SQL Project

Sales Data Exploration

Overview

In this Sales Data Exploration project, we immerse ourselves in the vast dataset representing the different region's commerce of the sales records of prominent companies like Walmart and Microsoft. With a focus on unraveling the intricate web of relationships between regions, sales representatives, web_event, accounts and orders, we aim to gain deep insights into the dynamics of business interactions within this geographic area. By leveraging SQL queries and data exploration techniques, our goal is to shed light on the patterns and connections that define sales performance and drive success in market landscape.

Problem Statement

Sales analysis is critical for businesses to understand their market performance, identify areas of growth, and optimize strategies for increased profitability. However, navigating through vast amounts of sales data poses challenges in extracting actionable insights efficiently. For businesses such as Walmart and Microsoft, understanding top sellers,

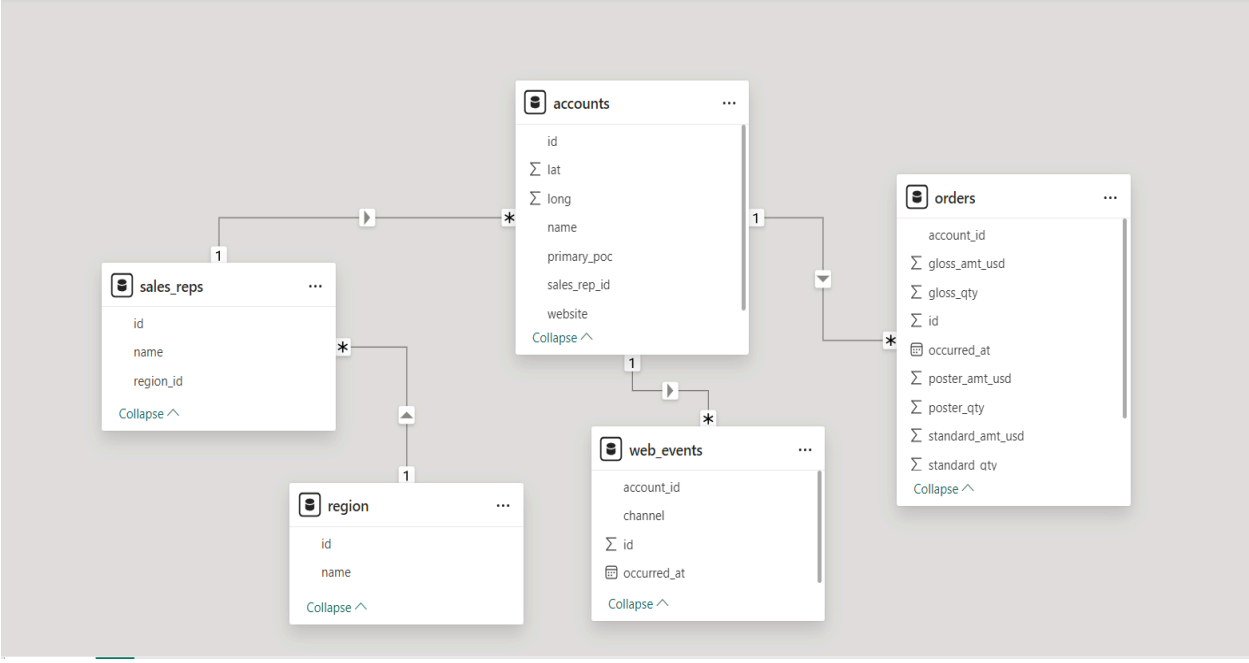
identifying the most valuable customers, and monitoring sales growth rates are vital for strategic planning and operational success. The challenge lies in efficiently extracting and analyzing this data to convert it into actionable insights that can drive business growth.

Objective

The main objective of this sales data exploration is to leverage a MySQL database to discover various insights. Specifically, this project aims to:

1. **Uncover Relationship Dynamics:** Utilize SQL queries to reveal the connections between regions, sales representatives, and accounts, providing clarity on the interactions shaping sales performance.
2. **Identify Key Patterns:** Analyze the data to identify patterns and trends in sales relationships, such as frequent interactions between specific sales representatives and accounts, and understand their implications for business success.
3. **Identify Biggest Customers:** Recognize the most significant customers by sales volume and revenue, highlighting key relationships crucial to the business.
4. **Analyze Sales Growth Rate:** Track and analyze sales growth over time to understand business trends and inform future strategies.
5. **Gain Actionable Insights:** Extract actionable insights from the data to guide businesses in optimizing sales strategies, enhancing customer relationships, and maximizing revenue potential.

Model View



Task 1: Exploring Midwest Business Dynamics.

Retrieve Sales Data by Region for the Midwest.

- The query fetches data from three tables: "region," "sales_reps," and "accounts."
- It selects specific columns from these tables:
- "name" from "region" is renamed as "Region."
- "name" from "sales_reps" is renamed as "Rep_name."
- "name" from "accounts" is renamed as "account_name."
- The tables are connected using JOIN clauses:
- "region" and "sales_reps" are joined on their IDs.
- "sales_reps" and "accounts" are joined on their IDs.
- The query filters the results to include only data from the "Midwest" region.
- The final result is sorted in ascending order based on the "account_name" column.

```
1 •  SELECT region.name AS Region,
2           sales_reps.name AS Rep_name,
3           accounts.name AS account_name
4    FROM region
5    JOIN sales_reps ON region.id = sales_reps.region_id
6    JOIN accounts ON sales_reps.id = accounts.sales_rep_id
7    WHERE region.name = 'Midwest'
8    ORDER BY account_name;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Region	Rep_name	account_name
Midwest	Chau Rowles	Abbott Laboratories
Midwest	Julie Starr	AbbVie
Midwest	Cliff Meints	Aflac
Midwest	Chau Rowles	Alcoa
Midwest	Charles Bidwell	Altria Group
Midwest	Delilah Krum	Amgen
Midwest	Charles Bidwell	Arrow Electronics
Midwest	Delilah Krum	AutoNation
Midwest	Delilah Krum	Capital One Financial
Midwest	Cordell Rieder	Centene
Midwest	Sherlene Wet...	Community Health S...
Midwest	Delilah Krum	Cummins
Midwest	Carletta Kosinski	Danaher

This query delves into the Midwest region's commerce, intertwining region, sales representatives, and account data to unveil a cohesive narrative of business relationships. By filtering for the "Midwest" region and organizing results alphabetically by account name, it provides a clear snapshot of commerce dynamics in this geographical area.

Task 2: Uncovering 'S' Factor in Midwest Sales.

Retrieve Sales Data for Midwest Region Sales Representatives with Names Starting with 'S'.

- The query retrieves data from three tables: "region," "sales_reps," and "accounts."
- It selects specific columns and assigns aliases:
- "name" from the "region" table is aliased as "Region".
- "name" from the "sales_reps" table is aliased as "Rep_name".
- "name" from the "accounts" table is aliased as "account_name."
- The tables are joined using JOIN clauses based on their respective IDs:
- "region" and "sales_reps" are joined on the "region_id" column.
- "sales_reps" and "accounts" are joined on the "sales_rep_id" column.
- The query applies filters in the WHERE clause:
- It selects data only from the "Midwest" region based on the "Region" column.

```
1 •  SELECT region.name AS Region,
2           sales_reps.name AS Rep_name,
3           accounts.name AS account_name
4    FROM region
5    JOIN sales_reps ON region.id = sales_reps.region_id
6    JOIN accounts ON sales_reps.id = accounts.sales_rep_id
7    WHERE region.name = 'Midwest' AND sales_reps.name LIKE 'S%'
8    ORDER BY account_name;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Region	Rep_name	account_name
Midwest	Sherlene Wetherington	Community Health Systems
Midwest	Sherlene Wetherington	Progressive
Midwest	Sherlene Wetherington	Rite Aid
Midwest	Sherlene Wetherington	Time Warner Cable
Midwest	Sherlene Wetherington	U.S. Bancorp

The query identifies Midwest sales representatives whose names start with 'S,' exemplified by Sherlene Wetherington. It showcases their connections with key accounts such as Community Health Systems, Progressive, Rite Aid, Time Warner Cable, and U.S. Bancorp. This insight offers a glimpse into their sales strategies and relationships within the Midwest market.

Task 3: Unveiling 'K' Factor in Midwest Sales.

Retrieve Sales Data for Midwest Region Sales Representatives with 'K' in Their Name.

- The query retrieves data from three tables: "region", "sales_reps", and "accounts".
- It selects specific columns and assigns aliases:
- "name" from the "region" table is aliased as "Region".
- "name" from the "sales_reps" table is aliased as "Rep_name".
- "name" from the "accounts" table is aliased as "account_name."
- The tables are joined using JOIN clauses based on their respective IDs:
- "region" and "sales_reps" are joined on the "region_id" column.
- "sales_reps" and "accounts" are joined on the "sales_rep_id" column.
- The query applies filters in the WHERE clause:
- It selects data only from the "Midwest" region based on the "Region" column.
- It further filters the results to include sales representatives with names containing 'K' with any character before and after 'K' in the "Rep_name" column.
- The final result is sorted in ascending order by the "account_name" column using the ORDER BY clause.

```
1  SELECT region.name AS Region,
2         sales_reps.name AS Rep_name,
3         accounts.name AS account_name
4  FROM region
5  JOIN sales_reps ON region.id = sales_reps.region_id
6  JOIN accounts ON sales_reps.id = accounts.sales_rep_id
7  WHERE region.name = 'Midwest' AND sales_reps.name LIKE '%K%'
8  ORDER BY account_name;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Region	Rep_name	account_name
Midwest	Delilah Krum	Amgen
Midwest	Delilah Krum	AutoNation
Midwest	Delilah Krum	Capital One Financial
Midwest	Delilah Krum	Cummins
Midwest	Carletta Kosinski	Danaher
Midwest	Carletta Kosinski	Dollar General
Midwest	Kathleen Lalonde	EMC
Midwest	Delilah Krum	Hartford Financial Services Group
Midwest	Carletta Kosinski	International Paper
Midwest	Delilah Krum	Kimberly-Clark
Midwest	Kathleen Lalonde	Kraft Heinz
Midwest	Carletta Kosinski	McDonald's
Midwest	Carletta Kosinski	Northrop Grumman

This query spotlights sales representatives with 'K' in their names, revealing the intricate ties between regions, reps, and accounts. it unveils the influence of the 'K' factor. This tale of selective synergy provides insights into the special connections these representatives have with a diverse range of accounts, offering a captivating snapshot of the complex web of business relationships that define the Midwest region.

Task 4: Analyzing Value in High-Volume Sales.

Calculate Unit Price for Orders with Standard Quantity Greater Than 100.

- The query retrieves data from four tables: "region," "sales_reps," "accounts," and "orders."
- It selects specific columns and assigns aliases:
- "name" from the "region" table is aliased as "region."
- "name" from the "accounts" table is aliased as "account_name."
- A calculated column "unit_price" is created as the result of dividing "total_amt_usd" by ("total" + 0.01).
- The tables are joined using JOIN clauses based on their respective IDs:
- "region" and "sales_reps" are joined on the "region_id" column.
- "sales_reps" and "accounts" are joined on the "sales_rep_id" column.
- "accounts" and "orders" are joined on the "account_id" column.
- The query includes a WHERE clause that filters the results:
- It selects data where the "standard_qty" in the "orders" table is greater than 100.

```
1  SELECT region.name AS Region,
2         accounts.name AS account_name,
3         (orders.total_amt_usd / (orders.total + 0.01)) AS unit_price
4  FROM region
5  JOIN sales_reps ON region.id = sales_reps.region_id
6  JOIN accounts ON sales_reps.id = accounts.sales_rep_id
7  JOIN orders ON accounts.id = orders.account_id
8  WHERE orders.standard_qty > 100;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Fetch rows:

Region	account_name	unit_price
Northeast	Walmart	5.759600023667239
Northeast	Walmart	5.965174820318739
Northeast	Walmart	5.44423612294756
Northeast	Walmart	5.960184231258712
Northeast	Walmart	6.1687185711808565
Northeast	Walmart	6.628910225211274
Northeast	Walmart	5.646522151667762
Northeast	Walmart	6.033416662415183
Northeast	Walmart	6.0194924371875205
Northeast	Walmart	6.10980429503357
Northeast	Walmart	6.391137875915393
Northeast	Exxon Mobil	5.054601578529048
Northeast	Apple	5.0965844789521535

The query dissects the connections between regions, accounts, and orders, focusing on those with a standard quantity exceeding 100.By calculating unit prices from total amounts and quantities, it offers insights into the worth of each unit within these sales transactions, underscoring the significance of pricing dynamics in commerce and inspiring businesses to embrace a deeper understanding of their sales units' true worth.

Task 5: Unveiling Pricing Dynamics in Sales.

Calculate Unit Price for Orders with Quantity Conditions and Sort by Unit Price.

- The query retrieves data from four tables: "region," "sales_reps," "accounts," and "orders."
- It selects specific columns and assigns aliases:
- "name" from the "region" table is aliased as "region."
- "name" from the "accounts" table is aliased as "account_name."
- A calculated column "unit_price" is created as the result of dividing "total_amt_usd" by ("total" + 0.01).
- The tables are joined using JOIN clauses based on their respective IDs:
- "region" and "sales_reps" are joined on the "region_id" column.
- "sales_reps" and "accounts" are joined on the "sales_rep_id" column.
- "accounts" and "orders" are joined on the "account_id" column.
- The query includes a WHERE clause that filters the results:
- It selects data where the "standard_qty" in the "orders" table is greater than 100.
- It also checks that the "poster_qty" in the "orders" table is greater than 50.
- The final result is sorted in ascending order by the "unit_price" column using the ORDER BY clause.

```sql
SELECT region.name AS Region,
       accounts.name AS account_name,
       (orders.total_amt_usd / (orders.total + 0.01)) AS unit_price
FROM region
JOIN sales_reps ON region.id = sales_reps.region_id
JOIN accounts ON sales_reps.id = accounts.sales_rep_id
JOIN orders ON accounts.id = orders.account_id
WHERE orders.standard_qty > 100 AND poster_qty > 50
ORDER BY unit_price;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Region	account_name	unit_price
West	Stanley Black & Decker	5.266395560073999
Northeast	Citigroup	5.273081210319704
Southeast	BlackRock	5.2782700457278775
Southeast	Nucor	5.2890939504788514
Northeast	Merck	5.29639144152687

This query intricately connects regions, accounts, and orders, with a specific focus on those with a standard quantity exceeding 100 and poster quantities surpassing 50. The results unveil a vital aspect: the unit price, diligently calculated and thoughtfully ordered in ascending fashion. and helps businesses understand how price and value work together in the changing world of buying and selling.

Task 6: Cracking the Sales Code.

Calculate Unit Price for Orders with Quantity Conditions and Sort by Unit Price (Descending).

- The query retrieves data from four tables: "region", "sales_reps", "accounts", and "orders".
- "name" from the "region" table is aliased as "region".
- "name" from the "accounts" table is aliased as "account_name".
- A calculated column "unit_price" is created as the result of dividing "total_amt_usd" by ("total" + 0.01).
- The tables are joined using JOIN clauses based on their respective IDs:
- "region" and "sales_reps" are joined on the "region_id" column.
- "sales_reps" and "accounts" are joined on the "sales_rep_id" column.
- "accounts" and "orders" are joined on the "account_id" column.
- The query includes a WHERE clause that filters the results:
- It selects data where the "standard_qty" in the "orders" table is greater than 100.
- It also checks that the "poster_qty" in the "orders" table is greater than 50.
- The final result is sorted in descending order by the "unit_price" column using the ORDER BY clause.

```sql
SELECT
    region.name AS Region,
    accounts.name AS account_name,
    (orders.total_amt_usd / (orders.total + 0.01)) AS unit_price
FROM region
JOIN sales_reps ON region.id = sales_reps.region_id
JOIN accounts ON sales_reps.id = accounts.sales_rep_id
JOIN orders ON accounts.id = orders.account_id
WHERE orders.standard_qty > 100 AND poster_qty > 50
ORDER BY unit_price DESC;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Region	account_name	unit_price
Northeast	IBM	8.089906082278144
West	Mosaic	8.066329210358129
West	Pacific Life	8.063022652514793
Northeast	CHS	8.018849326780114
West	Fidelity National Financial	7.9928024668468325
Midwest	Paccar	7.986961758718576
Southeast	PNC Financial Services Group	7.8951386043342335

This query serves as a compass in sales analysis, guiding businesses toward hidden treasures by isolating orders based on specific quantity criteria. By calculating unit prices and sorting them in descending order, it exposes products and accounts with the highest worth. This knowledge empowers businesses to optimize pricing strategies and enhance profitability, shaping the path for smarter decisions in sales.

Task 7: Streamlining Business Performance Analysis.

Calculate Average Order Amounts by Account Name.

- The query retrieves data from two tables: "accounts" and "orders".
- It selects specific columns and calculates average values, assigning aliases to these columns:
- "name" from the "accounts" table is aliased as "account_name."
- Calculates the average of the "standard_amt_usd" column in the "orders" table and is aliased as "avg_standard_amt_usd".
- Calculates the average of the "gloss_amt_usd" column in the "orders" table and is aliased as "avg_gloss_amt_usd".
- Calculates the average of the "poster_amt_usd" column in the "orders" table and is aliased as "avg_poster_amt_usd".
- The tables are joined using a JOIN clause based on the "account_id" column.
- The query includes a GROUP BY clause, which groups the results by the "account_name" column.

```
1 •  SELECT accounts.name AS account_name,
2           AVG(orders.standard_amt_usd) AS avg_standard_amt_usd,
3           AVG(orders.gloss_amt_usd) AS avg_gloss_amt_usd,
4           AVG(orders.poster_amt_usd) AS avg_poster_amt_usd
5     FROM accounts
6     JOIN orders ON accounts.id = orders.account_id
7     GROUP BY account_name;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

account_name	avg_standard_amt_usd	avg_gloss_amt_usd	avg_poster_amt_usd
Walmart	564.4937500000001	339.390625	318.7099999999999
Exxon Mobil	2629.73	104.86	0
Apple	2568.1866666666665	133.5716666666667	92.02666666666669
Berkshire Hathaway	5728.52	0	1745.8
McKesson	1267.4599999999998	209.72	243.6
UnitedHealth Group	2477.535	260.90166666666664	40.6
CVS Health	1378.903333333333	192.24333333333334	156.98666666666665
General Motors	4006.97	232.19	154.28
Ford Motor	1567.3136363636365	243.42500000000007	281.98545454545456
AT&T	758.3017857142858	106.1975	136.88000000000002
General Electric	732.2824999999999	292.11	99.47000000000001
AmerisourceBergen	1621.75	441.90999999999997	93.38
Verizon	1468.9312500000003	232.18999999999997	173.56499999999997
Chevron	1163.9175	255.59625	116.725

This query efficiently summarizes key sales figures for each account, providing decision-makers with a quick overview of business performance. By calculating the average order amounts for standard, gloss, and poster products associated with each account, it offers actionable insights into revenue generation and customer engagement.

Task 8: Comprehensive Account Performance Summary.

Calculate Average Order Amounts by Account Name and Sort by Total Amount (Descending).

- The query retrieves data from two tables: "accounts" and "orders".
- "name" from the "accounts" table is aliased as "account_name".
- Calculates the average of the "standard_amt_usd" column in the "orders" table and is aliased as "avg_standard_amt_usd."
- Calculates the average of the "gloss_amt_usd" column in the "orders" table and is aliased as "avg_gloss_amt_usd."
- Calculates the average of the "poster_amt_usd" column in the "orders" table and is aliased as "avg_poster_amt_usd."
- A new column "total" is created by summing the average values of "standard_amt_usd", "gloss_amt_usd", and "poster_amt_usd".
- The tables are joined using a JOIN clause based on the "account_id" column.
- The query includes a GROUP BY clause, which groups the results by the "account_name" column.
- The final result is sorted in descending order by the "total" column using the ORDER BY clause.

```
SELECT accounts.name AS account_name,
       AVG(orders.standard_amt_usd) AS avg_standard_amt_usd,
       AVG(orders.gloss_amt_usd) AS avg_gloss_amt_usd,
      AVG(orders.poster_amt_usd) AS avg_poster_amt_usd,
       AVG(orders.standard_amt_usd) + AVG(orders.gloss_amt_usd) + AVG(orders.poster_amt_usd) AS total
FROM accounts
JOIN orders ON accounts.id = orders.account_id
GROUP BY account_name
ORDER BY total DESC;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

account_name	avg_standard_amt_usd	avg_gloss_amt_usd	avg_poster_amt_usd	total
Pacific Life	2702.7087500000002	147.92749999999998	28732.62	31583.25625
Fidelity National Financial	2299.6771428571424	28.889999999999997	13221.680000000002	15550.247142857144
CBS	1482.03	239.68	6926.36	8648.07
Berkshire Hathaway	5728.52	0	1745.8	7474.320000000001
Edison International	1636.72	29.959999999999997	5651.5199999999995	7318.2
Costco	823.35	1956.7625	4461.94	7242.0525
CHS	1387.5764285714286	323.14	5525.080000000001	7235.79642857143
Mondelez International	2423.4766666666665	161.035	3323.7866666666664	5908.298333333332
Core-Mark Holding	5424.261315789474	83.96684210526314	119.4494736842105	5627.6776315789475
Huntsman	2501.653333333333	699.0666666666666	2354.7999999999997	5555.5199999999995

This query calculates and summarizes the average sales amounts for different product types associated with each account, providing decision-makers with valuable insights into account performance. By aggregating average order amounts for standard, gloss, and poster products, it offers a comprehensive view of account revenue generation in a streamlined manner.

Task 9: Top Engagement Channels by Sales Representative.

Count Web Event Occurrences by Sales Representative and Channel, Sorted by Occurrence Count (Descending).

- The query retrieves data from three tables: "web_events", "accounts", and "sales_reps".
- It selects specific columns and assigns aliases to these columns:
- "name" from the "sales_reps" table is aliased as "sales_rep_name".
- "channel" from the "web_events" table is aliased as "channel".
- A count of occurrences of the "channel" column is calculated and aliased as "number_of_occurrences".
- The tables are joined using JOIN clauses based on their respective IDs:
- "web_events" and "accounts" are joined on the "account_id" column.
- "accounts" and "sales_reps" are joined on the "sales_rep_id" column.
- The query includes a GROUP BY clause, which groups the results by "sales_rep_name" and "channel". This groups the data by sales representative and channel.
- The final result is sorted in descending order by the "number_of_occurrences" column using the ORDER BY clause.

```
1 •  SELECT sales_reps.name AS sales_rep_name,
2    web_events.channel AS channel,
3    COUNT(web_events.channel) AS number_of_occurrences
4    FROM web_events
5    JOIN accounts ON web_events.account_id = accounts.id
6    JOIN sales_reps ON accounts.sales_rep_id = sales_reps.id
7    GROUP BY sales_rep_name, channel
8    ORDER BY number_of_occurrences DESC;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

sales_rep_name	channel	number_of_occurrences
Earlie Schleusner	direct	234
Vernita Plump	direct	232
Moon Torian	direct	194
Tia Amato	direct	185
Maren Musto	direct	184
Nelle Meaux	direct	179
Dorotha Seawell	direct	161

This query counts how often each sales representative uses various engagement channels and sorts them to display the most frequently utilized ones. It assists decision-makers in quickly identifying which channels are most used by their sales team, enabling better strategy planning.

Task 10: Annual Sales Trends Analysis.

Sum Total USD Amount of Orders by Year and Sort by Total USD Amount.

- The query retrieves data from the "orders" table.
- It selects specific columns and performs calculations:
- The EXTRACT function is used to extract the year from the "occurred_at" column and aliases it as "year".
- The SUM function calculates the total sum of "total_amt_usd" and aliases it as "total_usd".
- The query includes a GROUP BY clause, which groups the results by the "year". This groups the data by the year in which orders occurred.
- The final result is sorted in ascending order by the "total_usd" column using the ORDER BY clause.

```
1 •   SELECT EXTRACT(YEAR FROM occurred_at) AS year,
2     SUM(total_amt_usd) AS total_usd
3     FROM orders
4     GROUP BY year
5     ORDER BY total_usd;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

year	total_usd
2017	23424.030000000002
2013	177253.97
2014	1676803.620000001
2015	2659633.120000005
2016	5891837.739999996

This query swiftly extracts and aggregates annual sales totals, presenting key revenue trends in ascending order. From the data, it's evident that revenue has increased steadily over the years, with 2016 marking the highest total sales at $5,891,837.74, followed by 2015 with $2,659,633.12. This rapid analysis offers decision-makers valuable insights into the company's revenue trajectory.

Task 11: Monthly Revenue Analysis for Specific Years

Sum Total USD Amount of Orders by Year and Month, Filter by Specific Years, and Sort by Year.

- The query retrieves data from the "orders" table.
- It selects specific columns and performs calculations:
- The EXTRACT function is used to extract the year from the "occurred_at" column and aliases it as "year".
- Another EXTRACT function extracts the month from the "occurred_at" column and aliases it as "month".
- The SUM function calculates the total sum of "total_amt_usd" and aliases it as "total_usd".
- The query includes a WHERE clause that filters the results:
- It selects data where the extracted year from "occurred_at" is either 2013 or 2017.
- The results are grouped by both the "year" and "month" using the GROUP BY clause. This groups the data by year and month.
- The final result is sorted in ascending order by the "year" using the ORDER BY clause.

```
1 •  SELECT EXTRACT(YEAR FROM occurred_at) AS year,
2           EXTRACT(MONTH FROM occurred_at) AS month,
3           SUM(total_amt_usd) AS total_usd
4    FROM orders
5    WHERE EXTRACT(YEAR FROM occurred_at) = 2013 OR EXTRACT(YEAR FROM occurred_at) = 2017
6    GROUP BY year, month
7    ORDER BY year, month;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

year	month	total_usd
2013	12	177253.97
2017	1	23424.030000000002

This query efficiently extracts and sums monthly sales totals for specific years, providing valuable insights into revenue patterns over time. With its succinct design, it swiftly identifies revenue trends, such as $23424.03 in January 2017 and $177253.97 in December 2013, offering decision-makers actionable insights into the company's performance across different years.

Task 12: Unveiling Daily Sales.

Sum Total USD Amount of Orders by Year, Month, and Day for 2017, and Sort by Total USD Amount.

- The query retrieves data from the "orders" table.
- It selects specific columns and performs calculations:
- The EXTRACT function is used to extract the year from the "occurred_at" column and aliases it as "year".
- Another EXTRACT function extracts the month from the "occurred_at" column and aliases it as "month".
- A third EXTRACT function extracts the day from the "occurred_at" column and aliases it as "day".
- The SUM function calculates the total sum of "total_amt_usd" and aliases it as "total_usd."
- The query includes a WHERE clause that filters the results:
- It selects data where the extracted year from "occurred_at" is equal to 2017.
- The results are grouped by "year", "month", and "day" using the GROUP BY clause. This groups the data by year, month, and day.
- The final result is sorted in ascending order by the "total_usd" column using the ORDER BY clause.

```
1 •   SELECT EXTRACT(YEAR FROM occurred_at) AS year,
2            EXTRACT(MONTH FROM occurred_at) AS month,
3            EXTRACT(DAY FROM occurred_at) AS day,
4            SUM(total_amt_usd) AS total_usd
5     FROM orders
6     WHERE EXTRACT(YEAR FROM occurred_at) = 2017
7     GROUP BY year, month, day
8     ORDER BY total usd;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

year	month	day	total_usd
2017	1	1	23424.030000000002

This concise query focuses on extracting and aggregating daily sales data for the year 2017, providing valuable insights into revenue trends on a day-to-day basis. With precision and efficiency, it highlights that on January 1st, 2017, sales amounted to $23,424.03, indicating a strong start to the year.

Task 13: Unlocking New Year's Sales Data.

Sum Total USD Amount of Orders for January 1st and Sort by Total USD Amount.

- The query retrieves data from the "orders" table.
- It selects specific columns and performs calculations:
- The EXTRACT function is used to extract the year from the "occurred_at" column and aliases it as "year".
- Another EXTRACT function extracts the month from the "occurred_at" column and aliases it as "month".
- A third EXTRACT function extracts the day from the "occurred_at" column and aliases it as "day".
- The SUM function calculates the total sum of "total_amt_usd" and aliases it as "total_usd".
- The query includes a WHERE clause that filters the results:
- It selects data where the extracted month from "occurred_at" is equal to 1 (January).
- It also checks that the extracted day from "occurred_at" is equal to 1.
- The results are grouped by "year," "month," and "day" using the GROUP BY clause. This groups the data by year, month, and day.
- The final result is sorted in ascending order by the "total_usd" column using the ORDER BY clause.

```
1  SELECT year, month, day, total_usd
2  FROM (SELECT EXTRACT(YEAR FROM occurred_at) AS year,
3  EXTRACT(MONTH FROM occurred_at) AS month,
4  EXTRACT(DAY FROM occurred_at) AS day,
5  SUM(total_amt_usd) AS total_usd
6  FROM orders
7  GROUP BY year, month, day
8  ORDER BY total_usd) AS newtable
9  WHERE month = 1 AND day = 1;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

year	month	day	total_usd
2016	1	1	6499.1
2014	1	1	7453.77
2015	1	1	7747.95
2017	1	1	23424.030000000002

This succinct query drills down into sales data specifically for January 1st, shedding light on the revenue generated on the first day of the year. With concise precision, it reveals that across multiple years (2014, 2015, 2016, and 2017), January 1st saw significant sales figures ranging from $6,499.10 to $23,424.03. These insights offer a glimpse into consumer behavior and spending patterns at the onset of each year.

Task 14: Optimizing Walmart's Monthly Gloss Sales.

Find the Account with the Highest Monthly Gloss Total for Walmart.

- The query retrieves data from two tables: "accounts" and "orders".
- It selects specific columns and performs calculations:
- "name" from the "accounts" table is aliased as "account_name".
- The EXTRACT function is used to extract the year from the "occurred_at" column in the "orders" table and is aliased as "year".
- Another EXTRACT function extracts the month from the "occurred_at" column and is aliased as "month".
- The SUM function calculates the total sum of "gloss_amt_usd" from the "orders" table and is aliased as "gloss_total_usd".
- The tables are joined using a JOIN clause based on the "account_id".
- The query includes a WHERE clause that filters the results:
- It selects data where the "account_name" from the "accounts" table is 'Walmart'.
- The results are grouped by "account_name," "year," and "month" using the GROUP BY clause. This groups the data by account name, year, and month.
- The final result is sorted in descending order by the "gloss_total_usd" column using the ORDER BY clause.
- The query includes a LIMIT 1 clause to retrieve only the top result.

```sql
SELECT accounts.name AS account_name,
EXTRACT(YEAR FROM orders.occurred_at) AS year,
EXTRACT(MONTH FROM orders.occurred_at) AS month,
SUM(orders.gloss_amt_usd) AS gloss_total_usd
FROM accounts
JOIN orders ON accounts.id = orders.account_id
WHERE accounts.name = 'Walmart'
GROUP BY account_name, year, month
ORDER BY gloss_total_usd DESC
LIMIT 1;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

account_name	year	month	gloss_total_usd
Walmart	2016	10	1071.07

This query delves into Walmart's sales data to pinpoint the highest monthly gloss sales, emphasizing efficiency and precision. In this specific instance, the query reveals that in October 2016, Walmart recorded gloss sales totaling $1071.07, indicating a notable performance milestone for the company during that period.