCT product,
            CAST(price_each AS FLOAT) AS price_each
     FROM sales)
INSERT INTO product (product, unit_price)
SELECT product, price_each FROM item;

SELECT * FROM product;
```

product_id	product	unit_price
1	20in Monitor	109.99
2	27in 4K Gaming Monitor	389.99
3	27in FHD Monitor	149.99
4	34in Ultrawide Monitor	379.99
5	AA Batteries (4-pack)	3.84
6	AAA Batteries (4-pack)	2.99
7	Apple Airpods Headphones	150
8	Bose SoundSport Headphones	99.99
9	Flatscreen TV	300
10	Google Phone	600
11	iPhone	700
12	LG Dryer	600
13	LG Washing Machine	600
14	Lightning Charging Cable	14.95
15	Macbook Pro Laptop	1700
16	ThinkPad Laptop	999.99

17	USB-C Charging Cable	11.95
18	Vareebadd Phone	400
19	Wired Headphones	11.99

Additional Updates to the Sales Table:

I will now add a month, day, and year column to the sales table to be used upon querying database for analysis later.

```
/* CREATING MONTH, DAY, AND YEAR COLUMNS IN THE SALES TABLE FROM THE order_date COLUMN */

ALTER TABLE sales
ADD date_month INT;

UPDATE sales
SET date_month = MONTH(order_date) FROM sales;

ALTER TABLE sales
ADD date_day INT;

UPDATE sales
SET date_day = DAY(order_date) FROM sales;

ALTER TABLE sales
ADD date_year INT;

UPDATE sales
SET date_year = YEAR(order_date) FROM sales;

SELECT TOP (5) * FROM sales;
```

sales_id	order_id	product	qty_ordered	price_each	order_date	purchase_address	date_month	date_day	date_year
B682DE57-7C23-ED11-9FDE-94E6F7BD022B	141234	iPhone	1	700	25:00.0	944 Walnut St, Boston, MA 02215	1	22	2019
B782DE57-7C23-ED11-9FDE-94E6F7BD022B	141235	Lightning Charging Cable	1	14.95	15:00.0	185 Maple St, Portland, OR 97035	1	28	2019

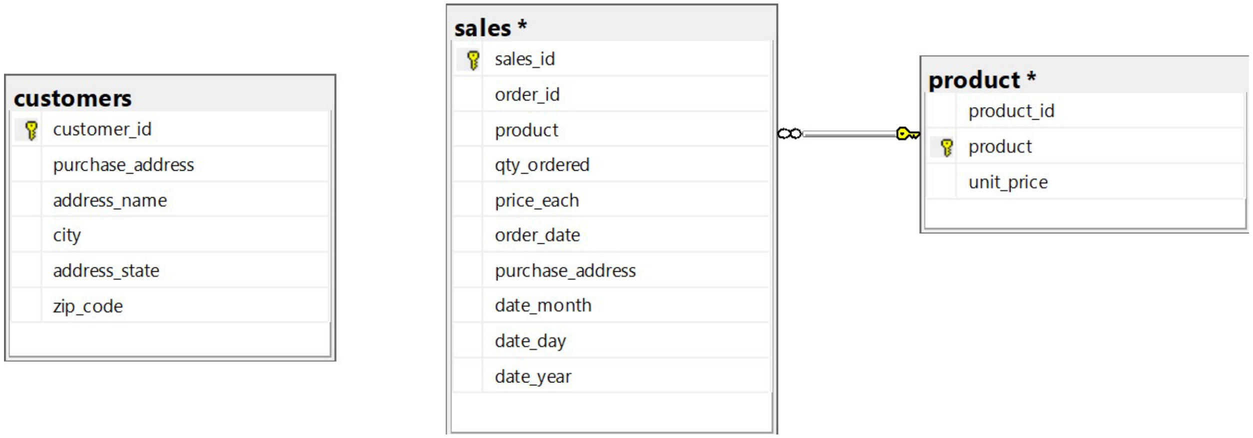
B882DE57-7C23-ED11-9FDE-94E6F7BD022B	141236	Wired Headphones	2	11.99	33:00.0	538 Adams St, San Francisco, CA 94016	1	17	2019
B982DE57-7C23-ED11-9FDE-94E6F7BD022B	141237	27in FHD Monitor	1	149.99	33:00.0	738 10th St, Los Angeles, CA 90001	1	5	2019
BA82DE57-7C23-ED11-9FDE-94E6F7BD022B	141238	Wired Headphones	1	11.99	59:00.0	387 10th St, Austin, TX 73301	1	25	2019

Assigning Foreign Keys:

```
/* ASSIGNING FOREIGN KEYS */

ALTER TABLE sales
ADD FOREIGN KEY (product) REFERENCES product (product);
```

Database Diagram



customers
- customer_id
- purchase_address
- address_name
- city
- address_state
- zip_code

sales *
- sales_id
- order_id
- product
- qty_ordered
- price_each
- order_date
- purchase_address
- date_month
- date_day
- date_year

product *
- product_id
- product
- unit_price

Creating Views:

Order View – will return list of all orders made in addition to the customer_id, all products within the order, number of items, order total and purchase address.

```
-- CREATING AN ORDERS VIEW --
-- ORDER VIEW WILL SHOW ALL ORDER IDS, CUSTOMER IDS ASSOCIATED WITH ORDERS, ALL PRODUCTS IN ORDER TOGETHER --
-- ,NUMBER OF ITEMS, ORDER TOTAL, AND PURCHASE ADDRESS --
```

```
CREATE VIEW orders
AS
SELECT DISTINCT s.order_id,
                c.customer_id,
                STRING_AGG(s.product,'/') AS all_products,
                SUM(s.qty_ordered) AS NumberofItems,
                SUM (s.qty_ordered*s.price_each) AS CompleteOrderTotal,
                s.purchase_address
FROM sales AS s
JOIN customers AS c
ON s.purchase_address = c.purchase_address
GROUP BY order_id, c.customer_id, s.purchase_address;
```

Checking Orders View:

```
SELECT TOP (15) * FROM orders;
```

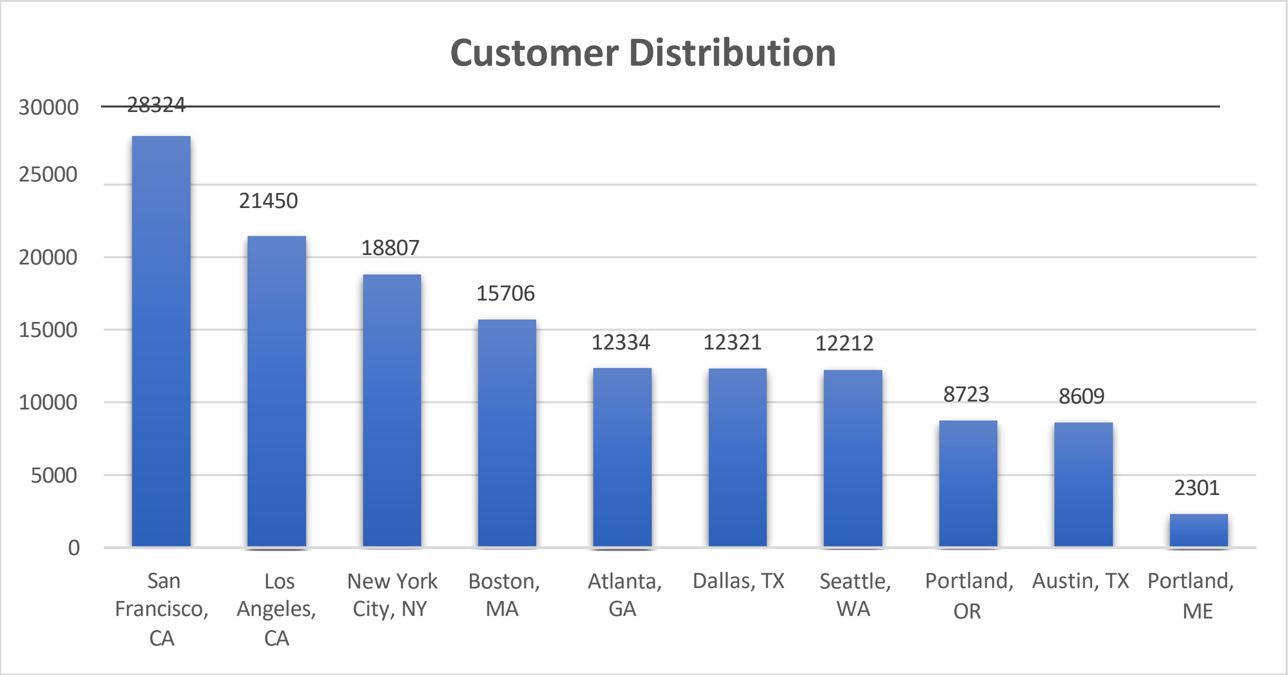
order_id	customer_id	all_products	NumberofItems	CompleteOrderTotal	purchase_address
165233	100000	USB-C Charging Cable	1	11.95	1 11th St, Atlanta, GA 30301
269712	100001	Macbook Pro Laptop	1	1700	1 11th St, Los Angeles, CA 90001
254600	100002	iPhone	1	700	1 11th St, San Francisco, CA 94016
162564	100003	Apple Airpods Headphones	1	150	1 12th St, Los Angeles, CA 90001
262841	100004	Wired Headphones	1	11.99	1 12th St, New York City, NY 10001
285943	100005	Wired Headphones	1	11.99	1 12th St, San Francisco, CA 94016

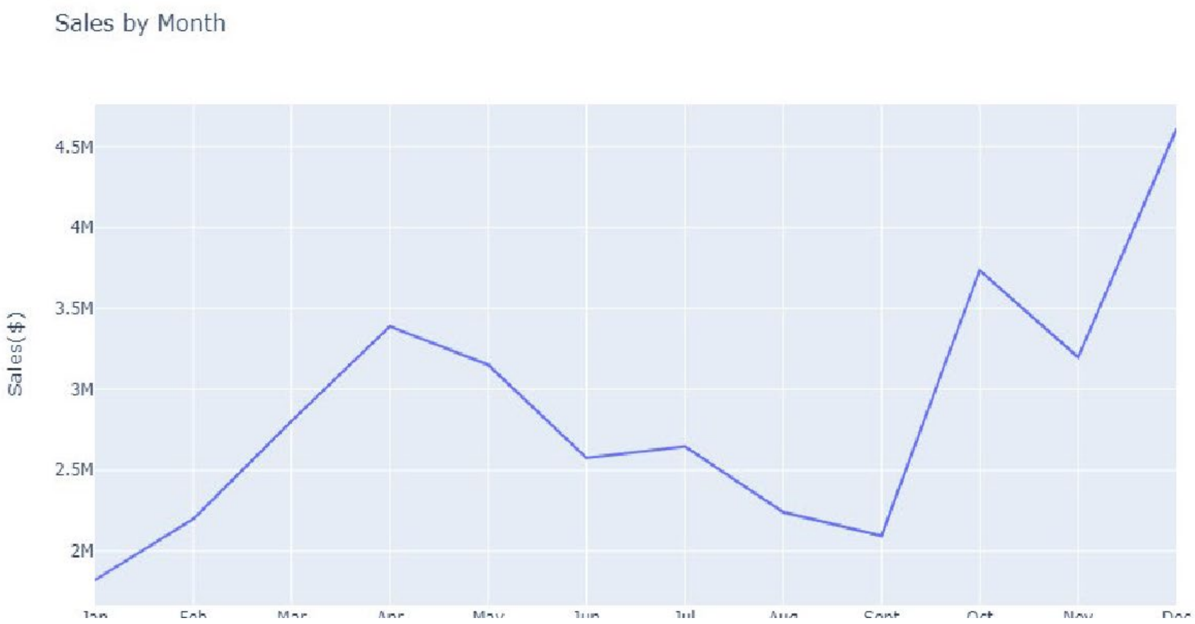
302402	100005	Macbook Pro Laptop	1	1700	1 12th St, San Francisco, CA 94016
175491	100006	Wired Headphones	1	11.99	1 13th St, San Francisco, CA 94016
180875	100007	Bose SoundSport Headphones	1	99.99	1 14th St, New York City, NY 10001
267290	100007	Lightning Charging Cable	1	14.95	1 14th St, New York City, NY 10001
264712	100008	Macbook Pro Laptop/USB-C Charging Cable	3	1723.9	1 14th St, Portland, OR 97035
250711	100009	AA Batteries (4-pack)	2	7.68	1 14th St, San Francisco, CA 94016
301827	100010	Apple Airpods Headphones	1	150	1 14th St, Seattle, WA 98101
186711	100011	Wired Headphones	1	11.99	1 1st St, Austin, TX 73301
213897	100012	USB-C Charging Cable	2	23.9	1 1st St, Dallas, TX 75001

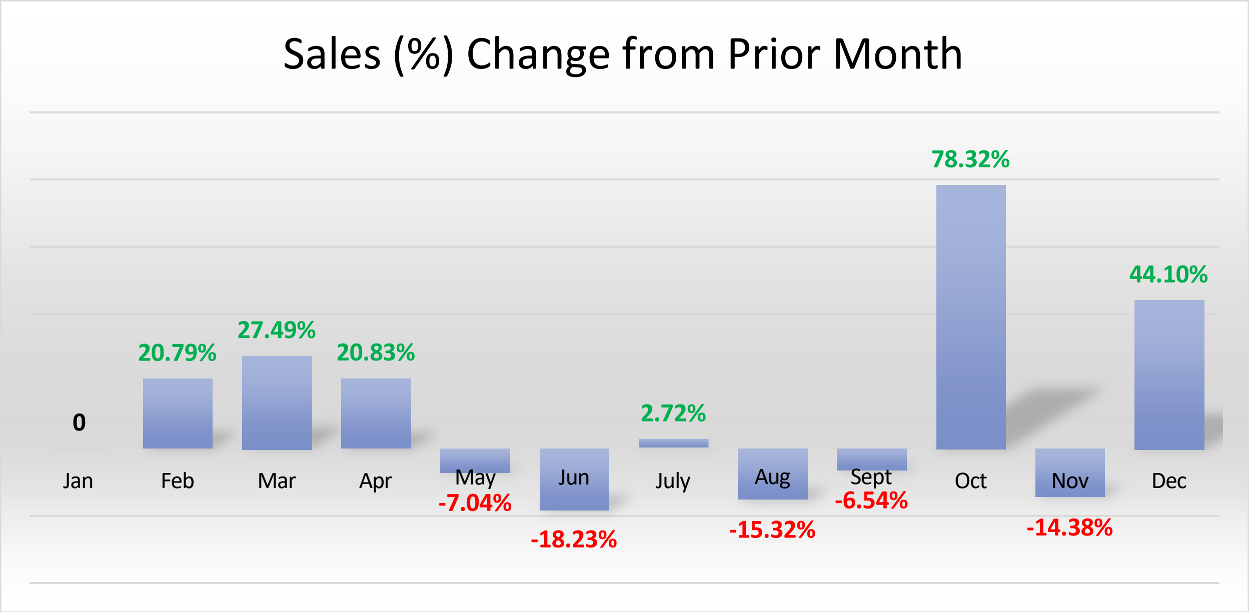
Opportunities for Database Design Improvement:

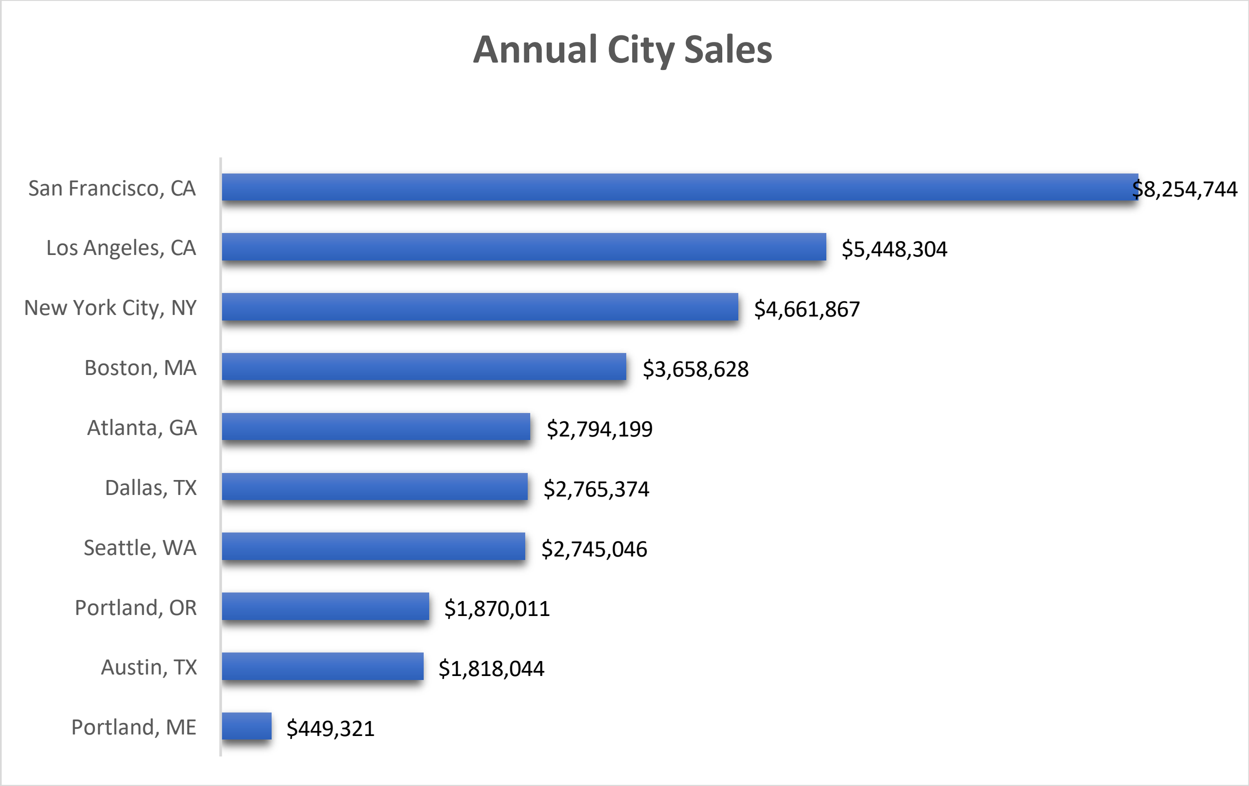
1. The original design had the purchase address as a Multipart Field (address not separated into Address, City, State, Zip Code columns). Changes in the online input form from the company's website should be made **prior** to new database implementation to better support the new design.

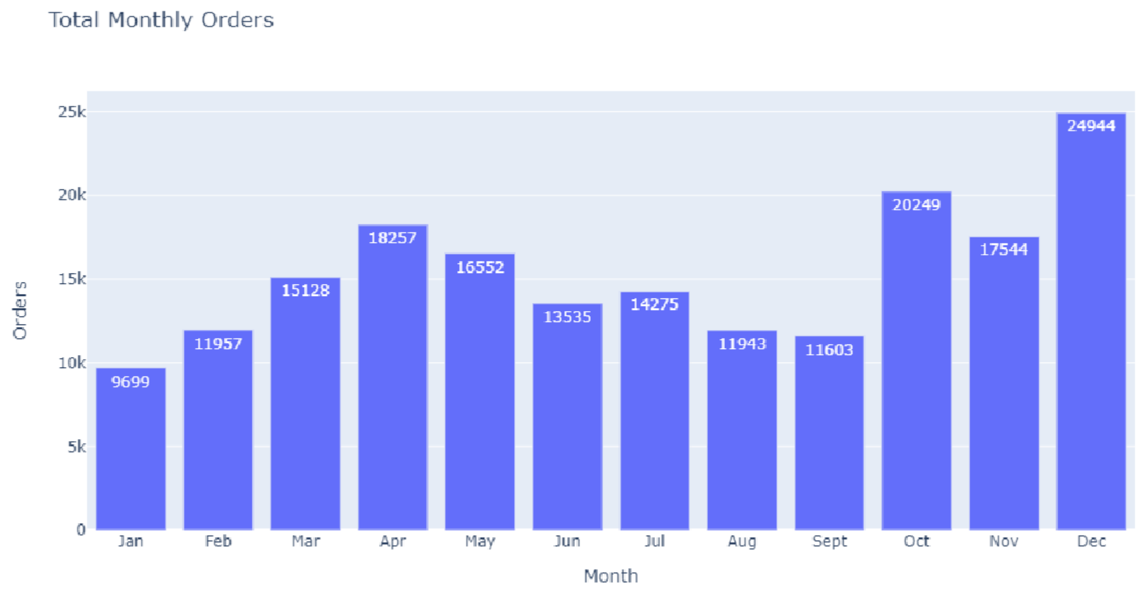
2. An inventory column could be added to the Products table to keep track of current (not sold) inventory (if data becomes available).


Customer Distribution
30000
25000
20000
15000
10000
5000
0
28324
21450
18807
15706
12334
12321
12212
8723
8609
2301
San Francisco, CA
Los Angeles, CA
New York City, NY
Boston, MA
Atlanta, GA
Dallas, TX
Seattle, WA
Portland, OR
Austin, TX
Portland, ME


Sales by Month
Sales($)
4.5M
4M
3.5M
3M
2.5M
2M
Jan
Feb
Mar
Apr
May
Jun
Jul
Aug
Sept
Oct
Nov
Dec


Sales (%) Change from Prior Month
0
Jan
20.79%
Feb
27.49%
Mar
20.83%
Apr
May
-7.04%
Jun
-18.23%
2.72%
July
Aug
-15.32%
Sept
-6.54%
78.32%
Oct
Nov
-14.38%
44.10%
Dec


Annual City Sales
San Francisco, CA
$8,254,744
Los Angeles, CA
$5,448,304
New York City, NY
$4,661,867
Boston, MA
$3,658,628
Atlanta, GA
$2,794,199
Dallas, TX
$2,765,374
Seattle, WA
$2,745,046
Portland, OR
$1,870,011
Austin, TX
$1,818,044
Portland, ME
$449,321



Top 3 States in Market:

State	Sales	Avg. Sales (Entire Market)	Amt. over Avg. Sales
CA	$13,703,047	$4,308,191	$9,394,856
NY	$4,661,867	$4,308,191	$353,676
TX	$4,583,418	$4,308,191	$275,227

PRODUCT SALES ANALYSIS

```
/*      -------------------------------------------------------*/--
/*      ---------------PRODUCT SALES ANALYSIS      ---------------*/---
/*      -------------------------------------------------------*/--


-- BASIC CUSTOMER BREAKDOWN INFORMATION --

-- FINDING THE NUMBER OF CUSTOMERS IN EACH STATE --

SELECT address_state,
       COUNT(customer_id) AS 'number of customers'
FROM customers
GROUP BY address_state
ORDER BY address_state;

-- SEEING THE BREAKDOWN OF NUMBER OF CUSTOMERS BY CITY --

SELECT city AS 'City',
       address_state AS 'State',
       COUNT(customer_id) AS 'Number of Customers'
FROM customers
GROUP BY city, address_state
ORDER BY address_state, city;
```

City	State	Number of Customers
Los Angeles	CA	21450
San Francisco	CA	28324
Atlanta	GA	12334
Boston	MA	15706
Portland	ME	2301
New York City	NY	18807
Portland	OR	8723
Austin	TX	8609
Dallas	TX	12321
Seattle	WA	12212

```sql
-- FINDING THE MONTHLY ORDER TOTALS --

SELECT date_month,
       ROUND(SUM(qty_ordered*price_each),2) AS 'monthly totals'
FROM sales
GROUP BY date_month
ORDER BY date_month;
```

date_month	monthly totals
1	1821413.16
2	2200078.08
3	2804973.35
4	3389217.98
5	3150616.23
6	2576280.15
7	2646461.32
8	2241083.37
9	2094465.69
10	3734777.86
11	3197875.05
12	4608295.7

```sql
-- FINDING THE SALES TOTALS FOR EACH product FOR EACH MONTH --

SELECT product,
       date_month,
       ROUND(SUM(qty_ordered*price_each),2) AS 'monthly_product_sales'
FROM sales
GROUP BY product, date_month
ORDER BY product, date_month;


-- FINDING THE SALES TOTALS FOR EACH STATE --

SELECT c.address_state,
       ROUND(SUM(s.qty_ordered*s.price_each),2) AS 'sales_totals'
FROM customers AS c
JOIN sales AS s
ON c.purchase_address = s.purchase_address
GROUP BY c.address_state
ORDER BY c.address_state;
```

address_state	sales_totals
CA	13703048
GA	2794199
MA	3658628
ME	449321.4
NY	4661867
OR	1870011
TX	4583418
WA	2745046

```
-- FINDING THE SALES TOTALS FOR EACH CITY --

SELECT c.city,
       c.address_state,
       ROUND(SUM(s.qty_ordered*s.price_each),2) AS 'sales_totals'
FROM customers AS c
JOIN sales AS s
ON c.purchase_address = s.purchase_address
GROUP BY c.city, c.address_state
ORDER BY c.city, c.address_state;
```

city	address_state	sales_totals
Atlanta	GA	2794199
Austin	TX	1818044
Boston	MA	3658628
Dallas	TX	2765374
Los Angeles	CA	5448304
New York City	NY	4661867
Portland	ME	449321.4
Portland	OR	1870011
San Francisco	CA	8254744
Seattle	WA	2745046

```
-- BREAKDOWN THE SALES TOTALS FOR EACH STATE PER MONTH --

SELECT c.address_state,
       s.date_month,
       ROUND(SUM(s.qty_ordered*s.price_each),2) AS 'sales_totals'
FROM customers AS c
JOIN sales AS s
ON c.purchase_address = s.purchase_address
GROUP BY c.address_state, s.date_month
ORDER BY c.address_state, s.date_month;
```

```
--- CONVERTED INTO PIVOT FOR BETTER QUERY RESULT READABILITY ---


SELECT *, CA + GA + MA + ME + NY + [OR] + TX + WA AS month_totals
FROM
(SELECT c.address_state,
       s.date_month AS 'month',
       CAST((s.qty_ordered*s.price_each) AS DECIMAL(10,2)) AS 'sales_totals'
FROM sales AS s
JOIN customers AS c
ON s.purchase_address = c.purchase_address) AS t
PIVOT
 (
      SUM(t.sales_totals)
      FOR address_state IN (CA, GA, MA, ME, NY, [OR], TX, WA))
AS pivot_table
ORDER BY 'month';
```

month	CA	GA	MA	ME	NY	OR	TX	WA	month_totals
1	724151.3	149159.5	201057.8	22708.8	259829.3	92276.76	231537.6	140692.1	1821413
2	888850.3	176458.3	213612.6	29845.49	305372.3	119594.4	295014.5	171330.3	2200078
3	1122716	231605.4	301023.8	30406.3	367226.3	156541.7	376765.3	218688.8	2804973
4	1362468	284422	353392.2	42536.49	449314.9	197441.6	423919.1	275724.1	3389218
5	1274554	238842	328791.7	57978.76	436120.4	173729.3	428961.8	211638.5	3150616
6	1063898	219801.5	254461.2	29998.43	323886.6	139462	330931.1	213841.6	2576280
7	1036045	211663.5	291478.4	32421.14	355698.2	143994.5	362650.1	212510.9	2646461
8	882878.9	169267.7	239260.3	35996.6	302401.5	116716.2	305227.1	189335.1	2241083
9	816945.7	171263.9	248231.8	28759.56	300401.9	103796.9	268696.6	156369.3	2094466
10	1478083	306159.1	367003.5	52022.52	486950.6	201766.4	525951.8	316841.3	3734778
11	1264136	275061.8	350834.1	34681.22	428156.3	173195.7	419741.3	252068.2	3197875
12	1788323	360494.3	509480.4	51966.07	646508.9	251495.1	614022	386006	4608296

```sql
-- COMPARING THE SALES OF THE CURRENT MONTH WITH THE PREVIOUS MONTH FOR EACH STATE ALONG WITH PERCENTAGE CHANGE --

WITH state_sales AS(
     SELECT c.address_state,
            s.date_month,
            ROUND(SUM(s.qty_ordered*s.price_each),2) AS 'sales_totals',
            LAG(ROUND(SUM(s.qty_ordered*s.price_each),2)) OVER (PARTITION BY address_state ORDER BY date_month) AS
            'previous_month_sales'
FROM customers AS c
JOIN sales AS s
ON c.purchase_address = s.purchase_address
GROUP BY c.address_state, s.date_month)
SELECT address_state AS 'state',
       date_month AS 'month',
       sales_totals,
       previous_month_sales,
       FORMAT((sales_totals-previous_month_sales) / previous_month_sales,'P') AS pct_change
FROM state_sales;
```

* only CA and GA shown below

state	month	sales_totals	previous_month_sales	pct_change
CA	1	724151.3	NULL	NULL
CA	2	888850.3	724151.3	22.74%
CA	3	1122716	888850.3	26.31%
CA	4	1362468	1122716	21.35%
CA	5	1274554	1362468	-6.45%
CA	6	1063898	1274554	-16.53%
CA	7	1036045	1063898	-2.62%
CA	8	882878.9	1036045	-14.78%
CA	9	816945.7	882878.9	-7.47%
CA	10	1478083	816945.7	80.93%
CA	11	1264136	1478083	-14.47%
CA	12	1788323	1264136	41.47%
GA	1	149159.5	NULL	NULL
GA	2	176458.3	149159.5	18.30%
GA	3	231605.4	176458.3	31.25%
GA	4	284422	231605.4	22.80%
GA	5	238842	284422	-16.03%
GA	6	219801.5	238842	-7.97%
GA	7	211663.5	219801.5	-3.70%

GA	8	169267.7	211663.5	-20.03%
GA	9	171263.9	169267.7	1.18%
GA	10	306159.1	171263.9	78.76%
GA	11	275061.8	306159.1	-10.16%
GA	12	360494.3	275061.8	31.06%

```
-- COMPARING THE SALES OF THE CURRENT MONTH WITH THE PREVIOUS MONTH ALONG WITH PERCENTAGE CHANGE --

WITH month_sales AS(
              SELECT date_month,
                     ROUND(SUM(qty_ordered*price_each),2) AS 'sales_totals',
                     LAG(ROUND(SUM(qty_ordered*price_each),2)) OVER (ORDER BY date_month)  AS 'previous_month_sales'
              FROM sales
              GROUP BY date_month)
SELECT date_month AS 'month',
       sales_totals,
       previous_month_sales,
       FORMAT((sales_totals-previous_month_sales) / previous_month_sales,'P') AS   pct_change
FROM month_sales;
```

month	sales_totals	previous_month_sales	pct_change
1	1821413	NULL	NULL
2	2200078	1821413	20.79%
3	2804973	2200078	27.49%
4	3389218	2804973	20.83%
5	3150616	3389218	-7.04%
6	2576280	3150616	-18.23%
7	2646461	2576280	2.72%
8	2241083	2646461	-15.32%
9	2094466	2241083	-6.54%
10	3734778	2094466	78.32%
11	3197875	3734778	-14.38%
12	4608296	3197875	44.10%

```
-- FINDING THE TOTAL NUMBER OF EACH PRODUCT ORDERED EACH MONTH --

SELECT product,
       date_month,
       SUM(qty_ordered) AS 'total_qty_ordered'
FROM sales
GROUP BY product, date_month
ORDER BY product, date_month;
```

```sql
-- FINDING THE TOTAL NUMBER OF EACH PRODUCT ORDERED FOR ENTIRE YEAR --

SELECT product,
       SUM(qty_ordered) AS 'total_ordered'
FROM sales
GROUP BY product
ORDER BY product;

-- FINDING THE NUMBER OF EACH PRODUCT SOLD IN EACH STATE + TOTALS --
-- USING PIVOT FOR BETTER QUERY RESULT READABILITY --

SELECT *, CA + GA + MA + ME + NY + [OR] + TX + WA AS total_units_sold
FROM
(SELECT c.address_state,
        s.product,
        s.qty_ordered AS inventory_sold
FROM sales AS s
JOIN customers AS c
ON s.purchase_address = c.purchase_address) AS t
PIVOT
 (
     SUM(t.inventory_sold)
     FOR address_state IN (CA, GA, MA, ME, NY, [OR], TX, WA))
AS pivot_table
ORDER BY product;
```

product	CA	GA	MA	ME	NY	OR	TX	WA	total_units_sold
20in Monitor	1658	341	394	58	560	219	572	324	4126
27in 4K Gaming Monitor	2461	492	675	85	841	349	798	538	6239
27in FHD Monitor	3032	587	797	114	1072	415	945	579	7541
34in Ultrawide Monitor	2398	482	670	76	867	327	840	532	6192
AA Batteries (4-pack)	10983	2193	3011	389	3629	1550	3682	2178	27615
AAA Batteries (4-pack)	12362	2358	3458	358	4119	1720	4168	2443	30986
Apple Airpods Headphones	6197	1266	1651	233	2094	864	2077	1255	15637
Bose SoundSport Headphones	5433	1082	1411	180	1791	707	1765	1061	13430
Flatscreen TV	1880	406	553	61	628	250	661	374	4813
Google Phone	2205	451	592	77	757	278	735	434	5529
iPhone	2778	544	752	79	881	371	896	546	6847

LG Dryer	259	59	59	6	77	31	99	56	646
LG Washing Machine	285	52	72	11	85	26	77	58	666
Lightning Charging Cable	9305	1874	2482	268	3039	1260	3167	1774	23169
Macbook Pro Laptop	1885	379	479	63	657	274	632	356	4725
ThinkPad Laptop	1603	357	447	53	560	221	555	332	4128
USB-C Charging Cable	9659	1912	2555	339	3263	1241	3098	1864	23931
Vareebadd Phone	811	173	214	17	281	107	286	179	2068
Wired Headphones	8222	1576	2222	279	2702	1081	2791	1651	20524

```sql
-- FINDING AVERAGE SALES TOTALS IN EACH STATE --
SELECT c.address_state AS 'State',
       (ROUND(AVG(s.qty_ordered*s.price_each),2)) AS 'Average Order Totals-State'
FROM customers AS c
JOIN sales AS s
ON c.purchase_address = s.purchase_address
GROUP BY c.address_state
ORDER BY c.address_state;

-- FINDING AVERAGE SALES TOTALS IN EACH CITY --

SELECT c.city AS 'City',
       (ROUND(AVG(s.qty_ordered*s.price_each),2)) AS 'Average Order Total-City'
FROM customers AS c
JOIN sales AS s
ON c.purchase_address = s.purchase_address
GROUP BY c.city
ORDER BY c.city;


-- TOTAL NUMBER OF ORDERS IN EACH STATE --

SELECT c.address_state AS 'State',
       COUNT(s.order_id) AS 'Total Number of Orders-State'
FROM customers AS c
JOIN sales AS s
ON c.purchase_address = s.purchase_address
GROUP BY c.address_state
ORDER BY c.address_state;


-- TOTAL NUMBER OF ORDERS IN EACH CITY --

SELECT c.city AS 'City',
       c.address_state AS 'State',
       COUNT(s.order_id) AS 'Total Number of Orders-City'
FROM customers AS c
JOIN sales AS s
ON c.purchase_address = s.purchase_address
GROUP BY c.city, c.address_state
ORDER BY c.city, c.address_state;
```

```
-- FINDING THE STATES WHERE YEARLY SALES TOTALS WERE GREATER THAN THE YEARLY AVERAGE SALES OF ENTIRE MARKET (ALL STATES) --

WITH CTE_1 AS
      (SELECT c.address_state,
             CAST(SUM(s.qty_ordered*s.price_each) AS INT) AS 'state_sales_totals'
       FROM customers AS c
       JOIN sales AS s
       ON c.purchase_address = s.purchase_address
       GROUP BY c.address_state),
     CTE_2 AS
     (SELECT AVG(state_sales_totals) AS 'market_avg_sales'
      FROM CTE_1)
SELECT *, (CTE_1.state_sales_totals - CTE_2.market_avg_sales) AS 'Sales_Amt_over_Avg'
FROM CTE_1
JOIN CTE_2
ON CTE_1.state_sales_totals > CTE_2.market_avg_sales;
```

address_state	state_sales_totals	market_avg_sales	Sales_Amt_over_Avg
TX	4583418	4308191	275227
CA	13703047	4308191	9394856
NY	4661867	4308191	353676

```
  -- BASED ON THE STATE WHERE YEARLY SALES TOTALS GREATER THAN YEARLY AVERAGE SALES (ENTIRE MARKET) --
  -- TAKING A LOOK AT THE TOTAL INVENTORY SOLD AND SALES OF EACH product IN EACH OF THOSE STATES --
  -- USING ROLLUP FOR SUBTOTALS --

SELECT COALESCE(c.address_state, 'All States Totals') AS 'top_states',
       COALESCE(s.product, 'All products') AS 'product',
       SUM(s.qty_ordered) AS 'inventory_sold',
       ROUND(SUM(s.price_each*s.qty_ordered),0) AS 'total_sales'
FROM sales AS s
JOIN customers AS c
ON s.purchase_address = c.purchase_address
WHERE c.address_state IN ('CA', 'NY', 'TX')
GROUP BY ROLLUP (s.product, c.address_state);
```

* only first 8 products shown below

top_states	product	inventory_sold	total_sales
CA	20in Monitor	1658	182363
NY	20in Monitor	560	61594
TX	20in Monitor	572	62914
All States Totals	20in Monitor	2790	306872
CA	27in 4K Gaming Monitor	2461	959765
NY	27in 4K Gaming Monitor	841	327982
TX	27in 4K Gaming Monitor	798	311212
All States Totals	27in 4K Gaming Monitor	4100	1598959
CA	27in FHD Monitor	3032	454770
NY	27in FHD Monitor	1072	160789
TX	27in FHD Monitor	945	141741
All States Totals	27in FHD Monitor	5049	757300
CA	34in Ultrawide Monitor	2398	911216
NY	34in Ultrawide Monitor	867	329451
TX	34in Ultrawide Monitor	840	319192
All States Totals	34in Ultrawide Monitor	4105	1559859
CA	AA Batteries (4-pack)	10983	42175
NY	AA Batteries (4-pack)	3629	13935
TX	AA Batteries (4-pack)	3682	14139
All States Totals	AA Batteries (4-pack)	18294	70249
CA	AAA Batteries (4-pack)	12362	36962
NY	AAA Batteries (4-pack)	4119	12316
TX	AAA Batteries (4-pack)	4168	12462
All States Totals	AAA Batteries (4-pack)	20649	61741
CA	Apple Airpods Headphones	6197	929550
NY	Apple Airpods Headphones	2094	314100
TX	Apple Airpods Headphones	2077	311550
All States Totals	Apple Airpods Headphones	10368	1555200

CA	Bose SoundSport Headphones	5433	543246
NY	Bose SoundSport Headphones	1791	179082
TX	Bose SoundSport Headphones	1765	176482
All States Totals	Bose SoundSport Headphones	8989	898810

```sql
-- FINDING THE TOTAL SALES OF EACH product WITHIN EACH STATE WITH FINAL TOTAL --
-- REPRESENTED AS PIVOT TABLE --

SELECT *, CA + GA + MA + ME + NY + [OR] + TX + WA AS total_sales
FROM
(SELECT c.address_state,
        s.product,
        CAST((s.qty_ordered*s.price_each) AS DECIMAL(10,2)) AS 'sales_totals'
 FROM sales AS s
 JOIN customers AS c
 ON s.purchase_address = c.purchase_address) AS t
 PIVOT
 (
     SUM(t.sales_totals)
     FOR address_state IN (CA, GA, MA, ME, NY,[OR],TX, WA))
AS pivot_table
ORDER BY total_sales DESC;
```

product	CA	GA	MA	ME	NY	OR	TX	WA	total_sales
Macbook Pro Laptop	3204500	644300	814300	107100	1116900	465800	1074400	605200	8032500
iPhone	1944600	380800	526400	55300	616700	259700	627200	382200	4792900
ThinkPad Laptop	1602984	356996.4	446995.5	52999.47	559994.4	220997.8	554994.5	331996.7	4127959
Google Phone	1323000	270600	355200	46200	454200	166800	441000	260400	3317400
27in 4K Gaming Monitor	959765.4	191875.1	263243.3	33149.15	327981.6	136106.5	311212	209814.6	2433148
34in Ultrawide Monitor	911216	183155.2	254593.3	28879.24	329451.3	124256.7	319191.6	202154.7	2352898
Apple Airpods Headphones	929550	189900	247650	34950	314100	129600	311550	188250	2345550
Flatscreen TV	564000	121800	165900	18300	188400	75000	198300	112200	1443900
Bose SoundSport Headphones	543245.7	108189.2	141085.9	17998.2	179082.1	70692.93	176482.4	106089.4	1342866
27in FHD Monitor	454769.7	88044.13	119542	17098.86	160789.3	62245.85	141740.6	86844.21	1131075
Vareebadd Phone	324400	69200	85600	6800	112400	42800	114400	71600	827200
20in Monitor	182363.4	37506.59	43336.06	6379.42	61594.4	24087.81	62914.28	35636.76	453818.7
LG Washing Machine	171000	31200	43200	6600	51000	15600	46200	34800	399600

LG Dryer	155400	35400	35400	3600	46200	18600	59400	33600	387600
Lightning Charging Cable	139109.8	28016.3	37105.9	4006.6	45433.05	18837	47346.65	26521.3	346376.6
USB-C Charging Cable	115425.1	22848.4	30532.25	4051.05	38992.85	14829.95	37021.1	22274.8	285975.5
Wired Headphones	98581.78	18896.24	26641.78	3345.21	32396.98	12961.19	33464.09	19795.49	246082.8
AA Batteries (4-pack)	42174.72	8421.12	11562.24	1493.76	13935.36	5952	14138.88	8363.52	106041.6
AAA Batteries (4-pack)	36962.38	7050.42	10339.42	1070.42	12315.81	5142.8	12462.32	7304.57	92648.14

```
-- FINDING THE customers WITH THE HIGHEST NUMBER OF ORDERS (INCLUDING THE CITY, STATE) FOR THE ENTIRE YEAR (2019) --

SELECT TOP 5 c.customer_id,
             c.city,
             c.address_state,
             COUNT(DISTINCT s.order_id) AS 'Total Orders'
FROM sales AS s
JOIN customers AS c
ON s.purchase_address = c.purchase_address
GROUP BY city, address_state, customer_id
ORDER BY 'Total Orders' DESC;
```

```
/*      -----------------------------------------------------------*/--
/*      -------------------MISCELLANEOUS      -------------------*/--
/*      -----------------------------------------------------------*/--


/* FINDING THE PRICE FOR EACH product IN EACH STATE */
/* LATER TO ADD THE CURRENT STATE SALES TAX TO CREATE ACTUAL COST COLUMN TO product TABLE
*/

SELECT DISTINCT p.product_id,
                p.product,
                p.unit_price, c.address_state
FROM product AS p
JOIN sales AS s
ON p.product = s.product
JOIN customers AS c
ON s.purchase_address = c.purchase_address
ORDER BY p.product_id;
```

* only first 2 products shown below

product_id	product	Unit_price	address_state
1	20in Monitor	109.99	WA
1	20in Monitor	109.99	MA
1	20in Monitor	109.99	OR
1	20in Monitor	109.99	GA
1	20in Monitor	109.99	TX
1	20in Monitor	109.99	NY
1	20in Monitor	109.99	ME
1	20in Monitor	109.99	CA
2	27in 4K Gaming Monitor	389.99	WA
2	27in 4K Gaming Monitor	389.99	ME
2	27in 4K Gaming Monitor	389.99	CA
2	27in 4K Gaming Monitor	389.99	OR
2	27in 4K Gaming Monitor	389.99	NY
2	27in 4K Gaming Monitor	389.99	TX
2	27in 4K Gaming Monitor	389.99	GA

2	27in 4K Gaming Monitor	389.99	MA

```sql
/* CREATING STATE SALES TAX TABLE */

CREATE TABLE #Sales_Tax( address_state CHAR(2), sales_tax DECIMAL(3,2));

INSERT INTO #Sales_Tax (address_state, sales_tax) VALUES ('CA', 7.25)

INSERT INTO #Sales_Tax (address_state, sales_tax) VALUES ('GA', 4)

INSERT INTO #Sales_Tax (address_state, sales_tax) VALUES ('MA', 6.25)

INSERT INTO #Sales_Tax (address_state, sales_tax) VALUES ('ME', 5.5)

INSERT INTO #Sales_Tax (address_state, sales_tax) VALUES ('NY', 4)

INSERT INTO #Sales_Tax (address_state, sales_tax) VALUES ('OR', 0)

INSERT INTO #Sales_Tax (address_state, sales_tax) VALUES ('TX', 6.25)

INSERT INTO #Sales_Tax (address_state, sales_tax) VALUES ('WA', 6.5)

SELECT *
FROM #Sales_Tax;
```

address_state	sales_tax
CA	7.25
GA	4
MA	6.25
ME	5.5
NY	4
OR	0
TX	6.25
WA	6.5

```
 -- DETERMINING THE ACTUAL COST OF EACH product ADDING THE STATE SALES TAX TO THE PRICE --
 -- USING CTEs, CASE STATEMENT, JOINS, AND TEMP TABLE --

 -- WHEN ADDRESS STATE FROM CTE_1 MATCHES THE ADDRESS STATE IN THE SALES TAX TEMP TABLE, THE EQUATION INPUTS THE CORRESPONDING STATE
 TAX TO RETURN THE ACTUAL COST OF THE product
 --

WITH CTE_price AS
       (SELECT DISTINCT p.product_id,
                        p.product,
                        p.unit_price,
                        c.address_state
       FROM product AS p
       JOIN sales AS s
       ON p.product = s.product
       JOIN customers AS c
       ON s.purchase_address = c.purchase_address),
CTE_tax AS
       (SELECT product_id,
        product,
        unit_price,
        t.sales_tax,
        t.address_state,
        CASE
           WHEN CTE_price.address_state = t.address_state
              THEN ROUND((unit_price+((t.sales_tax*unit_price)/100)),2)
           ELSE 0
        END AS actual_cost
        FROM CTE_price
        JOIN #Sales_Tax AS t
        ON t.address_state = CTE_price.address_state)
SELECT *
FROM CTE_tax
ORDER BY address_state, product_id;* only CA and GA shown below
```

product_id	product	unit_price	sales_tax	address_state	actual_cost
1	20in Monitor	109.99	7.25	CA	117.96
2	27in 4K Gaming Monitor	389.99	7.25	CA	418.26
3	27in FHD Monitor	149.99	7.25	CA	160.86
4	34in Ultrawide Monitor	379.99	7.25	CA	407.54
5	AA Batteries (4-pack)	3.84	7.25	CA	4.12
6	AAA Batteries (4-pack)	2.99	7.25	CA	3.21

7	Apple Airpods Headphones	150	7.25	CA	160.88
8	Bose SoundSport Headphones	99.99	7.25	CA	107.24
9	Flatscreen TV	300	7.25	CA	321.75
10	Google Phone	600	7.25	CA	643.5
11	iPhone	700	7.25	CA	750.75
12	LG Dryer	600	7.25	CA	643.5
13	LG Washing Machine	600	7.25	CA	643.5
14	Lightning Charging Cable	14.95	7.25	CA	16.03
15	Macbook Pro Laptop	1700	7.25	CA	1823.25
16	ThinkPad Laptop	999.99	7.25	CA	1072.49
17	USB-C Charging Cable	11.95	7.25	CA	12.82
18	Vareebadd Phone	400	7.25	CA	429
19	Wired Headphones	11.99	7.25	CA	12.86
1	20in Monitor	109.99	4	GA	114.39
2	27in 4K Gaming Monitor	389.99	4	GA	405.59
3	27in FHD Monitor	149.99	4	GA	155.99
4	34in Ultrawide Monitor	379.99	4	GA	395.19
5	AA Batteries (4-pack)	3.84	4	GA	3.99
6	AAA Batteries (4-pack)	2.99	4	GA	3.11
7	Apple Airpods Headphones	150	4	GA	156
8	Bose SoundSport Headphones	99.99	4	GA	103.99
9	Flatscreen TV	300	4	GA	312
10	Google Phone	600	4	GA	624
11	iPhone	700	4	GA	728
12	LG Dryer	600	4	GA	624
13	LG Washing Machine	600	4	GA	624
14	Lightning Charging Cable	14.95	4	GA	15.55
15	Macbook Pro Laptop	1700	4	GA	1768
16	ThinkPad Laptop	999.99	4	GA	1039.99

17	USB-C Charging Cable	11.95	4	GA	12.43
18	Vareebadd Phone	400	4	GA	416
19	Wired Headphones	11.99	4	GA	12.47

```
/* CREATING A MEMBERSHIP REWARDS TABLE */

CREATE TABLE #Rewards ( MembershipType CHAR(255), OverallDiscountPct INT,
OrdersGreaterthan100 INT, OrdersGreaterthan500 INT);


INSERT INTO #Rewards (MembershipType, OverallDiscountPct, OrdersGreaterthan100, OrdersGreaterthan500)
VALUES ('Silver', 5, 5, 10);

INSERT INTO #Rewards (MembershipType, OverallDiscountPct, OrdersGreaterthan100, OrdersGreaterthan500)
VALUES ('Gold', 7, 10, 12);

INSERT INTO #Rewards (MembershipType, OverallDiscountPct, OrdersGreaterthan100, OrdersGreaterthan500)
VALUES ('VIP', 10, 12, 15);

SELECT * FROM #Rewards;
```

MembershipType	OverallDiscountPct	OrdersGreaterthan100	OrdersGreaterthan500
Silver	5	5	10
Gold	7	10	12
VIP	10	12	15

```sql
-- DETERMINING REWARDS MEMBERSHIP CANDIDACY BASED ON YEAR END PURCHASE TOTALS FOR EACH CUSTOMER (EXCLUDES SALES TAX) --

SELECT c.customer_id,
       SUM(s.qty_ordered*s.price_each) AS 'year_end_totals',
       CASE
          WHEN SUM(s.qty_ordered*s.price_each) > 2500 THEN NCHAR(10004)
          -- NCHAR(10004) IS CHECKMARK --
          ELSE NCHAR(10008) -- NCHAR(10008) IS 'X' MARK –
       END AS VIP,
       CASE
          WHEN SUM(s.qty_ordered*s.price_each) >= 1000 THEN NCHAR(10004) ELSE NCHAR(10008)
       END AS Gold, CASE
          WHEN SUM(s.qty_ordered*s.price_each) >= 500 THEN NCHAR(10004)
          ELSE NCHAR(10008)
       END AS Silver
FROM customers AS c
JOIN sales AS s
ON c.purchase_address = s.purchase_address
GROUP BY c.customer_id
ORDER BY c.customer_id;
```

* only first 25 rows shown below

customer_id	year_end_totals	VIP	Gold	Silver
100000	11.95	✗	✗	✗
100001	1700	✗	✓	✓
100002	700	✗	✗	✓
100003	150	✗	✗	✗
100004	11.99	✗	✗	✗
100005	1711.99	✗	✓	✓
100006	11.99	✗	✗	✗
100007	114.94	✗	✗	✗
100008	1723.9	✗	✓	✓
100009	7.68	✗	✗	✗
100010	150	✗	✗	✗
100011	11.99	✗	✗	✗
100012	23.9	✗	✗	✗
100013	11.99	✗	✗	✗
100014	99.99	✗	✗	✗
100015	300	✗	✗	✗
100016	389.99	✗	✗	✗
100017	8.97	✗	✗	✗
100018	38.85	✗	✗	✗
100019	2.99	✗	✗	✗
100020	11.99	✗	✗	✗
100021	3.84	✗	✗	✗
100022	323.9	✗	✗	✗
100023	600	✗	✗	✓
100024	99.99	✗	✗	✗
100025	165.35	✗	✗	✗