Applied A.I. Solutions
Foundations of Data Management

Final Project

Group-10 members

1. Goyal, Vinayak
2. Bhasgauri, Harshal Shashikant
3. Sebastian, Arun
4. ., Himani
5. Singh, Satyajeet
6. Trongkitroongruang, Kajhonprom
7. Cheng, Qianfan

Table of Contents

1. Sub-domain: Data Modelling and Design

1.1 Principle of Formalization

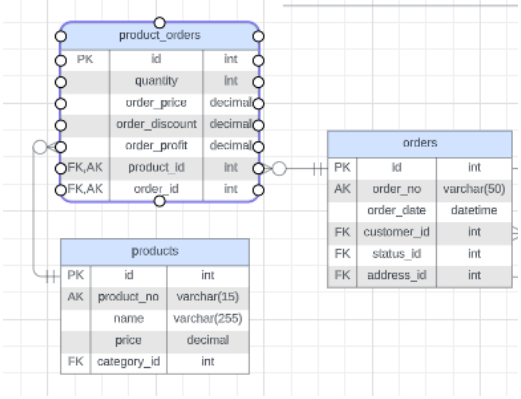
1.1.1 Policy: All entities must be normalized.

1.1.1.1 Procedure

Remove partial dependencies from entities. For example, 'product_name' attribute is solely dependent on 'product_no' attribute, consequently, we take 'product_no', 'product_name', 'price', and 'category_id' into a new entity 'products'.

1.1.1.2 Procedure

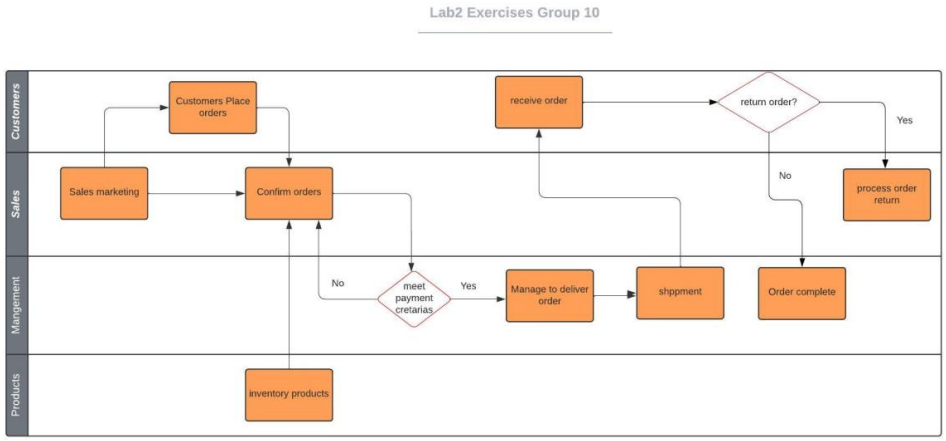
Transitive dependencies are all removed from entities, or many to many relationships are all removed by inserting a composite entity which contain the primary key of both entity as its foreign key. For example, 'orders' entity and 'products' entity have a many-to-many relationship, and after a composite entity is inserted:



1.1.2 Policy: All models must be fully documented with Data Flow Diagram and Entity Relationship Diagram

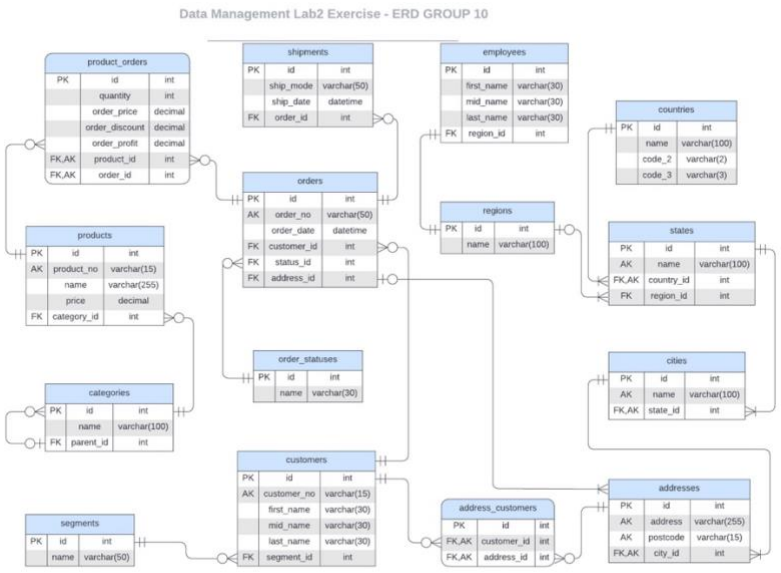
1.1.2.1 Procedure

Data Flow Diagram is created to describe how data flows between different processes and departments.



1.1.2.2 Procedure

Logical level Entity Relationship is created to document entities, attributes and relationships between entities.



1.2 Principle of Knowledge retention/documentation

1.2.1 Policy: Document metadata associated with all data models.

1.2.1.1 Procedure

Metadata entity is created to document entity definitions, including entity definition, attributes, domains, relationships, etc.

metadatas		
PK	id	int
AK	table_name	varchar(50)
AK	column_name	varchar(100)
	data_type	varchar(50)
	description	varchar(255)
	constraints	varchar(255)
	relationships	varchar(255)
	created_at	datetime
	updated_at	datetime

1.2.1.2 Procedure

Assumptions about attributes are documented to have definitions of attributes and rules for further usage.

1.2.2 Policy: Document all changes made to data sources and must be accessible and shared.

1.2.2.1 Procedure

All changes made to datasets are well documented to have better knowledge transfer between team members.

1.2.2.2 Procedure

Data models are properly shared in lucidchart.app.

2. Sub-domain: Data Integration and Interoperability

2.1 Principle of Data Quality Management

2.1.1 Policy: Identify and document all data sources that will be part of the consolidation process.

2.1.1.1 Procedure

There are three documents that are given as data sources, which are 'orders', 'people', and 'returns' sheets. 'orders' sheet contains information about customers, orders, products, addresses. 'people' sheet contains information of managers for regions. 'returns' sheet has information of orders that have been returned.

2.1.1.2 Procedure

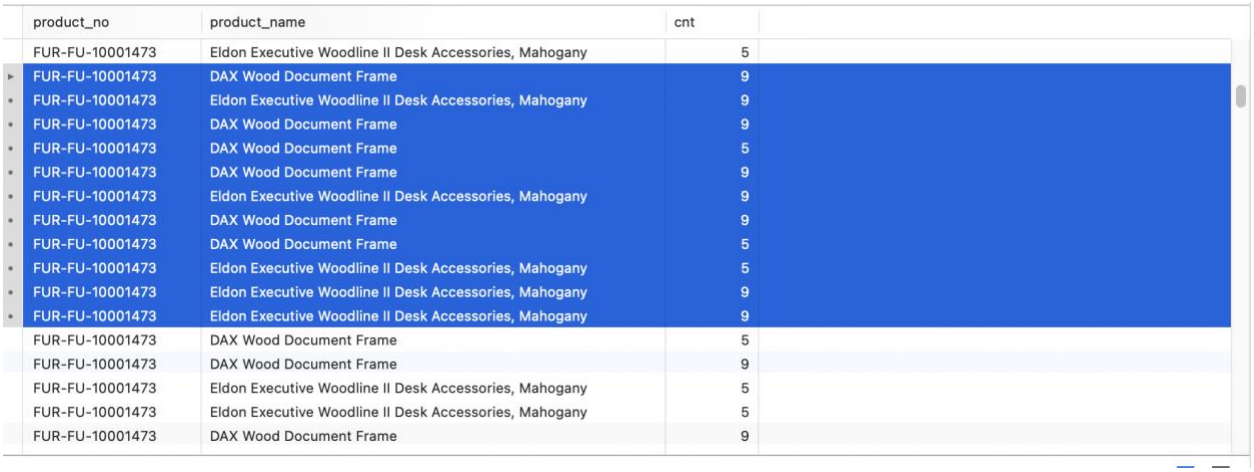
These data sources are analyzed and documented by column information that each sheet contains, existing duplicated rows, missing values, inconsistencies, and redundancies.

2.1.2 Policy: duplicated records must be removed, and values in unique columns must be kept unique.

2.1.2.1 Procedure

When analyzing data sources, 'returns' sheet is identified as contains many duplicated records, and duplicated columns are removed.

2.1.2.2 Procedure



product_no	product_name	cnt
FUR-FU-10001473	Eldon Executive Woodline II Desk Accessories, Mahogany	5
FUR-FU-10001473	DAX Wood Document Frame	9
FUR-FU-10001473	Eldon Executive Woodline II Desk Accessories, Mahogany	9
FUR-FU-10001473	DAX Wood Document Frame	9
FUR-FU-10001473	DAX Wood Document Frame	5
FUR-FU-10001473	DAX Wood Document Frame	9
FUR-FU-10001473	Eldon Executive Woodline II Desk Accessories, Mahogany	9
FUR-FU-10001473	DAX Wood Document Frame	9
FUR-FU-10001473	DAX Wood Document Frame	5
FUR-FU-10001473	Eldon Executive Woodline II Desk Accessories, Mahogany	5
FUR-FU-10001473	Eldon Executive Woodline II Desk Accessories, Mahogany	9
FUR-FU-10001473	Eldon Executive Woodline II Desk Accessories, Mahogany	9
FUR-FU-10001473	DAX Wood Document Frame	5
FUR-FU-10001473	DAX Wood Document Frame	9
FUR-FU-10001473	Eldon Executive Woodline II Desk Accessories, Mahogany	5
FUR-FU-10001473	Eldon Executive Woodline II Desk Accessories, Mahogany	5
FUR-FU-10001473	DAX Wood Document Frame	9

As the diagram above shows, there are many products that have different product names in accordance with one product no. All those product names are updated with unique values as the appear in the latest order.

2.2 Principle of Data Consolidation

2.2.1 Policy: All data sources must be extracted, transformed, and stored in the developed ERD structure.

2.2.1.1 Procedure

All data is extracted from sheets and transformed into compatible structures with of the structure in schema before they can be loaded into target data store. For example, 'sales' and 'profit' columns in 'orders' sheet contain more than 2 decimals, and the target structure will only hold 2 decimals, these values need be transformed into 2 decimals format.

2.2.1.2 Procedure

'returned' column in 'returns' sheet is transformed and mapped into integer values, which 1 means 'completed' and '2' means returned.

2.2.2 Policy: Missing values and inconsistencies must be taken care of.

2.2.2.1 Procedure

Missing values for 'postal code' are filled in with random values selected from the same city.

2.2.2.2 Procedure

product_no	order_no	product_name	quantity	discount
OFF-BI-10002160	CA-2020-129714	Acco Hanging Data Binders	1	0.20
OFF-BI-10004995	CA-2020-129714	GBC DocuBind P400 Electric Binding System	4	0.20
OFF-PA-10001970	CA-2020-129714	Xerox 1908	2	0.00
OFF-PA-10001970	CA-2020-129714	Xerox 1908	4	0.00
TEC-AC-10000290	CA-2020-129714	Sabrent 4-Port USB 2.0 Hub	1	0.00

In the above diagram, within the same order, there is one product with 2 different quantities, and this will result in inconsistencies in database. These records will be combined into one record with the quantity and profit being summed respectively.

3. Sub-domain: Data Storage and Operations

3.1 Principle of Ensuring the integrity of data assets

3.1.1 Policy: Implement data validation procedure to check data accuracy and completeness.

3.1.1.1 Procedure

Unique keys and secondary candidate keys are implemented in entities according to their definitions to prevent from inserting duplicated values.

3.1.1.2 Procedure

Compare number of records between data sources and targets, and make sure all records have been successfully loaded into tables.

3.1.2 Policy: All data must follow transaction processing.

3.1.2.1 procedure

Foreign Key Checks is turned on, which is default from MySQL database settings. This makes sure that tables with foreign keys constraints follow transaction processing.

3.1.2.2 Procedure

Foreign Key constrains on individual entities are placed when necessary to make sure fields in attribute will be updated or deleted in alignment.

3.2 Principle of Build with reuse in mind

3.2.1 Policy: Develop and store frequently used queries as reusable database views or stored procedures

3.2.1.1 Procedure

Executive report query is written into stored procedure to simplify the querying process for all applications.

```
DELIMITER $$
CREATE PROCEDURE get_executive_report(start_at, end_at)
BEGIN
SELECT
        EXTRACT(year FROM orders.order_date) AS year,
        regions.name AS region,
        orders.order_no AS orders_order_no,
        product_orders.order_price AS product_orders_order_price,
        product_orders.quantity AS product_orders_quantity,
        product_orders.order_discount AS product_orders_order_discount,
        product_orders.order_profit AS product_orders_order_profit
FROM regions
JOIN states
        ON regions.id = states.region_id
JOIN cities
        ON cities.state_id = states.id
JOIN addresses
        ON addresses.city_id = cities.id
JOIN address_customers
        ON address_customers.address_id = addresses.id
JOIN customers
        ON customers.id = address_customers.customer_id
JOIN orders
        ON orders.customer_id = customers.id
JOIN product_orders
        ON product_orders.order_id = orders.id
WHERE orders.order_date BETWEEN start_at AND end_at
GROUP BY
        orders.order_date,
        region,
        orders.order_no,
        product_orders.order_price,
        product_orders.quantity,
```

```
        product_orders.order_discount,
        product_orders.order_profit
ORDER BY
        year,
        region,
        orders.order_no
END$$
DELIMITER ;
```

3.2.1.2 Procedure

Operational report query is written into stored procedure to simplify the querying process for all applications.

```
DELIMITER $$
CREATE PROCEDURE get_operational_report(start_at, end_at)
BEGIN
SELECT
        date_format(orders.order_date, '%Y-%m') AS year_and_month,
        regions.name AS region, states.name AS state, cities.name AS city, cp.name AS category,
        count(orders.id) AS sum_orders,
        sum(case orders.status_id when 2 then 1 else 0 end) AS sum_returned,
        sum(product_orders.order_price * product_orders.quantity * (1 -
product_orders.order_discount)) AS order_sales,
        sum(product_orders.order_profit) AS order_profits,
        sum(product_orders.order_price * product_orders.quantity * (1 -
product_orders.order_discount) - product_orders.order_profit) AS order_cogs
FROM product_orders
INNER JOIN orders ON product_orders.order_id = orders.id
INNER JOIN products ON product_orders.product_id = products.id
INNER JOIN categories ON products.category_id = categories.id
INNER JOIN categories cp ON categories.parent_id = cp.id
INNER JOIN customers ON customers.id = orders.customer_id
INNER JOIN addresses ON orders.address_id = addresses.id
INNER JOIN cities ON addresses.city_id = cities.id
INNER JOIN states ON cities.state_id = states.id
INNER JOIN regions ON regions.id = states.region_id
WHERE orders.order_date BETWEEN start_at AND end_at
GROUP BY
        date_format(orders.order_date, '%Y-%m'),
        regions.name, states.name, cities.name, cp.name
ORDER BY year_and_month, region
END$$
DELIMITER;
```

3.2.2 Policy: Identify and develop common data models, and entities that can be reused across other applications.

3.2.2.1 Procedure

Addresses are developed into common data models, along with cities, states and counties that can be used with other applications as well. Especially cities, states, and countries because they are reference data.

3.2.2.2 Procedure

Category is also developed into common data model. It is a self-reference entity, consequently, it is possible to have multiple levels of hierarchy of categories.

4. Operational Report

Operational Sales Report: Analysis of Regional in the U.S. in January 2021

Period	Region	State	City	Category	Total Orders	Total Returns	Total Sales	Sales KPI	Total Profits	Total COGS
2021-01	Central	Illinois	Aurora	Furniture	1	0	69.375	69	-47.18	116.555
				Office Supplies	3	0	309.6	103	96.56	213.04
				Technology	1	0	2003.168	2003	250.4	1752.768
			Chicago	Office Supplies	4	0	46.64	12	-10.64	57.28
		Indiana	Richmond	Furniture	1	0	18.96	19	8.53	10.43
				Office Supplies	4	0	100.02	25	28.43	71.59
				Technology	3	0	239.72	80	74.22	165.5
		Iowa	Des Moines	Furniture	1	0	34.58	35	14.52	20.06
				Office Supplies	4	0	98.64	25	39.67	58.97
				Technology	1	0	207	207	51.75	155.25
			Marion	Furniture	1	0	14.91	15	4.62	10.29
				Office Supplies	3	0	121.29	40	55.25	66.04
		Kansas	Wichita	Office Supplies	1	0	279.9	L	137.15	142.75
		Michigan	Detroit	Furniture	1	0	210.98	211	21.1	189.88
				Technology	1	0	3059.982	3060	680	2379.982
			Jackson	Furniture	1	0	302.67	303	72.64	230.03
				Office Supplies	4	0	5603.95	1401	2578.58	3025.37
				Technology	3	0	1135.913	379	313.18	822.733
		Missouri	Springfield	Furniture	1	0	212.94	213	53.24	159.7
				Office Supplies	3	0	4406.39	1469	153.41	4252.98
		Texas	Austin	Office Supplies	2	0	37.06	19	-59.08	96.14
				Technology	2	0	110.576	55	27.7	82.876
			Dallas	Office Supplies	1	0	760.98	761	-1141.47	1902.45
			El Paso	Furniture	1	0	913.43	913	-169.64	1083.07
				Office Supplies	3	0	409.32	136	18.3	391.02
			Houston	Office Supplies	1	0	18.16	18	1.82	16.34
			Huntsville	Furniture	2	0	452.164	226	-214.02	666.184
				Office Supplies	5	0	502.766	101	-176.94	679.706
			Keller	Office Supplies	1	0	6	6	2.1	3.9
		Wisconsin	Franklin	Office Supplies	1	0	3.6	4	1.73	1.87
	East	Connecticut	Waterbury	Office Supplies	1	0	3.52	4	1.02	2.5
		District of Colum	Washington	Furniture	1	0	37.68	38	15.83	21.85
				Office Supplies	1	0	40.08	40	19.24	20.84
		Massachusetts	Quincy	Office Supplies	1	0	12.7	13	5.84	6.86
		New York	New York City	Furniture	1	0	207.846	208	2.31	205.536
				Office Supplies	1	0	5.22	5	2.4	2.82
				Technology	2	0	587.85	294	193.33	394.52
		Ohio	Kent	Office Supplies	1	0	14.016	14	1.75	12.266
				Technology	2	0	179.958	90	-36	215.958
			Lorain	Furniture	1	0	48.896	49	8.56	40.336
		Pennsylvania	Philadelphia	Furniture	2	0	919.239	460	-56.99	976.229
				Office Supplies	5	0	2259.49	452	95.34	2164.15
				Technology	1	0	429.6	430	-93.08	522.68
		Vermont	Burlington	Office Supplies	4	0	637.18	159	196.31	440.87
	South	Alabama	Hoover	Office Supplies	2	0	22.63	11	7.62	15.01
			Tuscaloosa	Office Supplies	1	0	33.74	34	15.52	18.22
		Florida	Miami	Furniture	1	0	419.136	419	-68.11	487.246
			Ormond Beach	Office Supplies	1	0	2.808	3	-1.97	4.778
		Georgia	Columbus	Furniture	1	0	62.72	63	24.46	38.26
				Technology	1	0	2939.93	2940	764.38	2175.55
			Smyrna	Office Supplies	1	0	5.67	6	0.11	5.56
		North Carolina	Charlotte	Office Supplies	2	0	383.992	192	1.84	382.152
			Jacksonville	Office Supplies	3	0	66.258	22	-26.35	92.608
				Technology	2	0	703.62	352	-27.14	730.76
		Tennessee	Johnson City	Office Supplies	4	4	63.102	16	12.93	50.172
				Technology	1	1	111.984	112	7	104.984
	West	Arizona	Tucson	Office Supplies	1	0	4.938	5	-3.62	8.558
				Technology	1	0	95.984	96	12	83.984
		California	Costa Mesa	Furniture	1	1	37.74	38	12.83	24.91
				Technology	1	1	239.97	240	26.4	213.57
			Long Beach	Office Supplies	4	0	332.8	83	80.69	252.11
			Los Angeles	Furniture	3	0	902.022	301	192.37	709.652
				Office Supplies	2	0	176.3	88	75.92	100.38
				Technology	2	0	177.366	89	15.86	161.506
			Rancho Cucamonga	Office Supplies	1	0	38.88	39	18.66	20.22
			San Francisco	Furniture	1	1	120.784	121	-13.59	134.374
				Office Supplies	11	2	2364.598	215	763.52	1601.078
				Technology	1	0	359.976	360	130.49	229.486
			San Jose	Office Supplies	2	0	302.644	151	96.67	205.974
				Technology	1	0	110.352	110	8.28	102.072
		Colorado	Aurora	Office Supplies	1	0	168.624	169	14.75	153.874
				Technology	1	0	169.064	169	-14.79	183.854
		Montana	Great Falls	Office Supplies	3	0	1189.43	396	75.57	1113.86
				Technology	1	0	2999.95	3000	1379.98	1619.97
		Washington	Seattle	Furniture	2	0	977.96	489	99.07	878.89
				Office Supplies	7	0	441.372	63	81.79	359.582
				Technology	3	0	871.09	290	155.51	715.58

This research delves into an in-depth analysis of sales across four pivotal regions in the United States: Central, East, South, and West. By examining the primary challenges in each operational city and evaluating the sales and return rates for 2021, our findings indicate a significant issue in the South region, particularly in Johnson City, Tennessee. The product category "office supply" has reported 4 returns, mirroring the exact count of defective orders. Upon dissecting the profit-loss metrics, challenges emerge across all regions:

1. Central: Illinois

2. East: Ohio and Pennsylvania

3. South: Florida and North Carolina

4. West: Arizona, California, and Colorado.

The most prevalent product category associated with these losses is **Office Supply**. Addressing these issues has become critical for increasing our profitability.

KPI Performance

We set a 10% growth goal for each year, which means sales of current year need be greater than the sales of previous year times 1.1.

KPI performance: Sales(current year) >= Sales(previous year) * (1 + 10%)

Sales = Unit Price of Product * quantity * (1 – discount)

COGS = Sales – ProfitsSales KPI = sales / orders

Our operational analytics suggest a return rate of roughly 6.45% 6.45% for the given month, implying 6 to 7 returns for every 100 purchases. This stresses the need of scrutinizing sectors such as the South region, particularly the Office Supply category. Our profit margin is roughly 16.24 % 16.24%, which means that for every $1 earned in sales, we net approximately $0.162 in profit. The average transaction value, as represented by the Sales KPI, is roughly $326.26, serving as a baseline for our average order value across all sectors and geographies.

Potential Future Directions

Logistics Redesign: Our logistics might use a new coat of paint. Consider route optimization, cost-cutting, and on-time delivery. Modern logistical tools could be the solution.

Improving Our Quality: The frequent "Office Supply" bug screams poor quality. It's past time we evaluated our quality control procedures, scrutinized our suppliers, and increased product inspections.

Warehouse Work: Time is of the essence, and our warehouses cannot afford to be late. We're talking about high-quality warehouse systems, process automation, and regular employee upskilling.

Solutions that can be implemented include improving our logistical operations, ensuring on-time delivery, increasing order correctness, and improving the efficiency of our warehouses and shipping protocols.

Assumptions made in this report:

1. 10% revenue increase is the goal.

2. The products are divided into three categories: furniture, office supplies, and technology.

5. Executive report

Executive Report: Analysis in the U.S. 2021 vs 2020

Region	Sales 2020 ($)	Sales 2021 ($)	Sales 2021 vs 2020 (%)	KPI Performance	Profit 2021 ($)	Profit 2021 vs 2020 (%)	COGS 2020 (%)	COGS 2021 (%)
Central	147426.9134	147100.5877	-0.22%	Below Target	7550.78	-62.05%	86.50%	94.87%
East	180673.1328	213083.3194	17.94%	Above Target	33230.46	64.99%	88.85%	84.40%
South	93618.3163	122905.197	31.28%	Above Target	8848.89	-50.01%	81.09%	92.80%
West	187479.5583	250118.9997	33.41%	Above Target	43809.04	82.15%	87.17%	82.48%

This research provides a comparative examination of sales in four key regions of the United States: Central, East, South, and West. To determine the performance trend, we compared sales, profit, and COGS from 2021 to statistics from 2020.

1. **Central Region**

While sales fell moderately by 0.22% from 2020, profit fell by a more significant 62.05%. The cost of goods sold (COGS) grew from 86.5% in 2020 to 94.87% in 2021. This region's performance is categorized as **"Below Target."**

2. **East Region**

Sales increased by 17.94% in 2021 over 2020. Profit increased significantly by 64.99%. COGS declined from 88.85% in 2020 to 84.4% in 2021, suggesting improved cost management and trend maintenance. This region's performance is rated **"Above Target."**

3. **South Region**

Sales in the South region increased by 31.28% from 2020. Profit, on the other hand, fell by 50.01%. The cost of goods sold increased from 81.09% in 2020 to 92.8% in 2021. Despite the drop in profits, the region's performance is rated **"Above Target."**

4. **West Region**

The West region saw the greatest increase in sales, with a 33.41% gain. Profits increased by 82.15% as well. COGS has decreased from 87.17% in 2020 to 82.48% in 2021. The performance of this region is also classified as **"Above Target."**

In Summary, while sales have increased across the board, profitability in the Central region is a source of worry. Except for the Central and South, most regions were able to maintain or cut their COGS. Strategic measures are required to assure long-term growth and profitability, particularly in regions that are falling short of their targets.

Assumptions made in this report:

1. 10% revenue increase is the goal.

6. Annex

6.1 Lab Exercise 1

Lab Exercises 1

When analyzing data sheets, Pandas Framework is used in the process of creating this document. All orders, personnel, and returns sheets are scrutinized for missing values, duplicated data, completeness, and discrepancies. The code used to complete this task is included in Appendix I.

1. **Analysis:** data analysis of **Sample Superstore** spreadsheet:

Data Overview:

1) Orders

- Number of Entries: 9994
- Number of Columns: 21

Columns Information:

- 1. Row ID
- 2. Order ID
- 3. Order Date
- 4. Ship Date
- 5. Ship Mode
- 6. Customer ID
- 7. Customer Name
- 8. Segment
- 9. Country/Region
- 10. City
- 11. State
- 12. Postal Code
- 13. Region
- 14. Product ID
- 15. Category
- 16. Sub-Category
- 17. Product Name
- 18. Sales
- 19. Quantity
- 20. Discount
- 21. Profit

Data Quality Analysis:

- Duplicate Rows: 0
- Missing Values:
  - Postal Code: 11 missing entries, if no further data will be provided to fill in these missing values, these missing values will be filled in according to its 'State' randomly.

Analysis Summary:

- The dataset is quite clean with no duplicate entries.

- There are some missing values in the 'Postal Code' column that might require attention depending on the use case.
- **Inconsistencies:** 'Sales' and 'Quantity' are unclear because it does not directly state whether sales value includes all quantities, hence further assumptions are made based on this.
- **Redundancies:** This dataset has no redundancies.
- **Detailed Analysis:** Dive deeper into the columns like 'Sales', 'Quantity', 'Discount', and 'Profit' to understand the data distribution and there are no possible errors or outliers.

2) People

- Number of Entries: 4
- Number of Columns: 2

Columns Information:

- 1. Regional Manager 2. Region

Data Quality Analysis:

- Duplicate Rows: 0
- Missing Values: None

Analysis Summary:

- The second spreadsheet contains information about Regional Managers and the regions in which they work.
- There are no missing or duplicate values, indicating good data quality for this small dataset.
- When we combine data from this sheet with data from the Orders sheet, we can use only one column to reduce redundant information.

3) Returns

- Number of Entries: 800
- Number of Columns: 2

Columns Information:

- 1. Returned 2. Order ID

Data Quality Analysis:

- Duplicate Rows: 504
- Missing Values: None

Analysis Summary:

- The third spreadsheet contains information about orders, including whether or not they were returned.
- There are **no missing values**, but there are a large number of **duplicate** rows (504), which could be deliberate (if several products per order can be returned) or could necessitate further research and cleaning.

Summary of All Datasets:

- **Sample Superstore Data:** A comprehensive dataset with details about orders, customers, and financials.
- **People Sheet:** A mapping between regional managers and their respective regions.
- **Returns Sheet:** Information about orders that were returned, though it contains many duplicate rows that might need further examination.
- The "**Order ID**" appears to be a common link between "Orders" and "Returns," which could provide information about returns. Similarly, the "**Region**" data in "Orders" might be related to the regional manager data in 'People'.

2. Target Audience

- **Operational Reports:**
  Target Audience: Operations Team, Regional Managers, Customer Service Team
  Intended Use:
  - **Monitor and Control:** Track sales, returns, and customer interactions to identify issues and opportunities in real-time.
  - **Performance Improvement:** Identify areas/products where returns are high, or sales are low and need improvements.
  - **Achievable:** Minimize returns, optimize stock levels, improve customer service, and enhance operational efficiency.
  - **Reduce cost:** Minimize costs by checking the location and planning to send to save money.
  - **Return Problems:** analyze the return product issue and resolve it.
  - **Delivery Days:** Analysts monitor inventory costs from day to day.

- **Discount:** Analysts discount the reasons for price reductions and the impact on profit loss.

- **Executive Reports**

  Target Audience: Executives, Strategic Planners, Marketing Team
  Intended Use:
  - **Decision Making:** Utilize data to strategize marketing efforts, manage resources, and plan future initiatives.
  - **Research Analysis:** Analyze trends, customer behavior, and sales performance for informed business strategy development.
  - **Achievable:** Make informed strategic decisions, identify market trends, optimize marketing efforts, and enhance overall business strategy.
  - **Pareto Principle:** Focus on the top 20% of customers using the Pareto Principle (80/20) in both profit and loss terms to improve profit and reduce loss.

3. **Context and Additional Assumptions**.
   - The data across all sheets is accurate and up to date.
   - The trends and patterns in the historical data are representative and can be utilized for future planning.
   - The "Sample Superstore" data represents the entirety of the sales, not a subset.
   - 'People' regional managers are in charge of all sales and returns in their particular territories.
   - By combining 'Returns' and 'Orders' data, analysts can follow product returns and identify problems.
   - Make a bar graph to compare each product and region.
   - The term 'Sales' in the sheet refers to the overall sales of the order.
   - One 'Sub-Category' will not belong to multiple 'Category's.

4. **Operational and Executive Reports**:
   - **Operational Reports**

    - **Information Displayed:** Sales, Returns, Customer Details, Regional Data, Distribution of Delivery Days
  - **KPIs:**
    - **Sales Performance**: $Sales\ KPIs = \frac{Total\ Sales}{Total\ Orders}$
    - **Sales Target Attainment:** $= \frac{Sales\ for\ the\ current\ period}{Sales\ target\ (Assume)} \times 100\%$
    - **Return Rate:** $Return\ Rate = \frac{Total\ Returns}{Total\ Orders} \times 100\%$

- **Executive Reports**
  - **Information Displayed**: Sales Summary, Return Overview, Financials

- (Profit, Loss), Customer segment ratio with Delivery days.
- **KPIs:**
  - **Cost of Goods Sold (COGS):** $= Sales - Gross\ Profit$
  - **Profit Margin:** $= \frac{Profit}{Sales} \times 100\%$
  - **Sales Growth:** $= \frac{Current\ peroid\ sales - Sales\ during\ past\ period}{Sales\ during\ period}$
  - **Customer Lifetime Value:** Assumed to be calculated using purchase history and retention rates (requires further data and analysis).
  - **Retention Rate:**

$$= \left[\frac{(Number\ of\ customers\ at\ the\ end\ of\ time\ period - New\ customer\ added)}{Number\ of\ customers\ at\ begining\ of\ the\ time\ period}\right]$$

5. Design empty templates

Information Display														
Region	State	City	Item	Sales Overview			Returns		Regional Performance		Discount	Discount	Gross	
				Total Sales ($)	Units Sold (Items)	Average Sales Per Order ($)	Total Returns (Items)	Return Rate (%)	Sales per region (%)	Returns per region (%)	Applied (%)	Cap (%)	Sale ($)	

KPIs		
Sales Performance	Sales Target Attainment	Return Rate

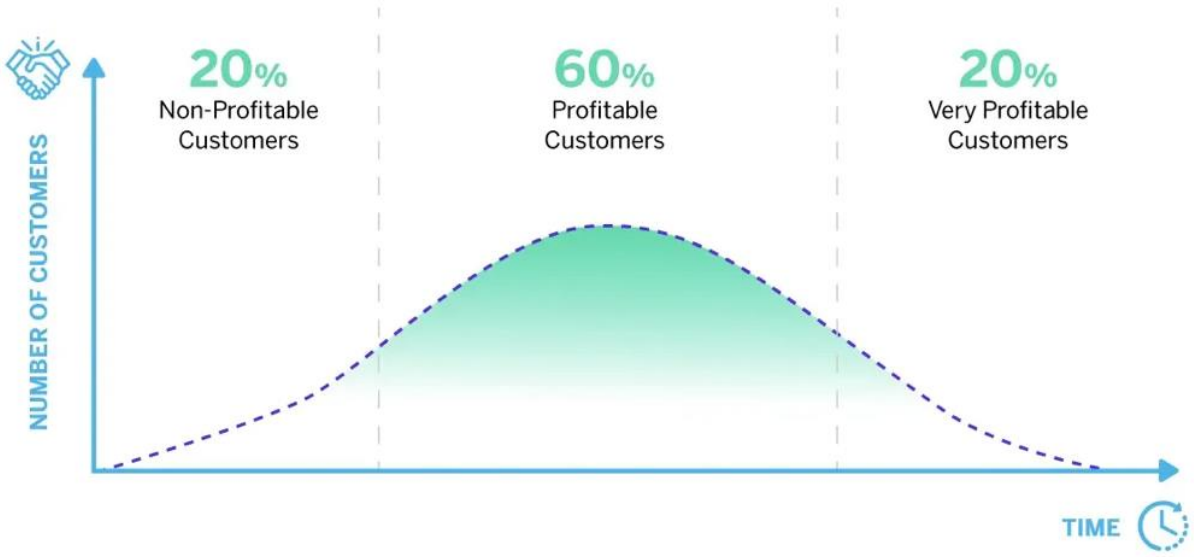
Executive Report Template

Province	Information Display								
	Financial Overview			Sales Summary			Market Trends		Strategic Insights
	Total Profit	Profit Margin	Losses due to Returns	Total Sales	Sales per Segment	Sales per region	Popular product	Emerging customer preferences	Data-driven recommendation and observations

Province	Top Product		Top Customer	
	Profit	Loss	Profit	Loss

qualtrics.XM

Customer Lifetime Value is the net profit contribution of the customer to the firm over time



KPIs											
Province	2021	2020	2019	2018	Sales Growth		Profit Margin		COGS		Retention Rate
					AVG	Growth (2021)	AVG	Growth (2021)	AVG	Growth (2021)	
	($)	($)	($)	($)	(%)	(%)	(%)	(%)	(%)	(%)	(Numbers)

Appendix I

```python
# dama_lab_exercise_1.py
import pandas as pd

# show the missing values
def missing_values(data: pd.DataFrame):
    for col in data:
        missing_data = data[col].isna().sum()
        if missing_data > 0:
            perc = missing_data / len(data) * 100
            print(f'Feature {col} >> Missing entries: {missing_data} \
                | Percentage: {round(perc, 2)} \
                | Data Type: {data[col].dtypes}')

# load data from sheets and store them in vars
df_orders = pd.read_excel('../Sample - Superstore.xls', sheet_name='Orders')
df_people = pd.read_excel('../Sample - Superstore.xls', sheet_name='People')
df_returns = pd.read_excel('../Sample - Superstore.xls', sheet_name='Returns')

# check basic information of orders
print(df_orders.info())

# check missing values of different data sheets
missing_values(df_orders)
missing_values(df_people)
missing_values(df_returns)

# check any duplicated rows existing in data sheet
print(df_orders[df_orders.duplicated()])
# returns sheet contains 504 duplicated rows
print(df_returns[df_returns.duplicated()])

# check inconsistency values that may exist
print(df_orders[df_orders.Sales < 0])
print(df_orders[df_orders.Quantity < 0]]
print(df_orders[df_orders.Discount < 0])
```

```
print(df_orders[df_orders['Order Date'] > df_orders['Ship Date']])
```

6.2 Lab Exercise 2

Lab Exercises 2

Introduction

In the rapidly evolving world of data, understanding the intricacies of data management and database design is paramount. This lab exercise is a testament to that exploration. The purpose of this exercise is to delve deep into the world of data analysis, design, and management using the "Sample Superstore" dataset as our primary data source. Through this endeavor, we aim to:

Understand and visually represent the flow of data within a system using a Data Flow Diagram.

Design and implement a database schema that accurately captures the relationships, entities, and attributes of our dataset, as depicted in the Entity Relationship Diagram.

Utilize programming and database tools, specifically Python and SQLalchemy, to automate data extraction, cleaning, and loading processes, ensuring data integrity and consistency.

Highlight the methodologies and codes employed in this intricate process, providing a clear roadmap for similar future projects.

Our journey begins with loading the raw data from the "Sample Superstore" dataset into a dedicated table, ‘superstore_orders'. This table serves as our base, holding the raw data in its original form. From there, using a combination of Python and SQL, we dissect, refine, and channel this data into specific entities, ensuring it's primed for analysis and reporting. The subsequent sections detail our approach, methodologies, and the results of this exercise, with visual representations and code snippets provided in the appendices.

Methodology

The methodology employed in this lab exercise was meticulously crafted to ensure a systematic and efficient approach to handling, transforming, and analyzing the data from the "Sample Superstore" dataset. The primary stages of our methodology are detailed below.

1. **Data Exploration**

Before any transformations or cleaning, it's crucial to understand the data's structure, attributes, and potential inconsistencies. Tools like Python, with its extensive data analysis libraries, provided a quick and comprehensive overview of the dataset, highlighting areas that required attention.

2. **Data Visualization with Lucidchart**

Visual representations often simplify complex data structures, making them more comprehensible. Lucidchart was chosen due to its intuitive interface and the ease with which it can create both Data Flow Diagrams and Entity Relationship Diagrams. These visual aids were instrumental in mapping out the data's flow and relationships, setting a clear path for subsequent stages.

3. **Data Cleaning & Transformation with Python and SQL**

Data rarely comes in a perfect format. Using Python, we were able to automate several cleaning tasks such as filling in missing values, removing duplicates, and splitting attributes (like names). SQL, on the other hand, provided a robust platform for data extraction, transformation, and loading (ETL). Its inherent capabilities in handling relational data made it an ideal choice for tasks like aggregations, joins, and filtering.

4. **Database Design with SQLalchemy**

The design of a database is paramount to its efficiency and ease of use. SQLalchemy, a versatile toolkit for SQL in Python, was used to streamline the creation of entities in the database, ensuring a smooth transition from raw data to a structured database format. Its Object Relational Mapping (ORM) capabilities allowed for a more intuitive interaction with the database using Python.

5. **Iterative Refinement**

Given the complex nature of data transformation and design, our approach was inherently iterative. After each major step, a series of checks and validations were performed to ensure data integrity and consistency. If inconsistencies were detected, the data was looped back for refinement, ensuring the highest quality in the final output.

Data analysis:

1. Metadata table:
    a. metadata

2. Reference data tables:
    1. countries
    2. states
    3. cities
    4. categories
    5. segments
    6. order statuses
    7. regions

3. Master data tables:
    1. customers
    2. addresses
    3. employees
    4. address customers
    5. products

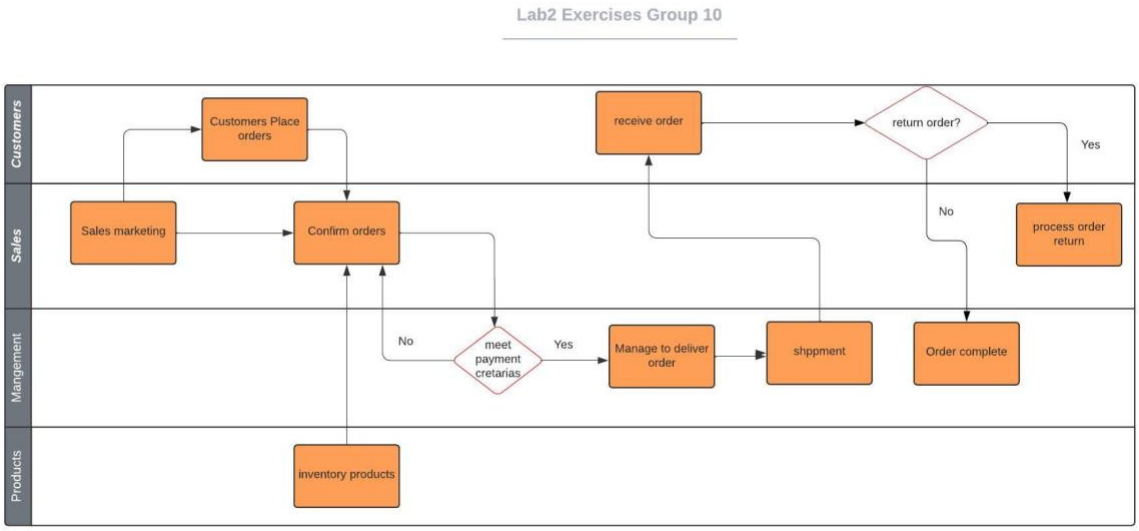
4. Transactional data tables:
    1. orders
    2. product orders
    3. shipments

5. Reporting data tables:
    1. Operational reports
    2. Executives' reports

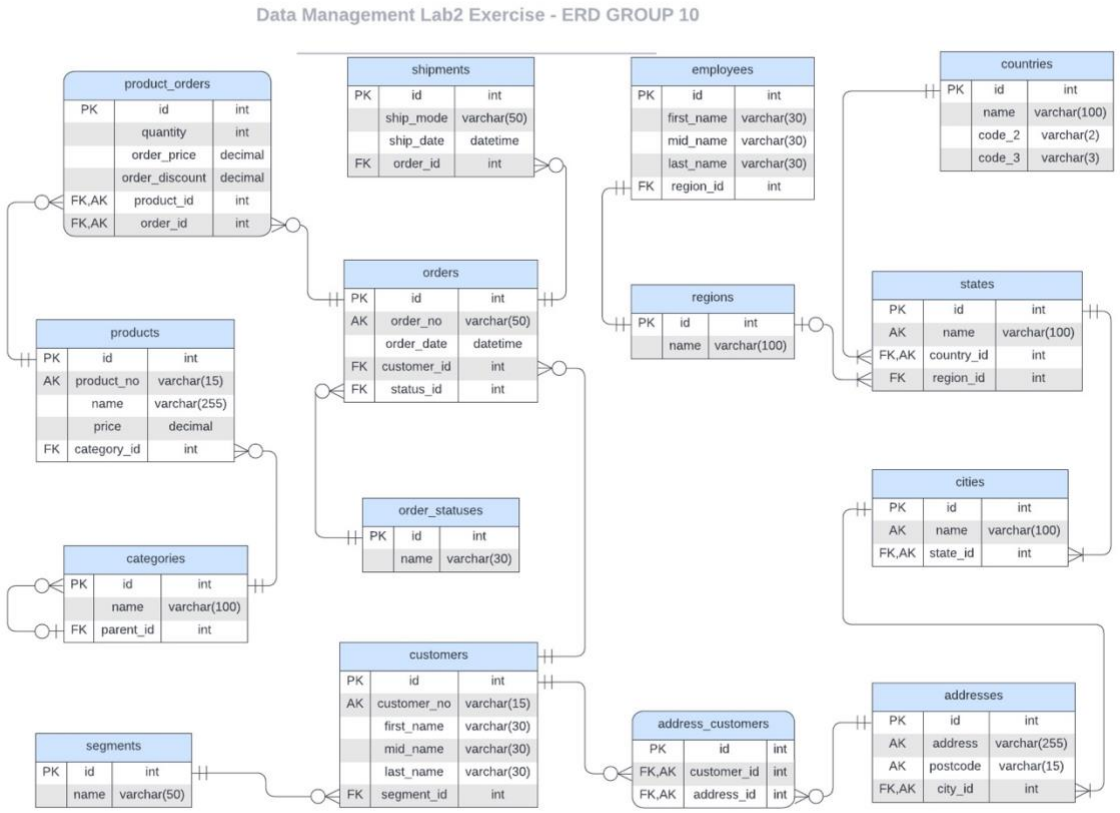
Data Modification:

1. Missing 'postal codes' are filled in with random values picked from the postal codes belonging to that city.
2. Some products have different names with the same product number, in this case, different names are updated with a unique name.
3. Order statuses are filled in with 'returned' and 'completed'. To accomplish this, the 'return_status_id' attribute is added to the superstore_orders table, values are filled if orders are returned, and kept 'null' if not.
4. The 'address' attribute is added to the address's entity, and no data will be filled in.
5. The 'price' attribute is added to the product entity, it means the unit price for a product.
6. Names are divided into 'first_name', 'mid_name', and 'last_name' attributes in all name-related attributes. Names will be split based on whitespace and saved in different attributes.
7. 'code_2' and 'code_3' attributes are added to the countries entity, for future possible usage.
8. The same product appears in the same order multiple times with different quantities and sales, in this scenario, these product's quantity will be summed to remove duplicates.

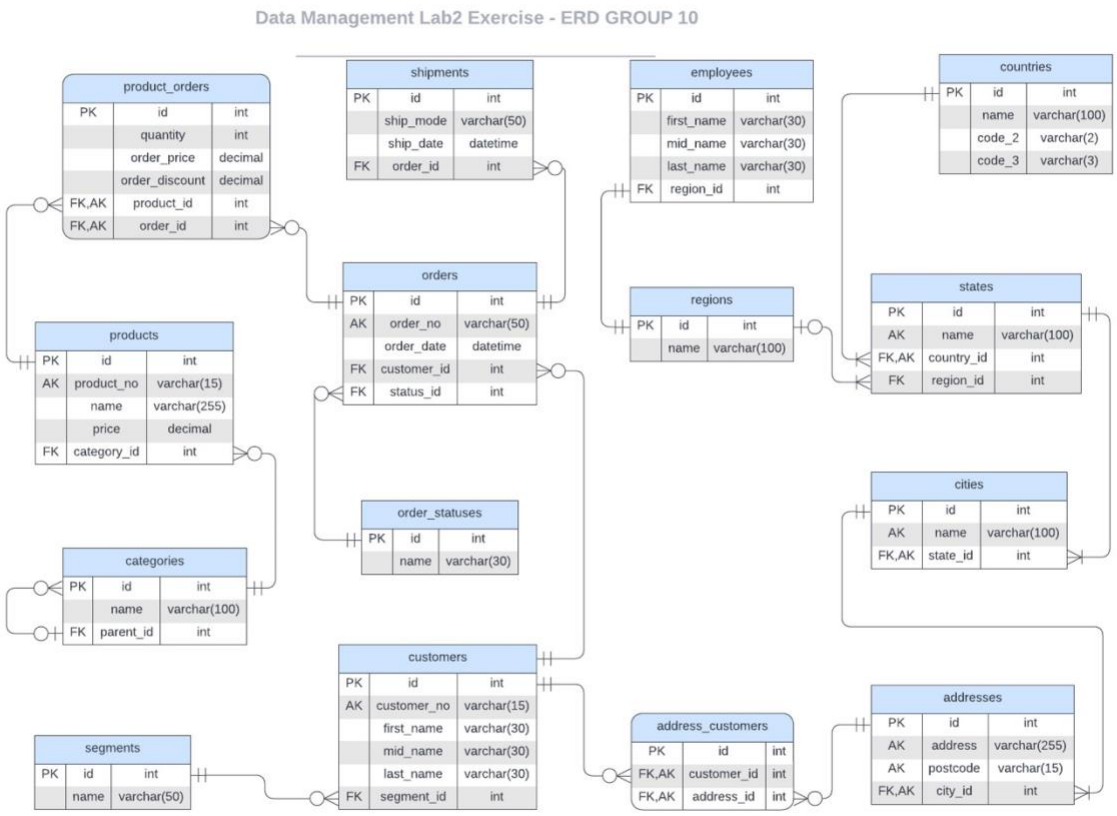
Appendix I – Data Flow Diagram



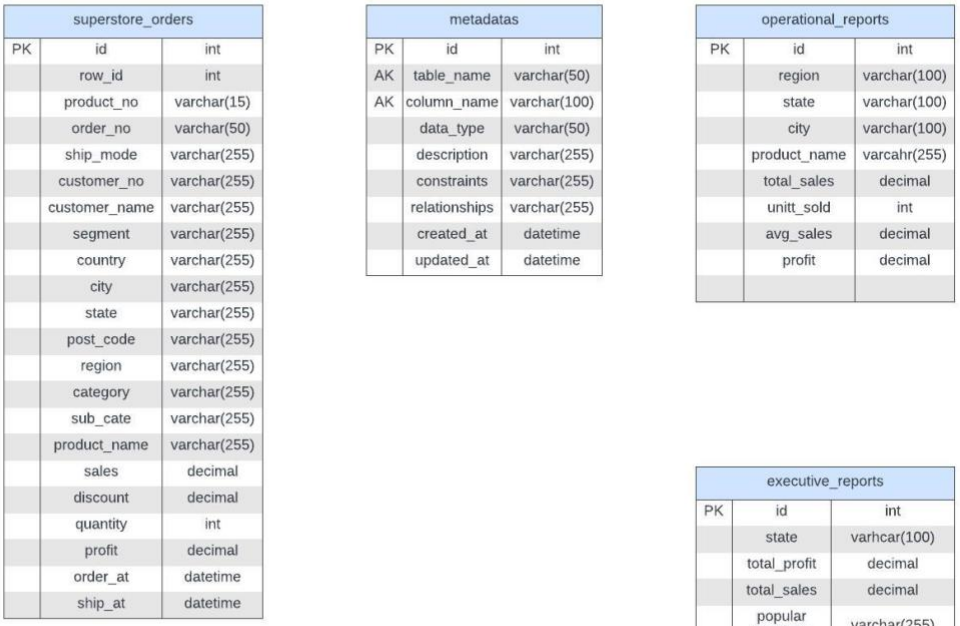
Appendix II – ERD



Data Schema


Data Management Lab2 Exercise - ERD GROUP 10
product_orders
PK id int
quantity int
order_price decimal
order_discount decimal
FK,AK product_id int
FK,AK order_id int
shipments
PK id int
ship_mode varchar(50)
ship_date datetime
FK order_id int
employees
PK id int
first_name varchar(30)
mid_name varchar(30)
last_name varchar(30)
FK region_id int
countries
PK id int
name varchar(100)
code_2 varchar(2)
code_3 varchar(3)
orders
PK id int
AK order_no varchar(50)
order_date datetime
FK customer_id int
FK status_id int
regions
PK id int
name varchar(100)
states
PK id int
AK name varchar(100)
FK,AK country_id int
FK region_id int
products
PK id int
AK product_no varchar(15)
name varchar(255)
price decimal
FK category_id int
order_statuses
PK id int
name varchar(30)
cities
PK id int
AK name varchar(100)
FK,AK state_id int
categories
PK id int
name varchar(100)
FK parent_id int
customers
PK id int
AK customer_no varchar(15)
first_name varchar(30)
mid_name varchar(30)
last_name varchar(30)
FK segment_id int
address_customers
PK id int
FK,AK customer_id int
FK,AK address_id int
addresses
PK id int
AK address varchar(255)
AK postcode varchar(15)
FK,AK city_id int
segments
PK id int
name varchar(50)

Lab2 Exercises Group 10



superstore_orders		
PK	id	int
	row_id	int
	product_no	varchar(15)
	order_no	varchar(50)
	ship_mode	varchar(255)
	customer_no	varchar(255)
	customer_name	varchar(255)
	segment	varchar(255)
	country	varchar(255)
	city	varchar(255)
	state	varchar(255)
	post_code	varchar(255)
	region	varchar(255)
	category	varchar(255)
	sub_cate	varchar(255)
	product_name	varchar(255)
	sales	decimal
	discount	decimal
	quantity	int
	profit	decimal
	order_at	datetime
	ship_at	datetime

metadatas		
PK	id	int
AK	table_name	varchar(50)
AK	column_name	varchar(100)
	data_type	varchar(50)
	description	varchar(255)
	constraints	varchar(255)
	relationships	varchar(255)
	created_at	datetime
	updated_at	datetime

operational_reports		
PK	id	int
	region	varchar(100)
	state	varchar(100)
	city	varchar(100)
	product_name	varcahr(255)
	total_sales	decimal
	unitt_sold	int
	avg_sales	decimal
	profit	decimal

executive_reports		
PK	id	int
	state	varhcar(100)
	total_profit	decimal
	total_sales	decimal
	popular products	varchar(255)
	top_customer	varchar(100)

Appendix III - Codes

Create models for entities:

```
from datetime import datetime
from decimal import Decimal
from typing import List
from typing import Optional

from sqlalchemy import ForeignKey
from sqlalchemy import func
from sqlalchemy import Numeric
# from sqlalchemy import Integer
from sqlalchemy.dialects.mysql import INTEGER
from sqlalchemy import String
from sqlalchemy import Index
```

```python
from sqlalchemy.orm import DeclarativeBase
from sqlalchemy.orm import Mapped
from sqlalchemy.orm import mapped_column
from sqlalchemy.orm import registry
from sqlalchemy.orm import relationship

mapper_registry = registry()

class Base(DeclarativeBase):
    pass

# metadata table
class Metadata(Base):
    __tablename__ = 'metadatas'

    id = mapped_column(INTEGER(unsigned=True),
                       primary_key=True,
                       autoincrement=True)
    table_name: Mapped[str] = mapped_column(String(50),
                                            nullable=False,
                                            comment='table name')
    column_name: Mapped[Optional[str]] = mapped_column(String(100),
                                                comment='column name')
    data_type: Mapped[Optional[str]] = mapped_column(String(50),
                                               comment='data type for the column, eg: int')
    description: Mapped[Optional[str]] = mapped_column(String(255),
                                                comment='description')
    constraints: Mapped[Optional[str]] = mapped_column(String(255),
                                                comment='constrains, eg: foreign key, index, unique')
    relationships: Mapped[Optional[str]] = mapped_column(String(255),
                                                  comment='relationships, eg: has many... belongs to...')

    created_at: Mapped[datetime] = mapped_column(insert_default=func.now())
    updated_at: Mapped[datetime] = mapped_column(insert_default=func.now())

    __table_args__ = (
        Index('table_column_name_index', 'table_name', 'column_name', unique=True),
```

```
    )

    def __repr__(self):
        return f'<Metadata {self.id} ({self.table_name} - {self.column_name})>'

# reporting data
class ExecutiveReport(Base):
    __tablename__ = 'executive_reports'

    id = mapped_column(INTEGER(unsigned=True), primary_key=True, autoincrement=True)
    region: Mapped[str] = mapped_column(String(100), nullable=True,
                                        comment='region name, such as "Ontario"')
    state: Mapped[str] = mapped_column(String(100), nullable=True,
                                       comment='state name, such as "Ontario"')
    city: Mapped[str] = mapped_column(String(100), nullable=True,
                                      comment='city name, such as "Ontario"')
    product_name: Mapped[str] = mapped_column(String(255), nullable=True,
                                      comment='city name, such as "Ontario"')
    total_sales: Mapped[int] = mapped_column(Numeric(12, 2), nullable=True,
                                       comment='product discount in order, such as "0.23"')
    unit_sold: Mapped[Optional[int]]
    avg_sales: Mapped[int] = mapped_column(Numeric(12, 2), nullable=True,
                                       comment='product discount in order, such as "0.23"')
    profit: Mapped[int] = mapped_column(Numeric(12, 2), nullable=True,
                                       comment='product discount in order, such as "0.23"')


# reporting data
class OperationalReport(Base):
    __tablename__ = 'operational_reports'

    id = mapped_column(INTEGER(unsigned=True), primary_key=True, autoincrement=True)
    state: Mapped[str] = mapped_column(String(100), nullable=True,
                                       comment='state name, such as "Ontario"')
    total_profit: Mapped[int] = mapped_column(Numeric(12, 2), nullable=True,
                                       comment='product discount in order, such as "0.23"')
    total_sales: Mapped[int] = mapped_column(Numeric(12, 2), nullable=True,
```

```
                                comment='product discount in order, such as "0.23"')

# reference data
class Segment(Base):
    __tablename__ = 'segments'

    id = mapped_column(INTEGER(unsigned=True), primary_key=True, autoincrement=True)
    name: Mapped[str] = mapped_column(String(50), nullable=False,
                                      comment='segment name, such as "Home Office"')

    # one segment can be assignment to multiple customers
    customers:Mapped[List['Customer']] = relationship(back_populates='segment')

    def __repr__(self):
        return f'<Metadata {self.id} - {self.name})>'

# master data
class Customer(Base):
    __tablename__ = 'customers'

    id = mapped_column(INTEGER(unsigned=True), primary_key=True, autoincrement=True)
    customer_no: Mapped[str] = mapped_column(String(15), unique=True,
                                         comment='generated customer number, such as "ABC-SDF-121123"')
    first_name: Mapped[str] = mapped_column(String(30), nullable=False,
                                        comment='first name')
    mid_name: Mapped[Optional[str]] = mapped_column(String(30),
                                            comment='middle name')
    last_name: Mapped[str] = mapped_column(String(30), nullable=False,
                                       comment='last name')

    segment_id: Mapped[INTEGER(unsigned=True)] = mapped_column(ForeignKey('segments.id',
                                                           ondelete='NO ACTION',
                                                           onupdate='CASCADE'),
                                                  nullable=False,
                                                  comment='fk: references to segments table')
    # one customer has one segment
    segment:Mapped['Segment'] = relationship(back_populates='customers')
```

```
    # one customer have many orders
    orders:Mapped[List['Order']] = relationship(back_populates='customer')
    # customer have mupltiple addresses
    address_customers:Mapped[List['AddressCustomer']] = relationship(back_populates='customer')

    def __repr__(self):
        return f'<Metadata {self.id} - {self.customer_no} references to segment: {self.segment_id})>'

# reference data
class OrderStatus(Base):
    __tablename__ = 'order_statuses'

    id = mapped_column(INTEGER(unsigned=True), primary_key=True, autoincrement=True)
    name: Mapped[str] = mapped_column(String(30), nullable=False,
                                      comment='order status label, such as "returned"')

    # one order status label can be assigned to multiple orders
    orders:Mapped[List['Order']] = relationship(back_populates='order_status')

    def __repr__(self):
        return f'<Metadata {self.id} - {self.name})>'

# transactional data
class Order(Base):
    __tablename__ = 'orders'

    id = mapped_column(INTEGER(unsigned=True), primary_key=True, autoincrement=True)
    order_no: Mapped[str] = mapped_column(String(50), unique=True, comment='generated order number, such as
"ABC-SDF-121123"')
    order_date: Mapped[datetime] = mapped_column(insert_default=func.now())
    customer_id: Mapped[INTEGER(unsigned=True)] = mapped_column(ForeignKey('customers.id', ondelete='NO
ACTION', onupdate='CASCADE'),
                                                nullable=False,
                                                comment='fk: references to customers table')

    status_id: Mapped[INTEGER(unsigned=True)] = mapped_column(ForeignKey('order_statuses.id', ondelete='NO
ACTION', onupdate='CASCADE'),
```

```
                                            nullable=False,
                                            comment='fk: references to order_statuses table')
    # one order belongs to one customer
    customer:Mapped['Customer'] = relationship(back_populates='orders')
    # one order has one order status label
    order_status:Mapped['OrderStatus'] = relationship(back_populates='orders')
    # one order has one shipment info
    shipment:Mapped['Shipment'] = relationship(back_populates='order')
    # one order has many product orders
    product_orders:Mapped[List['ProductOrder']] = relationship(back_populates='order')

    def __repr__(self):
        return f'<Metadata {self.id} - {self.order_no}, {self.customer_id})>'

# transactional data
class Shipment(Base):
    __tablename__ = 'shipments'

    id = mapped_column(INTEGER(unsigned=True), primary_key=True, autoincrement=True)
    ship_mode: Mapped[str] = mapped_column(String(50), nullable=False,
                                           comment='ship mode, such as ECONOMIC                                                    ')
    ship_date: Mapped[datetime] = mapped_column(insert_default=func.now())
    order_id: Mapped[INTEGER(unsigned=True)] = mapped_column(ForeignKey('orders.id',
                                                    ondelete='NO ACTION',
                                                    onupdate='CASCADE'),
                                            nullable=False,
                                            comment='fk: references to orders table')
    # one shippment info belongs to one order
    order:Mapped['Order'] = relationship(back_populates='shipment')
    # order_status:Mapped['OrderStatus'] = relationship(back_populates='orders')

    def __repr__(self):
        return f'<Metadata {self.id} - {self.ship_date} {self.order_id})>'

# master data
class Employee(Base):
    __tablename__ = 'employees'
```

```
    id = mapped_column(INTEGER(unsigned=True), primary_key=True, autoincrement=True)
    first_name: Mapped[str] = mapped_column(String(30), nullable=False,
                                            comment='first name')
    mid_name: Mapped[Optional[str]] = mapped_column(String(30),
                                                    comment='middle name')
    last_name: Mapped[str] = mapped_column(String(30), nullable=False,
                                           comment='last name')

    region_id: Mapped[INTEGER(unsigned=True)] = mapped_column(ForeignKey('regions.id',
                                                              ondelete='NO ACTION',
                                                              onupdate='CASCADE'),
                                                      nullable=False,
                                                      comment='fk: references to regions table')
    # one employee has one region
    region:Mapped['Region'] = relationship(back_populates='employee')

    def __repr__(self):
        return f'<Metadata {self.id} - {self.first_name} references to segment: {self.region_id})>'

# reference data
class Region(Base):
    __tablename__ = 'regions'

    id = mapped_column(INTEGER(unsigned=True), primary_key=True, autoincrement=True)
    name: Mapped[str] = mapped_column(String(100), nullable=False,
                                      comment='region name, such as "West"')

    # one region belongs to one employee
    employee:Mapped['Employee'] = relationship(back_populates='region')
    # one region has many states
    states:Mapped[List['State']] = relationship(back_populates='region')

    def __repr__(self):
        return f'<Metadata {self.id} - {self.name})>'

# reference data
```

```python
class Country(Base):
    __tablename__ = 'countries'

    id = mapped_column(INTEGER(unsigned=True), primary_key=True, autoincrement=True)
    name: Mapped[str] = mapped_column(String(100), nullable=False,
                                      comment='country name, such as "Canada"')
    code_2: Mapped[Optional[str]] = mapped_column(String(2),
                                            comment='country 2-alpha code, such as "CA"')
    code_3: Mapped[Optional[str]] = mapped_column(String(3),
                                            comment='country 3-alpha code, such as "CAN"')

    # one country has many states
    states:Mapped[List['State']] = relationship(back_populates='country')



    def __repr__(self):
        return f'<Metadata {self.id} - {self.name})>'

# reference data
class State(Base):
    __tablename__ = 'states'

    id = mapped_column(INTEGER(unsigned=True), primary_key=True, autoincrement=True)
    name: Mapped[str] = mapped_column(String(100), nullable=False,
                                      comment='state name, such as "Ontario"')

    country_id: Mapped[INTEGER(unsigned=True)] = mapped_column(ForeignKey('countries.id',
                                                    ondelete='NO ACTION',
                                                    onupdate='CASCADE'),
                                               nullable=False,
                                               comment='fk: references to countries table')
    region_id: Mapped[Optional[INTEGER(unsigned=True)]] = mapped_column(ForeignKey('regions.id',
                                                            ondelete='NO ACTION',
                                                            onupdate='CASCADE'),
                                               comment='fk: references to regions table')
    __table_args__ = (
        Index('countryid_name_idx', 'country_id', 'name', unique=True),
```

```
    )
    # one state belongs to one region
    region:Mapped['Region'] = relationship(back_populates='states')
    # one state belongs to one country
    country:Mapped['Country'] = relationship(back_populates='states')
    # one state has many cities
    cities:Mapped[List['City']] = relationship(back_populates='state')

    def __repr__(self):
        return f'<Metadata {self.id} - {self.name})>'

# reference data
class City(Base):
    __tablename__ = 'cities'

    id = mapped_column(INTEGER(unsigned=True), primary_key=True, autoincrement=True)
    name: Mapped[str] = mapped_column(String(100), nullable=False,
                                      comment='city name, such as "Toronto"')

    state_id: Mapped[INTEGER(unsigned=True)] = mapped_column(ForeignKey('states.id',
                                                        ondelete='NO ACTION',
                                                        onupdate='CASCADE'),
                                                    nullable=False,
                                                    comment='fk: references to states table')
    __table_args__ = (
        Index('stateid_name_idx', 'state_id', 'name', unique=True),
    )
    # one city belongs to one state
    state:Mapped['State'] = relationship(back_populates='cities')
    # one city has many addresses
    addresses:Mapped[List['Address']] = relationship(back_populates='city')

    def __repr__(self):
        return f'<Metadata {self.id} - {self.name})>'

# master data
class Address(Base):
```

```
    __tablename__ = 'addresses'

    id = mapped_column(INTEGER(unsigned=True), primary_key=True, autoincrement=True)
    address: Mapped[Optional[str]] = mapped_column(String(255),
                                          comment='address name, such as "52 fairglen ave"')
    postcode: Mapped[str] = mapped_column(String(15), nullable=False,
                                    comment='post code, such as "123123"')

    city_id: Mapped[INTEGER(unsigned=True)] = mapped_column(ForeignKey('cities.id',
                                                    ondelete='NO ACTION',
                                                    onupdate='CASCADE'),
                                                nullable=False,
                                                comment='fk: references to cities table')
    __table_args__ = (
        Index('cityid_address_postcode_idx', 'city_id', 'address', 'postcode', unique=True),
    )
    # one address belongs one city
    city:Mapped['City'] = relationship(back_populates='addresses')
    # one address has many customer addresses
    address_customers:Mapped[List['AddressCustomer']] = relationship(back_populates='address')

    def __repr__(self):
        return f'<Metadata {self.id} - {self.post_code})>'

# master data
class AddressCustomer(Base):
    __tablename__ = 'address_customers'

    id = mapped_column(INTEGER(unsigned=True), primary_key=True, autoincrement=True)
    customer_id: Mapped[INTEGER(unsigned=True)] = mapped_column(ForeignKey('customers.id',
                                                     ondelete='NO ACTION',
                                                     onupdate='CASCADE'),
                                                nullable=False,
                                                comment='fk: references to customers table')
    address_id: Mapped[INTEGER(unsigned=True)] = mapped_column(ForeignKey('addresses.id',
                                                    ondelete='NO ACTION',
                                                    onupdate='CASCADE'),
```

```
                                        nullable=False,
                                        comment='fk: references to addresses table')
    __table_args__ = (
        Index('ck_customerid_addressid', 'customer_id', 'address_id', unique=True),
    )

    # one address customer reflects one address info
    address:Mapped['Address'] = relationship(back_populates='address_customers')
    # one address customer reflects one customer info
    customer:Mapped['Customer'] = relationship(back_populates='address_customers')

    def __repr__(self):
        return f'<Metadata {self.id} - {self.customer_id})>'

# reference data
class Category(Base):
    __tablename__ = 'categories'

    id = mapped_column(INTEGER(unsigned=True), primary_key=True, autoincrement=True)
    name: Mapped[str] = mapped_column(String(100),
                                      nullable=False,
                                      comment='category name, such as "Fruit"')

    parent_id: Mapped[Optional[INTEGER(unsigned=True)]] = mapped_column(ForeignKey('categories.id',
                                                                                   ondelete='NO ACTION',
                                                                                   onupdate='CASCADE'),
                                                                        comment='fk: self-references to categories table')

    # one sub-category belongs to one category
    parent:Mapped[Optional['Category']] = relationship(back_populates='children')
    # one category has many sub-category
    children:Mapped[Optional[List['Category']]] = relationship(back_populates='parent',
                                                               remote_side='Category.id')
    # category can be assignment to many products
    products:Mapped[List['Product']] = relationship(back_populates='category')

    def __repr__(self):
```

```
        return f'<Metadata {self.id} - {self.name})>'

# master data
class Product(Base):
    __tablename__ = 'products'

    id = mapped_column(INTEGER(unsigned=True), primary_key=True, autoincrement=True)
    product_no: Mapped[str] = mapped_column(String(15), unique=True,
                                            comment='generated product number, such as "ABC-SDF-121123"')
    name: Mapped[str] = mapped_column(String(255), nullable=False,
                                      comment='product name, such as "Computer"')
    price: Mapped[int] = mapped_column(Numeric(12, 2), nullable=False,
                                       comment='product price, such as "1233.23"')
    # discount: Mapped[int] = mapped_column(Numeric(4, 2), nullable=False,
    #                                       comment='product discount, such as "0.23"')
    category_id: Mapped[INTEGER(unsigned=True)] = mapped_column(ForeignKey('categories.id',
                                                                           ondelete='NO ACTION',
                                                                           onupdate='CASCADE'),
                                                                nullable=False,
                                                                comment='fk: references to categories table')

    # one product belongs to one category
    category:Mapped['Category'] = relationship(back_populates='products')
    # one product belongs to many orders
    # orders:Mapped[List['Order']] = relationship(back_populates='product')
    # one product has many product orders
    product_orders:Mapped[List['ProductOrder']] = relationship(back_populates='product')

    def __repr__(self):
        return f'<Metadata {self.id} - {self.name}, {self.category_id})>'

# transactional data
class ProductOrder(Base):
    __tablename__ = 'product_orders'

    id = mapped_column(INTEGER(unsigned=True), primary_key=True, autoincrement=True)
    quantity: Mapped[int] = mapped_column(INTEGER(unsigned=True))
```

```
    order_price: Mapped[int] = mapped_column(Numeric(12, 2),
                                             nullable=False,
                                             comment='product price in order, such as "1233.23"')
    order_discount: Mapped[int] = mapped_column(Numeric(4, 2),
                                               nullable=False,
                                               comment='product discount in order, such as "0.23"')

    product_id: Mapped[INTEGER(unsigned=True)] = mapped_column(ForeignKey('products.id',
                                                                          ondelete='NO ACTION',
                                                                          onupdate='CASCADE'),
                                                               nullable=False,
                                                               comment='fk: references to products table')
    order_id: Mapped[INTEGER(unsigned=True)] = mapped_column(ForeignKey('orders.id',
                                                                        ondelete='NO ACTION',
                                                                        onupdate='CASCADE'),
                                                             nullable=False,
                                                             comment='fk: references to orders table')

    __table_args__ = (
        Index('ck_productid_orderid', 'product_id', 'order_id', unique=True),
    )

    # one product order reflects many products
    product:Mapped['Product'] = relationship(back_populates='product_orders')
    # one product order reflects many orders
    order:Mapped['Order'] = relationship(back_populates='product_orders')

    def __repr__(self):
        return f'<Metadata {self.id} - {self.order_id}, {self.product_id})>'


# this table contains data from sample-superstore.xsl orders sheet
# this table is used to ETL data into different tables
# technically, it's not a part of the project database
class SupserstoreOrder(Base):
    __tablename__ = 'superstore_orders'
```

```
    id = mapped_column(INTEGER(unsigned=True), primary_key=True, autoincrement=True)
    row_id: Mapped[int] = mapped_column(INTEGER(unsigned=True))
    product_no: Mapped[str] = mapped_column(String(15), nullable=False,
                                            comment='generated product number, such as "ABC-SDF-121123"')
    order_no: Mapped[str] = mapped_column(String(50), nullable=False,
                                          comment='generated order number, such as "ABC-SDF-121123"')
    ship_mode: Mapped[Optional[str]] = mapped_column(String(255))
    customer_no: Mapped[Optional[str]] = mapped_column(String(255))
    customer_name: Mapped[Optional[str]] = mapped_column(String(255))
    segment: Mapped[Optional[str]] = mapped_column(String(255))
    country: Mapped[Optional[str]] = mapped_column(String(255))
    city: Mapped[Optional[str]] = mapped_column(String(255))
    state: Mapped[Optional[str]] = mapped_column(String(255))
    post_code: Mapped[Optional[str]] = mapped_column(String(255))
    region: Mapped[Optional[str]] = mapped_column(String(255))
    category: Mapped[Optional[str]] = mapped_column(String(255))
    sub_cate: Mapped[Optional[str]] = mapped_column(String(255))
    product_name: Mapped[Optional[str]] = mapped_column(String(255))
    sales: Mapped[Optional[int]] = mapped_column(Numeric(12, 2))
    discount: Mapped[Optional[int]] = mapped_column(Numeric(4, 2))
    quantity: Mapped[int] = mapped_column(INTEGER(unsigned=True))
    profit: Mapped[int] = mapped_column(Numeric(12, 2), nullable=False,
                                        comment='product discount in order, such as "0.23"')
    return_status_id: Mapped[Optional[int]]
    order_at: Mapped[Optional[datetime]]
    ship_at: Mapped[Optional[datetime]]

    def __repr__(self):
        return f'<Metadata {self.id} - {self.order_no}, {self.product_name})>'
```

clean some inconsistent data:

```
import sqlalchemy as sa
import pandas as pd
import os
from sqlalchemy.orm import Session
from src.models.mapped_models import SupserstoreOrder
```

```
from src.database import engine as db_engine
from src.constants import ROOT_DIR

data_file = os.path.join(ROOT_DIR, 'data', 'Sample - Superstore.xls')

df_orders = pd.read_excel(data_file, sheet_name='Orders')
df_people = pd.read_excel(data_file, sheet_name='People')
df_returns = pd.read_excel(data_file, sheet_name='Returns')

engine = db_engine.sql_engine()

# update the product name so that it's aligned with product no
# doing this by updating every row of product name with the latest name used
def clean_products_v1():
    stmt_sets = (
        sa.select(SupserstoreOrder.product_no, SupserstoreOrder.product_name, sa.func.min(SupserstoreOrder.id))
        .group_by(SupserstoreOrder.product_no, SupserstoreOrder.product_name)
        .order_by(SupserstoreOrder.product_no)
    )
    with Session(bind=engine) as session:
        for p_no, p_name, _ in session.execute(stmt_sets):
            update_stmt = (
                sa.update(SupserstoreOrder)
                .where(SupserstoreOrder.product_no == p_no)
                .values(
                    {'product_name': p_name}
                )
            )
            session.execute(update_stmt)
            session.commit()

# add return status id to superstore orders table
def fillin_order_status():
    df_status = df_returns.drop_duplicates()

    with Session(bind=engine) as session:
        for i in range(len(df_status)):
```

```
        update_stmt = (
            sa.update(SupserstoreOrder)
            .where(SupserstoreOrder.order_no == df_status.iloc[i, 1])
            .values(
                {'return_status_id': 1 if df_status.iloc[i, 0].capitalize() == 'Yes' else 2}
            )
        )
        session.execute(update_stmt)
        session.commit()
```

load data into entities respectively:

```
import os
import pandas as pd
from sqlalchemy.orm import Session
import sqlalchemy as sa

from src.constants import ROOT_DIR
from src.database import engine as db_engine
from src.models import mapped_models as mm
from src.helpers import parse_name, unit_price

data_file = os.path.join(ROOT_DIR, 'data', 'Sample - Superstore.xls')
engine = db_engine.sql_engine()

df_orders = pd.read_excel(data_file, sheet_name='Orders')
df_people = pd.read_excel(data_file, sheet_name='People')
df_returns = pd.read_excel(data_file, sheet_name='Returns')

# dump orders data into superstore_orders table
def dump_orders_db():
    table_name = 'superstore_orders'
    # pick random post code from city 'Burlington'
    df_orders['Postal Code'] = df_orders['Postal Code'].fillna('52601')
    # print(df_orders.head(2))
    df_orders_insert = df_orders.rename(columns={
        'Row ID': 'row_id', 'Order ID':'order_no', 'Order Date':'order_at', \
```

```
        'Ship Date':'ship_at', 'Ship Mode':'ship_mode', \
        'Customer ID':'customer_no', 'Customer Name':'customer_name',\
        'Segment':'segment', 'Country/Region':'country', 'City':'city', \
        'State':'state', 'Postal Code':'post_code', 'Region':'region', \
        'Product ID':'product_no', 'Category':'category', \
        'Sub-Category':'sub_cate', 'Product Name':'product_name', \
        'Sales':'sales', 'Quantity':'quantity', 'Discount':'discount', \
        'Profit':'profit'
    })
    df_order_insert = df_orders_insert.to_dict(orient='records')
    stmt = sa.text(f'INSERT INTO {table_name} (row_id, order_no, order_at, ship_at,\
            ship_mode, customer_no, customer_name, segment, country, city,\
            state, post_code, region, product_no, category, sub_cate, \
            product_name, sales, quantity, discount, profit) \
        values (:row_id, :order_no, :order_at, :ship_at, :ship_mode, :customer_no,\
            :customer_name, :segment, :country, :city, :state, :post_code, :region,\
            :product_no, :category, :sub_cate, :product_name, :sales, :quantity, :discount, :profit)')
    with Session(bind=engine) as session:
        session.execute(stmt, df_order_insert)
        session.commit()

# dump metadata table with descriptions of tables
def insert_metadatas():
    with Session(bind=engine) as session:
        session.execute(
            sa.insert(mm.Metadata), [
                {'table_name': 'address_customers', 'column_name': '', 'data_type': '',
                    'description': 'this table stores customer_id and address_id',
                    'constraints': 'combination of customer_id and address_id is unqiue key in this table',
                    'relationships': 'references to custoemrs table and addresses table'},
                {'table_name': 'address_customers', 'column_name': 'id', 'data_type': 'unsigned int',
                    'description': 'primary key of the table',
                    'constraints': 'nullable=false, autoincrement',
                    'relationships': ''},
                {'table_name': 'address_customers', 'column_name': 'customer_id', 'data_type': 'unsigned int',
                    'description': 'foreign key of the table',
                    'constraints': 'nullable=false',
```

```
                'relationships': 'references to customers table'},
            {'table_name': 'address_customers', 'column_name': 'address_id', 'data_type': 'unsigned int',
                'description': 'foreign key of the table',
                'constraints': 'nullable=false',
                'relationships': 'references to addresses table'},
        ],
    )
    session.commit()

# Extract Transforn and Load  country into countries table
def etl_country():
    with Session(bind=engine) as session:
        session.execute(
            sa.insert(mm.Country), [
                {'name': df_orders['Country/Region'].unique()[0]}
            ]
        )
        session.commit()

# Extract Transforn and Load  people into employees and regions table
def etl_people():
    df_people['id'] = range(1, 1 + len(df_people))
    with Session(bind=engine) as session:
        session.execute(
            sa.insert(mm.Region), [
                {'id': id, 'name': region}
                    for id, region in zip(df_people['id'].tolist(), df_people['Region'].tolist())
            ]
        )
        session.execute(
            sa.insert(mm.Employee), [
                {'first_name': name.split()[1], 'last_name': name.split()[0], 'region_id': id}
                    for id, name in zip(df_people['id'].tolist(), df_people['Regional Manager'].tolist())
            ]
        )

        session.commit()
```

```
# Extract Transforn and Load state data into states table
def etl_state():
    subq = (
        sa.select(mm.SupserstoreOrder.country, mm.SupserstoreOrder.state, mm.SupserstoreOrder.region)
        .group_by(mm.SupserstoreOrder.country, mm.SupserstoreOrder.state, mm.SupserstoreOrder.region)
        .subquery()
    )

    stmt = sa.select(mm.Country.id, subq.c.state, mm.Region.id.label('region_id')).join_from(
        subq, mm.Country, mm.Country.name == subq.c.country
    ).join(
        mm.Region, subq.c.region == mm.Region.name
    )
    with Session(bind=engine) as session:
        session.execute(
            sa.insert(mm.State), [
                {'name': state, 'country_id': country_id, 'region_id': region_id}
                for country_id, state, region_id in session.execute(stmt)
            ],
        )
        session.commit()

# ETL city data to cities table
def etl_city():
    subq = (sa.select(mm.SupserstoreOrder.state, mm.SupserstoreOrder.city)
        .group_by(mm.SupserstoreOrder.state, mm.SupserstoreOrder.city)
        .subquery()
    )
    stmt = sa.select(mm.State.id.label('state_id'), subq.c.city).join_from(
        subq, mm.State, mm.State.name == subq.c.state
    )

    with Session(bind=engine) as session:
        session.execute(
            sa.insert(mm.City), [
                {'name': city, 'state_id': state_id}
```

```
            for state_id, city in session.execute(stmt)
        ],
    )
    session.commit()

# ETL address data to addresses table
def etl_address():
    subq_state = (sa.select(mm.SupserstoreOrder.state,
                            mm.SupserstoreOrder.city,
                            mm.SupserstoreOrder.post_code)
        .group_by(mm.SupserstoreOrder.state,
                  mm.SupserstoreOrder.city,
                  mm.SupserstoreOrder.post_code)
        .subquery()
    )
    subq_city = (sa.select(mm.State.id.label('state_id'),
                           subq_state.c.post_code,
                           subq_state.c.city).join_from(
        subq_state, mm.State, mm.State.name == subq_state.c.state
    ).subquery())
    stmt = sa.select(mm.City.id.label('city_id'), subq_city.c.post_code).join_from(
        subq_city, mm.City, sa.and_(mm.City.name == subq_city.c.city,
                                    mm.City.state_id == subq_city.c.state_id)
    )

    with Session(bind=engine) as session:
        session.execute(
            sa.insert(mm.Address), [
                {'postcode': post_code, 'city_id': city_id}
                for city_id, post_code in session.execute(stmt)
            ],
        )
        session.commit()

# ETL categories into categories table
def etl_category():
    cate_stmt = sa.select(sa.distinct(mm.SupserstoreOrder.category))
```

```
    subq = (sa.select(mm.SupserstoreOrder.category, mm.SupserstoreOrder.sub_cate)
        .group_by(mm.SupserstoreOrder.category, mm.SupserstoreOrder.sub_cate)
        .subquery()
    )
    sub_cate_stmt = sa.select(mm.Category.id.label('parent_id'), subq.c.sub_cate).join_from(
        subq, mm.Category, mm.Category.name == subq.c.category
    )
    with Session(bind=engine) as session:
        session.execute(
            sa.insert(mm.Category), [
                {'name': name} for name in session.scalars(cate_stmt)
            ],
        )
        session.commit()
        session.execute(
            sa.insert(mm.Category), [
                {'name': sub_cate, 'parent_id': parent_id}
                    for parent_id, sub_cate in session.execute(sub_cate_stmt)
            ],
        )
        session.commit()

    # print(cate_stmt)

# etl order statuses into table
def etl_order_status():
    with Session(bind=engine) as session:
        session.execute(
            sa.insert(mm.OrderStatus), [
                {'name': 'completed'},
                {'name': 'returned'}
            ],
        )
        session.commit()

# etl segment into table
def etl_segment():
```

```
    stmt = sa.select(sa.distinct(mm.SupserstoreOrder.segment))
    with Session(bind=engine) as session:
        session.execute(
            sa.insert(mm.Segment), [
                {'name': name} for name in session.scalars(stmt)
            ],
        )
        session.commit()

# etl customers into customers table
def etl_customer():
    subq = (sa.select(mm.SupserstoreOrder.customer_no,
                      mm.SupserstoreOrder.customer_name,
                      mm.SupserstoreOrder.segment)
        .group_by(mm.SupserstoreOrder.customer_no,
                  mm.SupserstoreOrder.customer_name,
                  mm.SupserstoreOrder.segment)
        .subquery()
    )
    stmt = sa.select(mm.Segment.id.label('segment_id'),
                     subq.c.customer_no,
                     subq.c.customer_name).join_from(
        subq, mm.Segment, mm.Segment.name == subq.c.segment
    )
    # print(stmt)
    with Session(bind=engine) as session:
        session.execute(
            sa.insert(mm.Customer), [
                {'customer_no': customer_no, 'segment_id': segment_id,
                'first_name': parse_name(customer_name)[0],
                'mid_name': parse_name(customer_name)[1],
                'last_name': parse_name(customer_name)[2]}
                for segment_id, customer_no, customer_name in session.execute(stmt)
            ],
        )
        session.commit()
```

```
# etl customer address table
def etl_address_customer():
    with Session(bind=engine) as session:
        q = (session.query(sa.distinct(mm.SupserstoreOrder.customer_no),
                           mm.Customer.id, mm.Address.id)
             .join(mm.Customer, mm.SupserstoreOrder.customer_no == mm.Customer.customer_no)
             .join(mm.Address, mm.SupserstoreOrder.post_code == mm.Address.postcode)
             .all())

        session.execute(
            sa.insert(mm.AddressCustomer), [
                {'customer_id': customer_id, 'address_id': address_id}
                    for _, customer_id, address_id in q
            ]
        )
        session.commit()

# etl products table
def etl_product():
    subq_id = (sa.select(sa.func.min(mm.SupserstoreOrder.id).label('min_id'))
        .group_by(mm.SupserstoreOrder.product_no, mm.SupserstoreOrder.product_name)
        # .subquery()
    )
    subq_products = (sa.select(mm.SupserstoreOrder.product_no,
                               mm.SupserstoreOrder.product_name,
                               mm.SupserstoreOrder.sub_cate,
                               mm.SupserstoreOrder.sales,
                               mm.SupserstoreOrder.quantity,
                               mm.SupserstoreOrder.discount)
                     .where(mm.SupserstoreOrder.id.in_(subq_id))
                     .subquery()
    )
    stmt = sa.select(mm.Category.id.label('category_id'),
                     subq_products.c.product_no,
                     subq_products.c.product_name,
                     subq_products.c.sales,
                     subq_products.c.quantity,
```

```
                subq_products.c.discount).join_from(
        subq_products, mm.Category, mm.Category.name == subq_products.c.sub_cate
    )
    # print(stmt)
    with Session(bind=engine) as session:
        session.execute(
            sa.insert(mm.Product), [
                {'product_no': product_no,
                 'category_id': category_id,
                'name': product_name,
                'price': unit_price(sales, quantity, discount)}
                for category_id, product_no, product_name, sales, quantity, discount
                    in session.execute(stmt)
            ],
        )
        session.commit()

# etl orders table
def etl_orders():
    subq = (sa.select(mm.SupserstoreOrder.customer_no, mm.SupserstoreOrder.order_no,
                      mm.SupserstoreOrder.order_at, mm.SupserstoreOrder.return_status_id)
                     .group_by(mm.SupserstoreOrder.customer_no, mm.SupserstoreOrder.order_no,
                     mm.SupserstoreOrder.order_at, mm.SupserstoreOrder.return_status_id)
                    .subquery()
    )
    stmt = sa.select(mm.Customer.id.label('customer_id'), subq.c.order_no,
                     subq.c.order_at, subq.c.return_status_id).join_from(
        subq, mm.Customer, subq.c.customer_no == mm.Customer.customer_no
    )
    with Session(bind=engine) as session:
        session.execute(
            sa.insert(mm.Order), [
                {'order_no': order_no, 'customer_id': customer_id,
                'status_id': 1 if status_id == 1 else 2,
                'order_date': order_date}
                for customer_id, order_no, order_date, status_id in session.execute(stmt)
            ],
```

```
        )
        session.commit()

# ETL product_order table
def etl_product_order():
    subq = (sa.select(mm.SupserstoreOrder.order_no, mm.SupserstoreOrder.product_no,
                      sa.func.sum(mm.SupserstoreOrder.sales).label('sum_sales'),
                      sa.func.sum(mm.SupserstoreOrder.quantity).label('sum_quantity'))
                    .group_by(mm.SupserstoreOrder.order_no, mm.SupserstoreOrder.product_no)
                    .subquery()
    )
    stmt = sa.select(subq.c.sum_sales, subq.c.sum_quantity,
                     mm.Product.price, mm.Product.id.label('product_id'),
                     mm.Order.id.label('order_id')).join(
        mm.Product, mm.Product.product_no == subq.c.product_no
    ).join(
        mm.Order, mm.Order.order_no == subq.c.order_no
    )
    # print(stmt)
    with Session(bind=engine) as session:
        session.execute(
            sa.insert(mm.ProductOrder),[
                {'quantity': sum_quantity, 'order_price': price,
                 'order_discount': round(1-sum_sales/(sum_quantity*price), 2),
                 'order_id': order_id,
                 'product_id': product_id}
                for sum_sales, sum_quantity, price, product_id, order_id
                    in session.execute(stmt)
            ]

        )
        session.commit()

# etl shipment data
def etl_shipment():
    subq = (sa.select(mm.SupserstoreOrder.order_no, mm.SupserstoreOrder.ship_mode,
                      mm.SupserstoreOrder.ship_at)
```

```
            .subquery()
)
stmt = sa.select(subq.c.ship_mode, subq.c.ship_at, mm.Order.id.label('order_id')).join_from(
    subq, mm.Order, subq.c.order_no == mm.Order.order_no
)
with Session(bind=engine) as session:
    session.execute(
        sa.insert(mm.Shipment), [
            {'order_id':order_id,
             'ship_mode': ship_mode,
             'ship_date': ship_at}
            for ship_mode, ship_at, order_id in session.execute(stmt)
        ]
    )
    session.commit()
```

6.3 Lab Exercise 3

Operational Sales Report: Analysis of Regional in the U.S. in January 2021

Period	Region	State	City	Category	Total Orders	Total Returns	Total Sales	Sales KPI	Total Profits	Total COGS
2021-01	Central	Illinois	Aurora	Furniture	1	0	69.375	69	-47.18	116.555
				Office Supplies	3	0	309.6	103	96.56	213.04
				Technology	1	0	2003.168	2003	250.4	1752.768
			Chicago	Office Supplies	4	0	46.64	12	-10.64	57.28
		Indiana	Richmond	Furniture	1	0	18.96	19	8.53	10.43
				Office Supplies	4	0	100.02	25	28.43	71.59
				Technology	3	0	239.72	80	74.22	165.5
		Iowa	Des Moines	Furniture	1	0	34.58	35	14.52	20.06
				Office Supplies	4	0	98.64	25	39.67	58.97
				Technology	1	0	207	207	51.75	155.25
			Marion	Furniture	1	0	14.91	15	4.62	10.29
				Office Supplies	3	0	121.29	40	55.25	66.04
		Kansas	Wichita	Office Supplies	1	0	279.9	L	137.15	142.75
		Michigan	Detroit	Furniture	1	0	210.98	211	21.1	189.88
				Technology	1	0	3059.982	3060	680	2379.982
			Jackson	Furniture	1	0	302.67	303	72.64	230.03
				Office Supplies	4	0	5603.95	1401	2578.58	3025.37
				Technology	3	0	1135.913	379	313.18	822.733
		Missouri	Springfield	Furniture	1	0	212.94	213	53.24	159.7
				Office Supplies	3	0	4406.39	1469	153.41	4252.98
		Texas	Austin	Office Supplies	2	0	37.06	19	-59.08	96.14
				Technology	2	0	110.576	55	27.7	82.876
			Dallas	Office Supplies	1	0	760.98	761	-1141.47	1902.45
			El Paso	Furniture	1	0	913.43	913	-169.64	1083.07
				Office Supplies	3	0	409.32	136	18.3	391.02
			Houston	Office Supplies	1	0	18.16	18	1.82	16.34
			Huntsville	Furniture	2	0	452.164	226	-214.02	666.184
				Office Supplies	5	0	502.766	101	-176.94	679.706
			Keller	Office Supplies	1	0	6	6	2.1	3.9
		Wisconsin	Franklin	Office Supplies	1	0	3.6	4	1.73	1.87
	East	Connecticut	Waterbury	Office Supplies	1	0	3.52	4	1.02	2.5
		District of Colum	Washington	Furniture	1	0	37.68	38	15.83	21.85
				Office Supplies	1	0	40.08	40	19.24	20.84
		Massachusetts	Quincy	Office Supplies	1	0	12.7	13	5.84	6.86
		New York	New York City	Furniture	1	0	207.846	208	2.31	205.536
				Office Supplies	1	0	5.22	5	2.4	2.82
				Technology	2	0	587.85	294	193.33	394.52
		Ohio	Kent	Office Supplies	1	0	14.016	14	1.75	12.266
				Technology	2	0	179.958	90	-36	215.958
			Lorain	Furniture	1	0	48.896	49	8.56	40.336
		Pennsylvania	Philadelphia	Furniture	2	0	919.239	460	-56.99	976.229
				Office Supplies	5	0	2259.49	452	95.34	2164.15
				Technology	1	0	429.6	430	-93.08	522.68
		Vermont	Burlington	Office Supplies	4	0	637.18	159	196.31	440.87
	South	Alabama	Hoover	Office Supplies	2	0	22.63	11	7.62	15.01
			Tuscaloosa	Office Supplies	1	0	33.74	34	15.52	18.22
		Florida	Miami	Furniture	1	0	419.136	419	-68.11	487.246
			Ormond Beach	Office Supplies	1	0	2.808	3	-1.97	4.778
		Georgia	Columbus	Furniture	1	0	62.72	63	24.46	38.26
				Technology	1	0	2939.93	2940	764.38	2175.55
			Smyrna	Office Supplies	1	0	5.67	6	0.11	5.56
		North Carolina	Charlotte	Office Supplies	2	0	383.992	192	1.84	382.152
			Jacksonville	Office Supplies	3	0	66.258	22	-26.35	92.608
				Technology	2	0	703.62	352	-27.14	730.76
		Tennessee	Johnson City	Office Supplies	4	4	63.102	16	12.93	50.172
				Technology	1	1	111.984	112	7	104.984
	West	Arizona	Tucson	Office Supplies	1	0	4.938	5	-3.62	8.558
				Technology	1	0	95.984	96	12	83.984
		California	Costa Mesa	Furniture	1	1	37.74	38	12.83	24.91
				Technology	1	1	239.97	240	26.4	213.57
			Long Beach	Office Supplies	4	0	332.8	83	80.69	252.11
			Los Angeles	Furniture	3	0	902.022	301	192.37	709.652
				Office Supplies	2	0	176.3	88	75.92	100.38
				Technology	2	0	177.366	89	15.86	161.506
			Rancho Cucamonga	Office Supplies	1	0	38.88	39	18.66	20.22
			San Francisco	Furniture	1	1	120.784	121	-13.59	134.374
				Office Supplies	11	2	2364.598	215	763.52	1601.078
				Technology	1	0	359.976	360	130.49	229.486
			San Jose	Office Supplies	2	0	302.644	151	96.67	205.974
				Technology	1	0	110.352	110	8.28	102.072
		Colorado	Aurora	Office Supplies	1	0	168.624	169	14.75	153.874
				Technology	1	0	169.064	169	-14.79	183.854
		Montana	Great Falls	Office Supplies	3	0	1189.43	396	75.57	1113.86
				Technology	1	0	2999.95	3000	1379.98	1619.97
		Washington	Seattle	Furniture	2	0	977.96	489	99.07	878.89
				Office Supplies	7	0	441.372	63	81.79	359.582
				Technology	3	0	871.09	290	155.51	715.58

This research focuses on a detailed analysis of sales throughout four key regions in the United States: Central, East, South, and West. We will investigate the fundamental challenges in each operational city, evaluating sales and return rates for 2021.

Using the findings from our analysis, it is clear that the South region, notably Johnson City, Tennessee, is a major source of concern. The product category "office supply" receives 4 returns, which corresponds exactly to the total number of faulty orders.

Upon dissecting the profit-loss metrics, challenges emerge across all regions:

1. Central: Illinois

2. East: Ohio and Pennsylvania

3. South: Florida and North Carolina

4. West: Arizona, California, and Colorado.

The most prevalent product category associated with these losses is **Office Supply**. Addressing these issues has become critical for increasing our profitability.

KPI Performance

We set a 10% growth goal for each year, which means sales of current year need be greater than the sales of previous year times 1.1.

KPI performance: Sales(current year) >= Sales(previous year) * (1 + 10%)

Sales = Unit Price of Product * quantity * (1 – discount)

COGS = Sales – ProfitsSales KPI = sales / orders

Our operational analytics suggest a return rate of roughly 6.45% 6.45% for the given month, implying 6 to 7 returns for every 100 purchases. This stresses the need of scrutinizing sectors such as the South region, particularly the Office Supply category. Our profit margin is roughly 16.24 % 16.24%, which means that for every $1 earned in sales, we net approximately $0.162 in profit. The average transaction value, as represented by the Sales KPI, is roughly $326.26, serving as a baseline for our average order value across all sectors and geographies.

Potential Future Directions

Logistics Redesign: Our logistics might use a new coat of paint. Consider route optimization, cost-cutting, and on-time delivery. Modern logistical tools could be the solution.

Improving Our Quality: The frequent "Office Supply" bug screams poor quality. It's past time we evaluated our quality control procedures, scrutinized our suppliers, and increased product inspections.

Warehouse Work: Time is of the essence, and our warehouses cannot afford to be late. We're talking about high-quality warehouse systems, process automation, and regular employee upskilling.

Solutions that can be implemented include improving our logistical operations, ensuring on-time delivery, increasing order correctness, and improving the efficiency of our warehouses and shipping protocols.

Executive Report: Analysis in the U.S. 2021 vs 2020

Region	Sales 2020 ($)	Sales 2021 ($)	Sales 2021 vs 2020 (%)	KPI Performance	Profit 2021 ($)	Profit 2021 vs 2020 (%)	COGS 2020 (%)	COGS 2021 (%)
Central	147426.9134	147100.5877	-0.22%	Below Target	7550.78	-62.05%	86.50%	94.87%
East	180673.1328	213083.3194	17.94%	Above Target	33230.46	64.99%	88.85%	84.40%
South	93618.3163	122905.197	31.28%	Above Target	8848.89	-50.01%	81.09%	92.80%
West	187479.5583	250118.9997	33.41%	Above Target	43809.04	82.15%	87.17%	82.48%

This research provides a comparative examination of sales in four key regions of the United States: Central, East, South, and West. To determine the performance trend, we compared sales, profit, and COGS from 2021 to statistics from 2020.

5. Central Region

While sales fell moderately by 0.22% from 2020, profit fell by a more significant 62.05%. The cost of goods sold (COGS) grew from 86.5% in 2020 to 94.87% in 2021. This region's performance is categorized as **"Below Target."**

6. East Region

Sales increased by 17.94% in 2021 over 2020. Profit increased significantly by 64.99%. COGS declined from 88.85% in 2020 to 84.4% in 2021, suggesting improved cost management and trend maintenance. This region's performance is rated **"Above Target."**

7. South Region

Sales in the South region increased by 31.28% from 2020. Profit, on the other hand, fell by 50.01%. The cost of goods sold increased from 81.09% in 2020 to 92.8% in 2021. Despite the drop in profits, the region's performance is rated **"Above Target."**

8. West Region

The West region saw the greatest increase in sales, with a 33.41% gain. Profits increased by 82.15% as well. COGS has decreased from 87.17% in 2020 to 82.48% in 2021. The performance of this region is also classified as **"Above Target."**

In Summary, while sales have increased across the board, profitability in the Central region is a source of worry. Except for the Central and South, most regions were able to maintain or cut their COGS. Strategic measures are required to assure long-term growth and profitability, particularly in regions that are falling short of their targets.