Table of Contents

Project Scenario

Furniture companies have great competitiveness that have different forms of selling products through in-store, online, wholesale etc. The group looks for a dataset that contains information about selling furniture. This project is aiming to analyze this dataset and compute a most competitive form of furniture selling over the U.S. region.

The dataset we choose is "US Regional Sales data" from data.world - the fictitious sales data for a certain company across the US regions. The dataset contains information about sales, customers, store locations, products, regions, and sales teams. Specifically, it has over 8,000 rows and 40 columns. All information is stored in an excel spreadsheet, which is a perfect fit for demonstrating an advantageous transition process from a spreadsheet to a database management system. A database system fabricates an organized information collection from the traditional information stored in a spreadsheet in a way that businesses stay organized and keep information easily accessible. Ultimately, it saves time and effort to retrieve and store data as the business grows. Dataset accommodates sufficient and detailed row data, which is supportive of this project. The process of making the transition happen benefits applying class knowledge to a real-world business circumstance.

Since the dataset that the group chose is pretty comprehensive, which included multiple parts in the selling products process. The comprehensive information allows the group to analyze the relationship between each combination. The preliminary idea of relations between attributes relates to sales and orders, orders and products, customers and orders, sales and locations, sales and sales teams, or relations after the group demonstrates more profound for this project.

- The initial plan of Transition
- The foremost first step is to look through the dataset and understand the data
- Normalize unstructured data (1NF, 2NF, 3NF)
- Design relational database schema using ER diagrams
- Transform and load the data to the database system by using Python
- Explanation of the importance of each step

A database management system helps facilitate data quality activities by providing a structure. Higher-quality data is generated through improved data management methods, which leads to better decision-making across an organization. This project analysis of the U.S. region sales dataset can help create a simulated framework for retailers or wholesalers to realize how to make decisions on targeting customers with consumer preferences and regional sales. In

addition to better decision-making, DBMS maintenance is consistent and creates reliability in data management. Thus, increasing productivity.

The goal of the project is to help this e-commerce company based in the US analyze its sales performance to generate insights and create strategies to increase its revenue. First, we can look into the seasonality of the sales throughout the year based on regions and different products sold to help plan and allocate inventories accordingly. With this information, the company could learn what the most popular products are and focus on advertising them.
Second, we can analyze customer profiles. For example, what proportion of the customers are repeat customers, and from there, the company can tailor its strategy accordingly and decide whether to focus on attracting new customers or to establish a royalty program to reinforce its repeat customers.

Moreover, we can analyze which sales channel generates the most revenue and the company will be able to focus on and invest in those channels for advertising purposes. Same when it comes to sales teams, from the data, we will be able to analyze which sales team generates the most and least revenue. With this information, the company will be able to identify areas to improve, better train the sales teams, and optimize sales performance.

Team Contract

1NF

Sales_order

OrderNumber	Sales Channel	WarehouseCode	ProcuredDate	OrderDate	ShipDate	DeliveryDate
CurrencyCode	_SalesTeamID	_CustomerID	_StoreID	_ProductID	Order Quantity	Discount Applied
Customer Names	City Name	County	StateCode	State	Type	Latitude
Longitude	AreaCode	Population	Household Income	Median Income	Land Area	Water Area
Time Zone	Product Name	Region	Sales Team			

The group separated the 1NF forms into 2NF forms by detailed tables based on 2NF rules: every non-key attribute must be fully dependent on the primary key.

2NF

Store_location'

_StoreID	City Name	County	statecode	State	Type	Latitude
Longitude	AreaCode	Population	Household Income	Median Income	Land Area	Water Area
Time Zone						

Customers'

_CustomerID	Customer Names

State'

StateCode	State	Region

Salesteam'

_SalesTeamID	Sales Team	Region

Product'

_ProductID	Product Name

Sales_order'

OrderNumber	Sales Channel	WarehouseCode	ProcuredDate	OrderDate	ShipDate	DeliveryDate
CurrencyCode	Order Quantity	Discount Applied	Unit Price	Unit Cost		

Order'

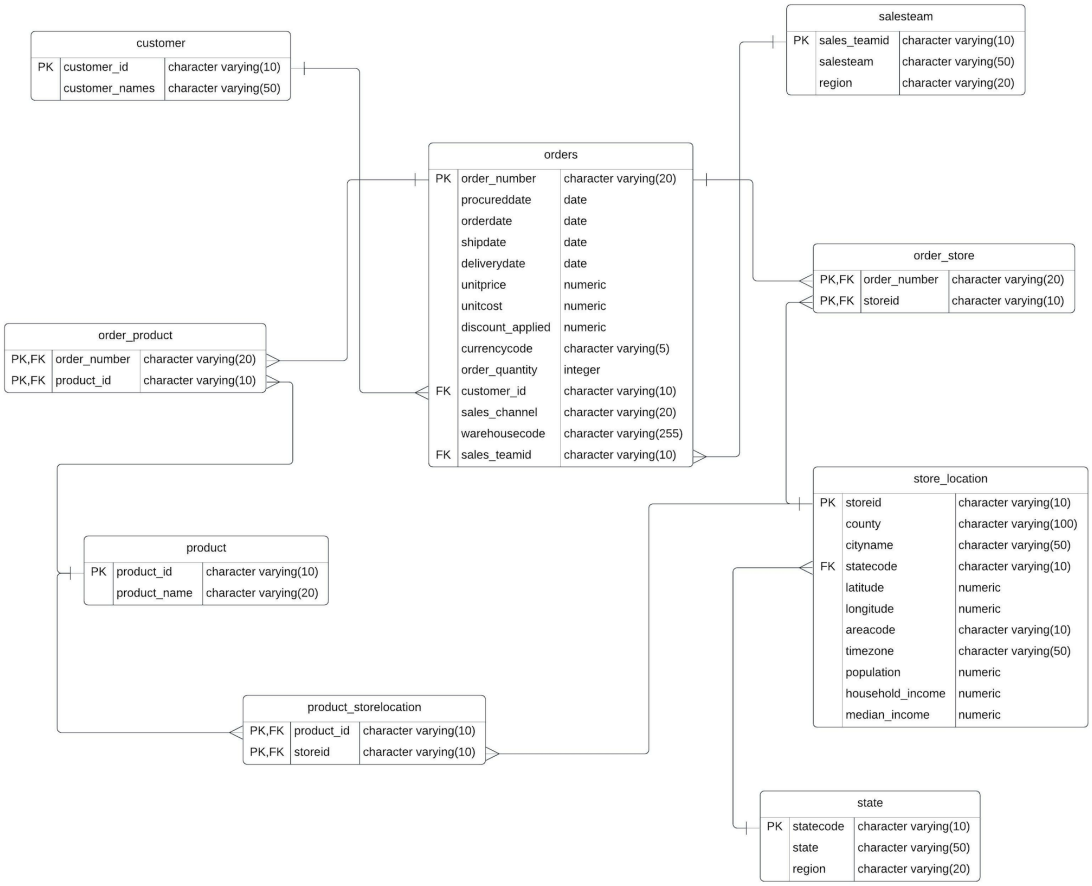
OrderNumber	_SalesTeamID	_CustomerID	_StoreID	_ProductID

After second normalization, we separated all attributes based on the requirement of 3NF which is that all attributes only depend on the key.

3NF						
Product'						
Productid (PK)	Product_name					
State'						
statecode (PK)	state	region				
Store_location'						
StoreID (PK)	City Name	County	Time Zone	Population	Household Income	Median Income
Statecode (FK)	Latitude	Longitude	Areacode			
Customer'						
CustomerID (PK)	customer_names					
Salesteam'						
Salesteam_id (PK)	Salesteam	region				
Order'						
OrderNumber(PK)	Sales Channel	WarehouseCode	ProcuredDate	OrderDate	ShipDate	DeliveryDate
CurrencyCode	Order Quantity	Discount Applied	Unit Price	Unit Cost	CustomerID (FK)	Salesteam_id (FK)
Order-store'						
OrderNumber(PK, FK)	_StoreID(PK,FK)					
Order-Product'						
OrderNumber(PK, FK)	Productid(PK, FK)					
Product_storelocation'						
Productid (PK,FK)	_StoreID(PK,FK)					

SQL code for the normalized tables:
https://drive.google.com/drive/folders/1B427hsjSOxSR_WrfaTl3Et3T0VhTHU9i?usp=sharing

Lucidchart Link:
https://lucid.app/lucidchart/c08f1cf5-abb3-468c-8e26-11eab3f43d0f/edit?invitationId=inv_e7e7915b-f812-481e-a725-38425cc89e3e



ETL

- Present ETL process *in detail*.

  Github Link:
  https://github.com/yyyukeqi/5310-SQL-Project/blob/main/ETL%20Process

- Explain all work for normalization and ETL.

For normalization, in the 3NF, we have 9 tables - state, salesteam, product_storelocation, order_product, product, customer, orders, store_location, and order_store. We split the tables this way based on the relationships between different relations and the data analysis we plan to

do with the relationships.

3NF

Product'

Productid (PK)	Product_name

State'

statecode (PK)	state	region

Store_location'

StoreID (PK)	City Name	County	Time Zone	Population	Household Income	Median Income	Statecode (FK)	Latitude	Longitude	Areacode

Customer'

CustomerID (PK)	customer_names

Salesteam'

Salesteam_id (PK)	Salesteam	region

Order'

OrderNumber(PK)	Sales Channel	WarehouseCode	ProcuredDate	OrderDate	ShipDate	DeliveryDate	CurrencyCode	Order Quantity	Discount Applied	Unit Price	Unit Cost	CustomerID (FK)	Salesteam_id (FK)

Order-store

OrderNumber(PK, FK)	_StoreID(PK,FK)

Order-Product

OrderNumber(PK, FK)	Productid(PK, FK)

Product_storelocation

Productid (PK,FK)	_StoreID(PK,FK)

For ETL, first, we connected Python to the PostgreSQL database, and then we created all our tables with primary key and foreign key constraints, and loaded the data in a csv file into different tables.
With the database and all tables created (3NF), it is now time to extract, transform and load (ETL) the dataset into the database. In order to do so we will have to perform several data transformations on the loaded dataframe, `df` in order to create all primary keys and maintain proper relationships.
Since our dataset already contains a subset of Regions, we simply load state data to the database. In addition, all column names we have in the database are different from our dataset, so we need to rename them before loading into the database.

```
#table state
df = pd.read_excel(r'US_Regional_Sales_Data.xlsx', sheet_name='Regions Sheet')
state = df[['StateCode', 'State','Region']]
state.rename(columns={'State':'state', 'Region':'region', 'StateCode':'statecode'},inplace=True)
state.head()
```

	statecode	state	region
0	AL	Alabama	South
1	AR	Arkansas	South
2	AZ	Arizona	West
3	CA	California	West
4	CO	Colorado	West

```
state.to_sql(name='state', con=engine, if_exists='append', index=False)
```

Also, since our dataset already contains a subset of Sales Team, Product, Customers, Store Locations, Orders, we simply load sales team, product, customers, store_location, orders data to

the database. Same as above, we rename column names before loading into the database. In addition, for pushing the database, we need to lower the case of all column names of the database.

```
#table salesteam
df = pd.read_excel(r'US_Regional_Sales_Data.xlsx', sheet_name='Sales Team Sheet')
salesteam = df[['_SalesTeamID','Sales Team', 'Region']]
salesteam.rename(columns={'_SalesTeamID':'sales_teamid', 'Sales Team':'salesteam' , 'Region':'region'},inplace=True)
salesteam.head()
```

	sales_teamid	salesteam	region
0	1	Adam Hernandez	Northeast
1	2	Keith Griffin	Northeast
2	3	Jerry Green	West
3	4	Chris Armstrong	Northeast
4	5	Stephen Payne	South

```
salesteam.to_sql(name='salesteam', con=engine, if_exists='append', index=False)
```

```
#table product
df = pd.read_excel(r'US_Regional_Sales_Data.xlsx', sheet_name='Products Sheet')
product = df[['_ProductID', 'Product Name']]
product.rename(columns={'_ProductID': 'Product_ID', 'Product Name': 'Product_name'},inplace=True)
product.columns = product.columns.str.lower()
product.head()
```

	product_id	product_name
0	1	Cookware
1	2	Photo Frames
2	3	Table Lamps
3	4	Serveware
4	5	Bathroom Furniture

```
product.to_sql(name='product', con=engine, if_exists='append', index=False)
```

```
#table customers
df = pd.read_excel(r'US_Regional_Sales_Data.xlsx', sheet_name='Customers Sheet')
customer = df[['_CustomerID', 'Customer Names']]
customer.rename(columns={'_CustomerID': 'Customer_ID', 'Customer Names': 'Customer_Names'},inplace=True)
customer.columns = customer.columns.str.lower()
customer.head()
```

	customer_id	customer_names
0	1	Avon Corp
1	2	WakeFern
2	3	Elorac, Corp
3	4	ETUDE Ltd
4	5	Procter Corp

```
customer.to_sql(name='customer', con=engine, if_exists='append', index=False)
```

```
#table store_location
df = pd.read_excel(r'US_Regional_Sales_Data.xlsx', sheet_name='Store Locations Sheet')
store_location = df[['_StoreID', 'County', 'City Name', 'StateCode', 'Latitude', 'Longitude', 'AreaCode', 'Time Zone',
store_location.rename(columns={'_StoreID': 'StoreID', 'City Name': 'CityName', 'Time Zone' : 'TimeZone', 'Household Inc
store_location.columns = store_location.columns.str.lower()
store_location.head()
```

	storeid	county	cityname	statecode	latitude	longitude	areacode	timezone	population	household_income	median_income
0	1	Shelby County/Jefferson County	Birmingham	AL	33.52744	-86.79905	205	America/Chicago	212461	89972	31061
1	2	Limestone County/Madison County	Huntsville	AL	34.69901	-86.67298	256	America/Chicago	190582	78554	48775
2	3	Mobile County	Mobile	AL	30.69436	-88.04305	251	America/Chicago	194288	76170	38776
3	4	Montgomery County	Montgomery	AL	32.36681	-86.29997	334	America/Chicago	200602	79866	42927
4	5	Pulaski County	Little Rock	AR	34.74648	-92.28959	501	America/Chicago	197992	79902	46085

```
store_location.to_sql(name='store_location', con=engine, if_exists='append', index=False)
```

```
#table5 orders
df = pd.read_excel(r'US_Regional_Sales_Data.xlsx', sheet_name='Sales Orders Sheet')
orders = df[['OrderNumber','ProcuredDate','OrderDate','ShipDate','DeliveryDate','Unit Price','Unit Cost',\
             'Discount Applied','CurrencyCode','Order Quantity', '_CustomerID','Sales Channel','WarehouseCode',\
            '_SalesTeamID']]

orders.rename(columns={'OrderNumber': 'order_number','_ProductID': 'product_id','Discount Applied':'discount_applied',
                       'Sales Channel':'sales_channel','_CustomerID':'customer_id',
                       'Unit Price':'unitprice','Unit Cost':'unitcost',
                       '_SalesTeamID':'Sales_TeamID','Order Quantity':'Order_quantity'},inplace=True)
orders.columns=orders.columns.str.lower()

orders.head()
```

For bridge table order_product, we only select order_number and product_id from Sales Order dataset. We can now assign these two attributes as composite keys and foreign key as well. We do not need to remove duplicates because order_number is already unique.

```
#table order_product
df = pd.read_excel(r'US_Regional_Sales_Data.xlsx', sheet_name='Sales Orders Sheet')
order_product = df[['OrderNumber', '_ProductID']]
order_product.rename(columns={'OrderNumber': 'Order_number', '_ProductID': 'Product_ID'},inplace=True)
order_product.columns = order_product.columns.str.lower()
order_product.head()
```

Moreover, for bridge table product_storelocation, we only select product_id and storeID from Sales Order dataset. In order to assign these two attributes as composite key as well, we need to drop duplicate rows and push to the database.

```
#table product_storelocation
df = pd.read_excel(r'US_Regional_Sales_Data.xlsx', sheet_name='Sales Orders Sheet')
product_storelocation = df[['_StoreID', '_ProductID']]
product_storelocation.rename(columns={'_StoreID': 'storeID', '_ProductID': 'Product_ID'},inplace=True)
product_storelocation = product_storelocation.drop_duplicates()
product_storelocation.columns = product_storelocation.columns.str.lower()
product_storelocation.head()
```

Last but not least, for bridge table order_store, we only select order_number and storeID from Sales Order dataset. We do not need to remove duplicates because order_number is already unique.

```
#table order_store
df = pd.read_excel(r'US_Regional_Sales_Data.xlsx', sheet_name='Sales Orders Sheet')
order_store = df[['OrderNumber', '_StoreID']]
order_store.rename(columns={'OrderNumber': 'Order_number', '_StoreID': 'storeID'},inplace=True)
order_store.columns = order_store.columns.str.lower()
order_store.head()
```

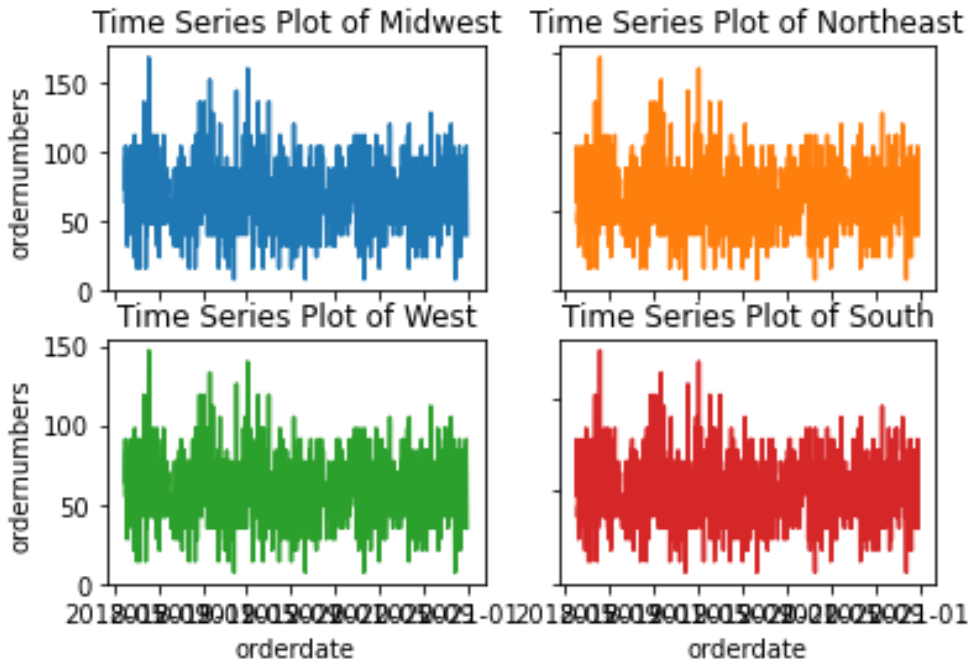
Analytical Procedures

1. What is the seasonality of the sales throughout a year in different regions, if any?

```
queryCmd = """SELECT o.orderdate, count(o.order_number) as number, s.region
            FROM orders o, salesteam s
            WHERE s.region = 'Midwest'
            GROUP BY orderdate, s.region
            ORDER BY orderdate DESC;
        """
cur.execute(queryCmd)
conn.commit()

results = connection.execute(queryCmd).fetchall()
column_names = results[0].keys()
df1 = pd.DataFrame(results, columns = column_names)
df1
queryCmd2 = """SELECT o.orderdate,
                count(o.order_number) as number, s.region
            FROM orders o, salesteam s
            WHERE s.region = 'Northeast'
            GROUP BY orderdate, s.region
            ORDER BY orderdate DESC;
        """
cur.execute(queryCmd2)
conn.commit()

results = connection.execute(queryCmd2).fetchall()
column_names = results[0].keys()
df2 = pd.DataFrame(results, columns = column_names)
df2
```

```
queryCmd3 = """SELECT o.orderdate, count(o.order_number) as number, s.region
            FROM orders o, salesteam s
            WHERE s.region = 'West'
            GROUP BY orderdate, s.region
            ORDER BY orderdate DESC;
        """
cur.execute(queryCmd3)
conn.commit()
results = connection.execute(queryCmd3).fetchall()
column_names = results[0].keys()
df3 = pd.DataFrame(results, columns = column_names)
df3

queryCmd4 = """SELECT o.orderdate, count(o.order_number) as number, s.region
            FROM orders o, salesteam s
            WHERE s.region = 'South'
            GROUP BY orderdate, s.region
            ORDER BY orderdate DESC;
        """
cur.execute(queryCmd4)
conn.commit()

results = connection.execute(queryCmd4).fetchall()
column_names = results[0].keys()
df4 = pd.DataFrame(results, columns = column_names)
df4

import matplotlib.pyplot as plt
fig, axs = plt.subplots(2, 2)
axs[0, 0].plot(df1["orderdate"],df1["number"])
axs[0, 0].set_title('Time Series Plot of Midwest')
axs[0, 1].plot(df2["orderdate"],df2["number"], 'tab:orange')
axs[0, 1].set_title('Time Series Plot of Northeast')
axs[1, 0].plot(df3["orderdate"],df3["number"], 'tab:green')
axs[1, 0].set_title('Time Series Plot of West')
axs[1, 1].plot(df4["orderdate"],df4["number"], 'tab:red')
axs[1, 1].set_title('Time Series Plot of South')
```

```
for ax in axs.flat:
    ax.set(xlabel='orderdate', ylabel='ordernumbers')

# Hide x labels and tick labels for top plots and y ticks for right plots.
for ax in axs.flat:
    ax.label_outer()
```



2. What are the top ten best selling products (in terms of order quantity) in each year?

```
#2018:
stmt = """ select orders.order_number, orders.orderDate,
                product.product_name
        from orders, product
        where EXTRACT(YEAR FROM orders.orderdate)='2018';
"""
#Execute the statement and get the results from the database
results = connection.execute(stmt).fetchall()

#Extract column names
column_names = results[0].keys()
```

```
#Store results from the database in a panda DataFrame
df3 = pd.DataFrame(results, columns=column_names)
```

```
df3.groupby('product_name').size()
```

```
product_name
Accessories           1836
Audio                 1836
Bakeware              1836
Bar Tools             1836
Baseball              1836
Basketball            1836
Bathroom Furniture    1836
Bean Bags             1836
Bedroom Furniture     1836
Blankets              1836
```

```
#2019:
stmt = """ select orders.order_number, orders.orderDate,
            product.product_name
        from orders, product
        where EXTRACT(YEAR FROM orders.orderdate)='2019';
"""

#Execute the statement and get the results from the database
results = connection.execute(stmt).fetchall()

#Extract column names
column_names = results[0].keys()
#Store results from the database in a panda DataFrame
df4 = pd.DataFrame(results, columns=column_names)
```

```
df4.groupby('product_name').size()
```

```
product_name
Accessories           1836
Audio                 1836
Bakeware              1836
Bar Tools             1836
Baseball              1836
Basketball            1836
Bathroom Furniture    1836
Bean Bags             1836
Bedroom Furniture     1836
Blankets              1836
```

```
#2020
stmt = """ select orders.order_number, orders.orderDate,
                product.product_name
          from orders, product
          where EXTRACT(YEAR FROM orders.orderdate)='2020';
"""
#Execute the statement and get the results from the database
results = connection.execute(stmt).fetchall()
```

```
df5.groupby('product_name').size()
```

```
product_name
Accessories           3125
Audio                 3125
Bakeware              3125
Bar Tools             3125
Baseball              3125
Basketball            3125
Bathroom Furniture    3125
Bean Bags             3125
Bedroom Furniture     3125
Blankets              3125
```

3. What proportion of the customers ordered over 165?

```
SELECT COUNT(DISTINCT customer_id) FROM orders
#50

WITH a AS  (SELECT customer_id AS loyalist FROM orders GROUP BY customer_id
HAVING COUNT(order_number) > 165)
SELECT ROUND(COUNT(loyalist)/(1.00*50), 2) AS proportion
FROM a
```

```
#0.3
```

4. Which sales team performs the best (involved in most sales) in each year?

```
#2018:
SELECT s.sales_teamid, s.salesteam, COUNT(o.order_number) AS number_sales,
o.orderdate
FROM salesteam s, orders o
```

```
WHERE date_part('year',orderdate) = 2018
GROUP BY s.sales_teamid,o.orderdate
ORDER BY number_sales DESC, s.sales_teamid ASC;
```

```
#sales_teamid salesteam       number_sales  orderdate
0       1       Adam Hernandez          21    2018-08-04
1       10      Jonathan Hawkins        21    2018-08-04
2       11      Joshua Little       21    2018-08-04
3       12      Carl Nguyen         21    2018-08-04
4       13      Todd Roberts         21   2018-08-04
```

```
#2019:
SELECT s.sales_teamid, s.salesteam, COUNT(o.order_number) AS number_sales,
o.orderdate
FROM salesteam s, orders o
WHERE date_part('year',orderdate) = 2019
GROUP by s.sales_teamid,o.orderdate
ORDER BY number_sales DESC, s.sales_teamid ASC;
```

```
sales_teamid   salesteam       number_sales  orderdate
0       1       Adam Hernandez          20          2019-05-04
1       10      Jonathan Hawkins        20          2019-05-04
2       11      Joshua Little           20          2019-05-04
3       12      Carl Nguyen             20          2019-05-04
4       13      Todd Roberts            20          2019-05-04
```

```
#2020:
SELECT s.sales_teamid, s.salesteam, COUNT(o.order_number) AS number_sales,
o.orderdate
FROM salesteam s, orders o
WHERE date_part('year',orderdate) = 2020
GROUP BY s.sales_teamid,o.orderdate
ORDER BY number_sales DESC, s.sales_teamid ASC;
```

	sales_teamid	salesteam	number_sales	orderdate
0	1	Adam Hernandez	16	2020-09-21
1	10	Jonathan Hawkins	16	2020-09-21
2	11	Joshua Little	16	2020-09-21
3	12	Carl Nguyen	16	2020-09-21
4	13	Todd Roberts	16	2020-09-21

5. What are the five store locations that have the most orders each year?

```
SELECT COUNT (order_number) AS total_order, s.storeid,
FROM orders
JOIN order_product using (order_number)
JOIN product using (product_id)
JOIN product_storelocation using (product_id)
JOIN store_location s using (storeid)
GROUP BY s.storeid
ORDER BY total_order DESC
LIMIT 5;
```

#	total_order	storeid
0	4645	262
1	4601	21
2	4561	26
3	4489	128
4	4414	245

6. Which sales channel has generated the most profits?

```
stmt = """
SELECT sales_channel, unitprice, unitcost
FROM orders;
"""

#Execute the statement and get the results from the database
results = connection.execute(stmt).fetchall()

#Extract column names
column_names = results[0].keys()
```

```
#Store results from the database in a panda DataFrame
df = pd.DataFrame(results, columns=column_names)
df['profit']=df['unitprice']-df['unitcost']
df.head()
df.groupby('sales_channel').sum().sort_values(['profit'],ascending=False)
#In-store has generated the most profit
```

sales_channel	unitprice	unitcost	profit
In-Store	7487169.60000	4698862.02300	2788307.57700
Online	5544846.30000	3479394.42000	2065451.88000
Distributor	3201762.50000	2002160.33000	1199602.17000
Wholesale	2021952.80000	1261984.46300	759968.33700

7. Which state has the most stores?

```
WITH a AS (SELECT COUNT(storeID) AS number_of_stores, StateCode
FROM store_location
GROUP BY StateCode)
SELECT StateCode, number_of_stores
FROM a
WHERE number_of_stores = (SELECT MAX(number_of_stores) FROM a);
```

```
"CA"   74
```

8. Which region has generated the most profits?

```
stmt = """
SELECT orders.unitprice, orders.unitcost, salesteam.region
FROM orders
JOIN salesteam ON orders.sales_teamid=salesteam.sales_teamid;
"""
#Execute the statement and get the results from the database
results = connection.execute(stmt).fetchall()
#Extract column names
column_names = results[0].keys()
#Store results from the database in a panda DataFrame
df2 = pd.DataFrame(results, columns=column_names)
```

```
df2.head()
```

9. What are the proportions of sales channels accounted for total orders?

```
WITH total AS
(SELECT sales_channel, COUNT(order_number) n
FROM orders
GROUP BY sales_channel),

tn AS (SELECT COUNT(order_number) t FROM orders)

SELECT sales_channel, round((1.00*n)/(1.00*t),2) AS percentage
FROM total, tn
```

```
"Online" 0.30
"Wholesale" 0.11
"In-Store" 0.41
"Distributor" 0.17
```

10. Which product is the most popular among all the records?

```
SELECT COUNT (order_number) AS total_order, p.product_id, p.product_name
FROM orders
JOIN order_product using (order_number)
JOIN product p using (product_id)
GROUP BY p.product_id
ORDER BY total_order DESC
LIMIT 1;
```

```
  total_order   product_id      product_name
0        200           37                 Platters
```

- Coding
  https://github.com/yyyukeqi/5310-SQL-Project/blob/main/Analytical%20Procedures

Analyst vs. Executive: Methodology and Tools

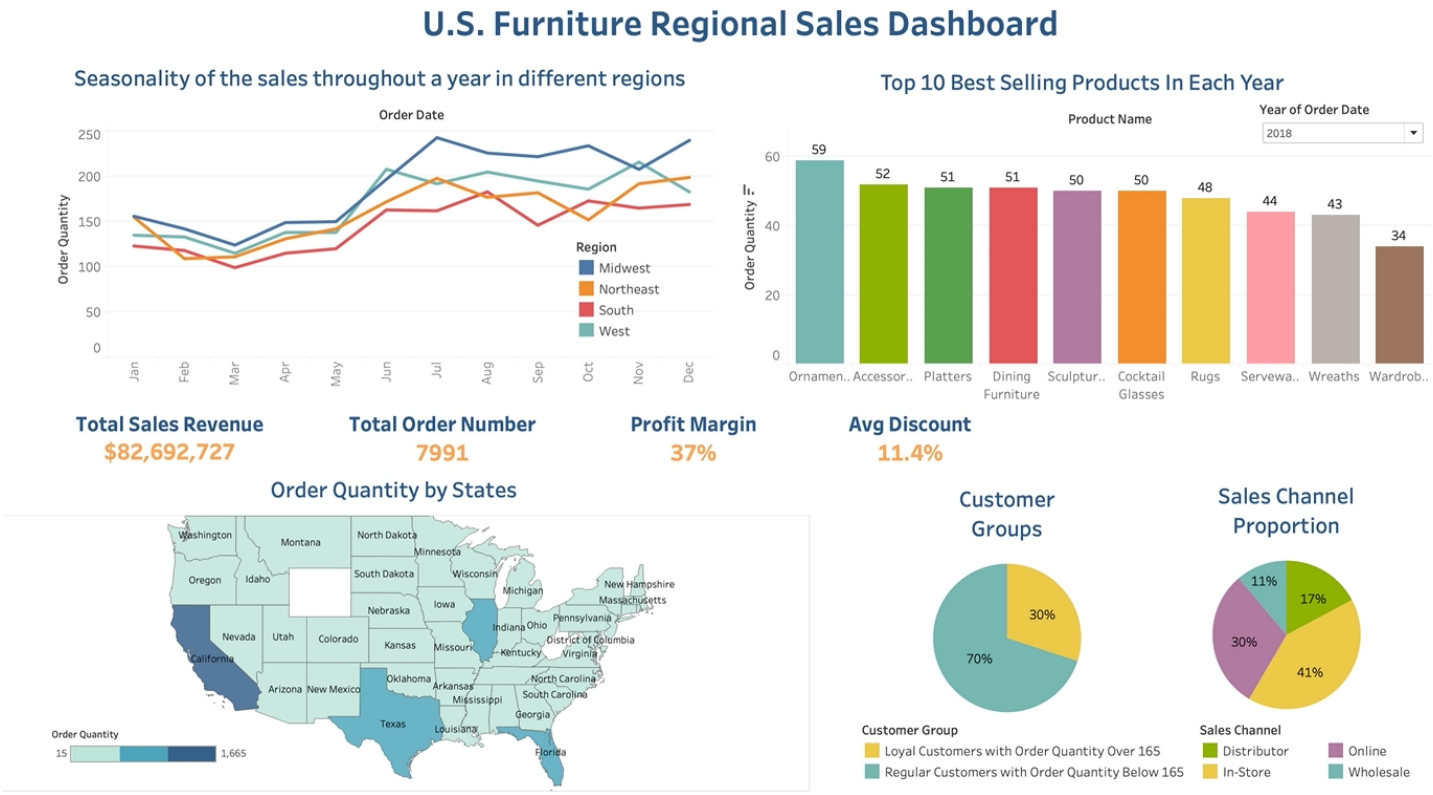
Relating to the project proposal, our primary goal is to create a simulated framework to guide on customers targeting decision-making for retailers or wholesalers by analyzing the U.S region sales dataset. To accomplish our goal, we will combine both direct querying implementation and data visualization report of the designed database system to different roles. Specifically, we will provide a query-driven report to show the facts of our sales regional performance and answer some proposed questions based on the dataset to help make decisions on targeting customers for middle management roles. We are using Python, SQL, Excel, and Word to reach the goal. For "C" level officers, we are providing a Tableau and Metabase dashboard to tell the story and give them a visualization about the data without explaining our step-by-step analytical and technical procedures which will help them better understand the relationships. Basing programming languages on database actions improves efficiency and effectiveness by letting developers work on data in memory and persisting changes to storage, intelligently caching data and rolling back changes as required, and dynamically adjusting definitions. On-technical personnel from different roles can interact with the database using different techniques.

In avoiding the event our database crashed (redundancy) or had a lot of traffic or demands on the database that could possibly negatively impact performance, we include database normalization process to ensure redundant data is eliminated and their dependencies are enforced correctly. If our database crashes, we will have a backup procedure of the database to restore the whole dataset. As we design the database system, indexes are included in our databases. We have primary keys and foriegn keys in the tables, and all the tables are related to each other.

In addition, “Horizontal scaling”, which is to add additional servers will help ease the load on individual servers and help answer the server requests. Multiple servers are linked together, communicating with each other to make sure each request is answered and everyone has access to the service. Horizontal scaling also requires assigning the specific responsibility to each certain machine since adding the server might increase the complexity of operation. The hardware of any given server unit doesn’t have to be overly robust, since it doesn’t carry the weight of the traffic by itself. There might be other reasons that cause the redundancy such as low maintenance of the database and wrong servers in some way. In this case, we need to take another approach to solve the problem according to the specific issue that might occur. We could also prevent redundancy and traffic by preventive care and good handling which ensure a longer trouble free life for servers.

The tendency of the data storage is cloud since it will be more convenient and cost-saving. We suggest the company to use the hybrid method which contains on-premise and cloud. If the company is only using on-premise, which means there is a cost running servers internally. Most of the companies that only use shared servers so that they need to reduce their cost at a minimal range. There might be a significant cost for license of the servers while companies are operating them. Cloud servers are managed by the third-party and they will be using certain applications which require technical skills that support requests from end users. If the company is combining both on-premise and cloud, it would allow the company to share the data both internally and externally with lower cost. In this way, the company can have the research and share the data in house and provide an ideal solution for outside of the organization.

Tableau Dashboard



Conclusion

Based on the analysis, we have few conclusions that we want to illustrate.

First, the benefit of RDMS provide clients the ability to create meaningful information by joining the tables, and analysis using PostgresSQL functions and logical transformations. Indeed, RDMS provides question insights such as what region has the most sales? What type of products further with particular products and much more. In addition, RDMS eliminates data redundancy. The order table only needs to store a link to the customer table. In other words, normalization. Lastly, relational databases are transactional, ability to remain consistent at any moment, and offer easy export and import options, making backup etc.

Second, there are some benefits of ETL such as facilitating performance of the database and providing a visual flow. It also leverages an existing development framework and provides operations resilience. ETL could handle big data in some way since it enables advanced data profiling and cleansing. Meanwhile, ETL tracks the data lineage and performs impact analysis overall.

Due to our work, It enables us to retrieve an entirely new table from data in one or more tables with a single query. It also allows our business to better understand the relationships among all available data and gain new insights for making better decisions or identifying new opportunities.

Based on the project analysis we analyzed, it helps furniture company to understand the sales pattern (eg. seasonality), top selling products, proportion of returning customers, the best performing sales team, the most profitable sales channel, customer demographics, store locations, region profitability, and so on. Then our client could make better decisions on choosing the most suitable furniture for sale. Furthermore, our client can also better understand the relationship among all available data.