E-commerce Database Design & SQL Analysis

1. Database Structuring

Proposed Database Schema

To build an efficient and scalable e-commerce system, we have structured our database with the following tables:

1. **Customers**: Stores customer details.
2. **Sellers**: Contains seller information.
3. **Orders**: Manages order details.
4. **Order Items**: Stores items within each order.
5. **Order Reviews**: Captures customer reviews.
6. **Products**: Contains product information.
7. **Category Name English**: Maps category names to English.
8. **Geolocation**: Stores geographical location data.
9. **Payments**: Manages order payment details.

Primary & Foreign Keys

Primary Key (PK): A unique identifier for each record in a table. It cannot be `NULL`.
Foreign Key (FK): A column that links one table to another by referencing its Primary Key.

1. **Customers**
   - Primary Key: `customer_id`
   - Foreign Key: None

2. **Orders**
   - Primary Key: `order_id`
   - Foreign Key:
     - `customer_id` → **Customers (`customer_id`)**

3. **Order Items**
   - Primary Key: `order_item_id`
   - Foreign Key:
     - `order_id` → **Orders (`order_id`)**
     - `product_id` → **Products (`product_id`)**

        - `seller_id` → **Sellers (`seller_id`)**

4. **Products**
    - Primary Key: `product_id`
    - Foreign Key:
        - `product_category_name` → **Category Name English (`product_category_name`)**

5. **Category Name English**
    - Primary Key: `product_category_name`
    - Foreign Key: None

6. **Order Reviews**
    - Primary Key: `review_id`
    - Foreign Key:
        - `order_id` → **Orders (`order_id`)**

7. **Sellers**
    - Primary Key: `seller_id`
    - Foreign Key:
        - `seller_zip_code_prefix` → **Geolocation (`geolocation_zip_code_prefix`)**

8. **Geolocation**
    - Primary Key: `geolocation_zip_code_prefix`
    - Foreign Key: None

9. **Payments (New Table Added)**
    - Primary Key: `order_id`
    - Foreign Key:
        - `order_id` → **Orders (`order_id`)**

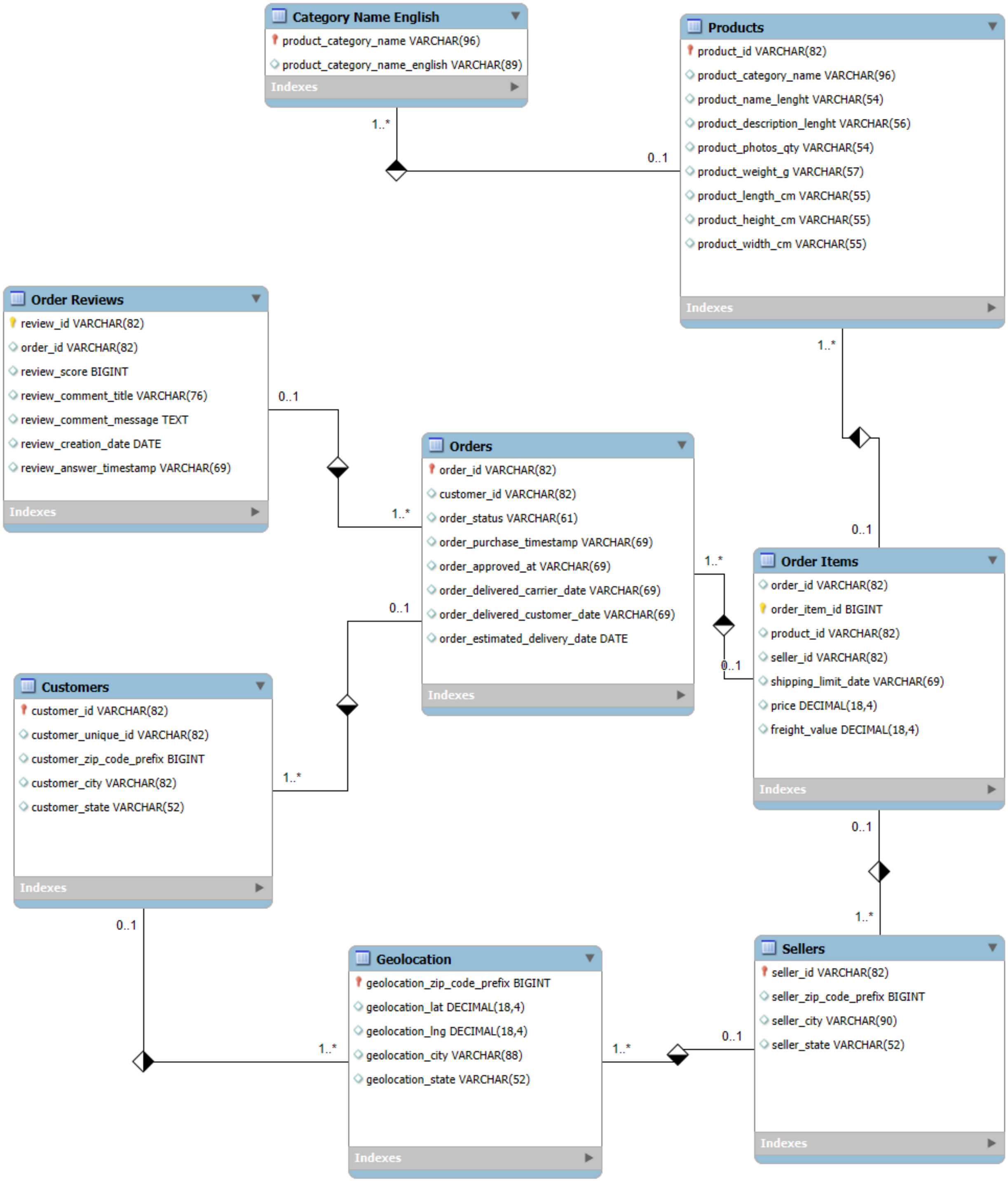
Indexing Strategy:

- Add indexes to frequently queried fields like `order_status`, `order_purchase_timestamp`, `product_category_name`, `customer_city`, and `seller_id` for performance optimization.

Normalization Consideration:

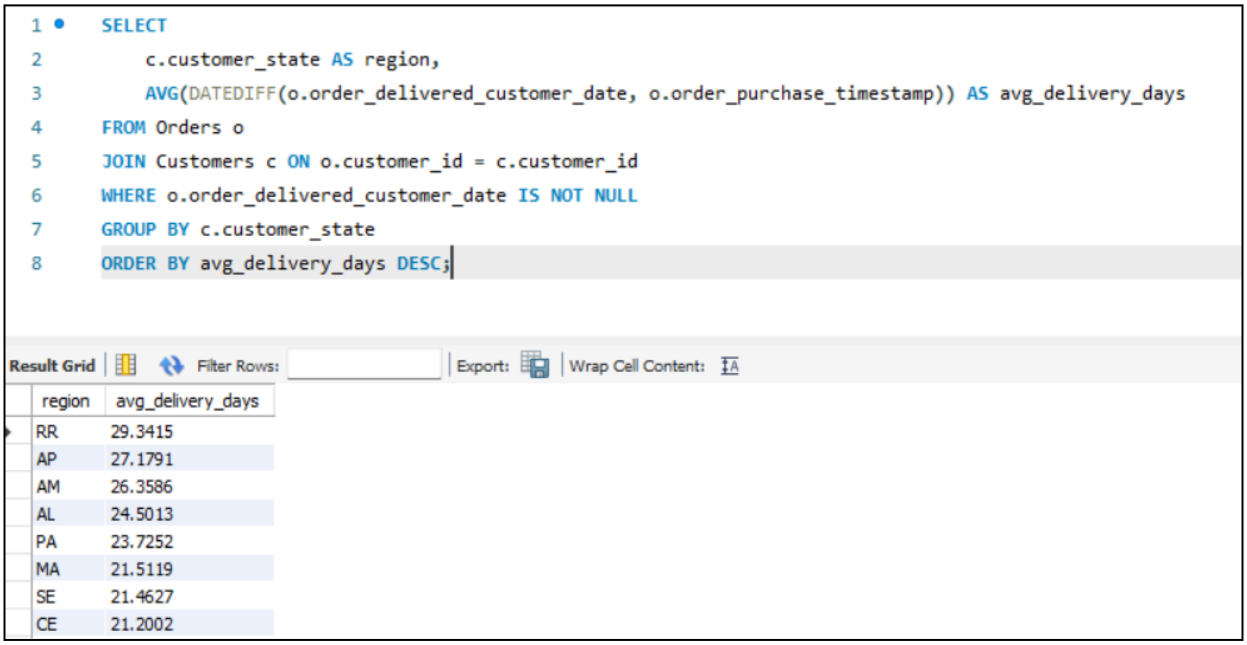
- Data appears well-normalized (3NF), ensuring minimal redundancy.
- Consider creating a `Region` table linking ZIP codes to predefined areas for faster geographic segmentation

2. ER Diagram


Category Name English
product_category_name VARCHAR(96)
product_category_name_english VARCHAR(89)
Indexes
Products
product_id VARCHAR(82)
product_category_name VARCHAR(96)
product_name_lenght VARCHAR(54)
product_description_lenght VARCHAR(56)
product_photos_qty VARCHAR(54)
product_weight_g VARCHAR(57)
product_length_cm VARCHAR(55)
product_height_cm VARCHAR(55)
product_width_cm VARCHAR(55)
Indexes
1..*
0..1
Order Reviews
review_id VARCHAR(82)
order_id VARCHAR(82)
review_score BIGINT
review_comment_title VARCHAR(76)
review_comment_message TEXT
review_creation_date DATE
review_answer_timestamp VARCHAR(69)
Indexes
1..*
0..1
Orders
order_id VARCHAR(82)
customer_id VARCHAR(82)
order_status VARCHAR(61)
order_purchase_timestamp VARCHAR(69)
order_approved_at VARCHAR(69)
order_delivered_carrier_date VARCHAR(69)
order_delivered_customer_date VARCHAR(69)
order_estimated_delivery_date DATE
Indexes
1..*
0..1
1..*
0..1
Order Items
order_id VARCHAR(82)
order_item_id BIGINT
product_id VARCHAR(82)
seller_id VARCHAR(82)
shipping_limit_date VARCHAR(69)
price DECIMAL(18,4)
freight_value DECIMAL(18,4)
Indexes
0..1
1..*
Customers
customer_id VARCHAR(82)
customer_unique_id VARCHAR(82)
customer_zip_code_prefix BIGINT
customer_city VARCHAR(82)
customer_state VARCHAR(52)
Indexes
0..1
1..*
0..1
1..*
Geolocation
geolocation_zip_code_prefix BIGINT
geolocation_lat DECIMAL(18,4)
geolocation_lng DECIMAL(18,4)
geolocation_city VARCHAR(88)
geolocation_state VARCHAR(52)
Indexes
1..*
0..1
Sellers
seller_id VARCHAR(82)
seller_zip_code_prefix BIGINT
seller_city VARCHAR(90)
seller_state VARCHAR(52)
Indexes

3. SQL Queries

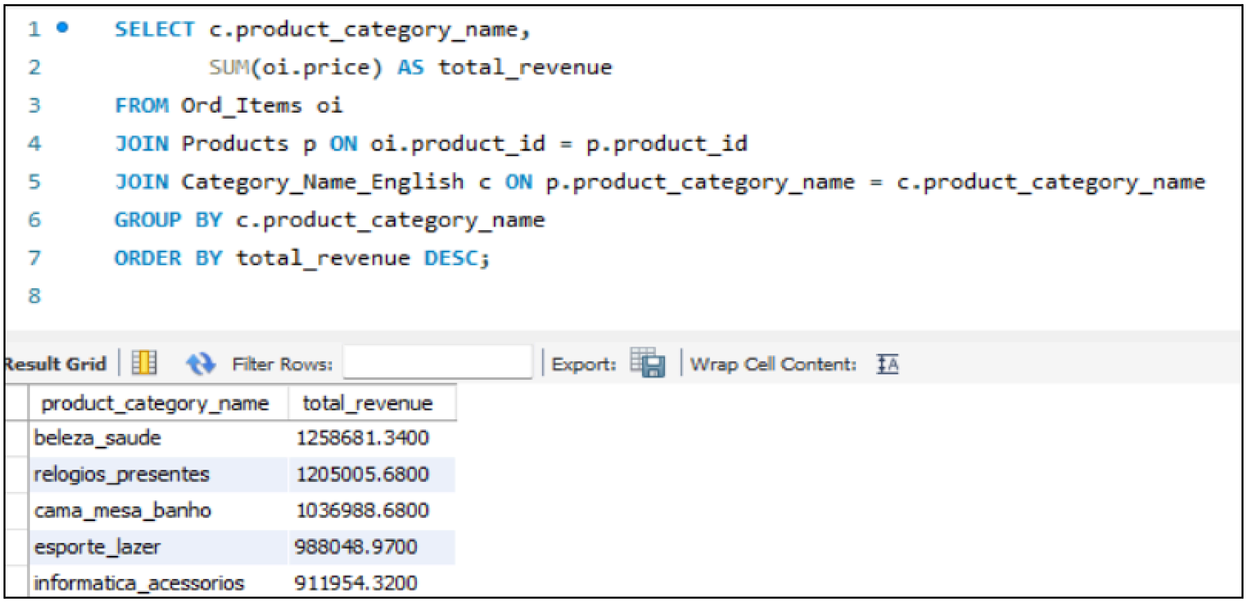
1. Average Delivery Time by Region



```sql
SELECT
    c.customer_state AS region,
    AVG(DATEDIFF(o.order_delivered_customer_date, o.order_purchase_timestamp)) AS avg_delivery_days
FROM Orders o
JOIN Customers c ON o.customer_id = c.customer_id
WHERE o.order_delivered_customer_date IS NOT NULL
GROUP BY c.customer_state
ORDER BY avg_delivery_days DESC;
```

region	avg_delivery_days
RR	29.3415
AP	27.1791
AM	26.3586
AL	24.5013
PA	23.7252
MA	21.5119
SE	21.4627
CE	21.2002

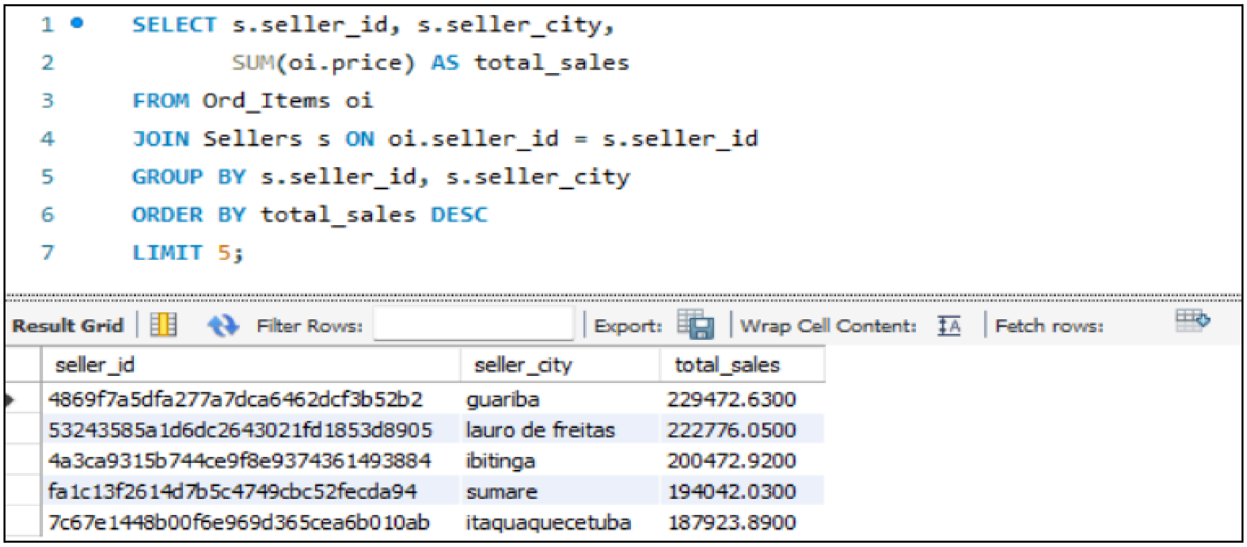
2. Total Revenue by Product Category



```sql
SELECT c.product_category_name,
       SUM(oi.price) AS total_revenue
FROM Ord_Items oi
JOIN Products p ON oi.product_id = p.product_id
JOIN Category_Name_English c ON p.product_category_name = c.product_category_name
GROUP BY c.product_category_name
ORDER BY total_revenue DESC;
```

product_category_name	total_revenue
beleza_saude	1258681.3400
relogios_presentes	1205005.6800
cama_mesa_banho	1036988.6800
esporte_lazer	988048.9700
informatica_acessorios	911954.3200

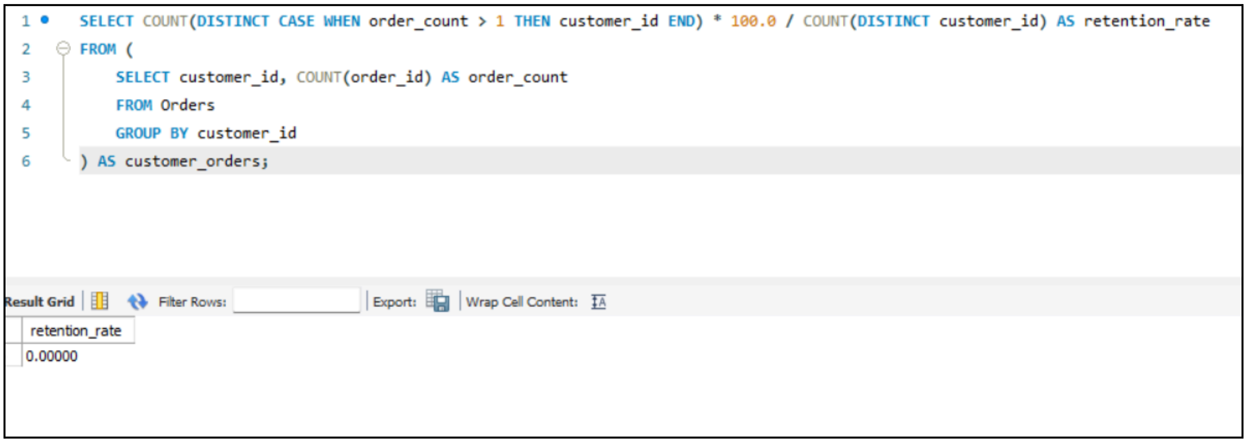
3. Top 5 Performing Sellers Based on Revenue



```sql
SELECT s.seller_id, s.seller_city,
       SUM(oi.price) AS total_sales
FROM Ord_Items oi
JOIN Sellers s ON oi.seller_id = s.seller_id
GROUP BY s.seller_id, s.seller_city
ORDER BY total_sales DESC
LIMIT 5;
```

seller_id	seller_city	total_sales
4869f7a5dfa277a7dca6462dcf3b52b2	guariba	229472.6300
53243585a1d6dc2643021fd1853d8905	lauro de freitas	222776.0500
4a3ca9315b744ce9f8e9374361493884	ibitinga	200472.9200
fa1c13f2614d7b5c4749cbc52fecda94	sumare	194042.0300
7c67e1448b00f6e969d365cea6b010ab	itaquaquecetuba	187923.8900

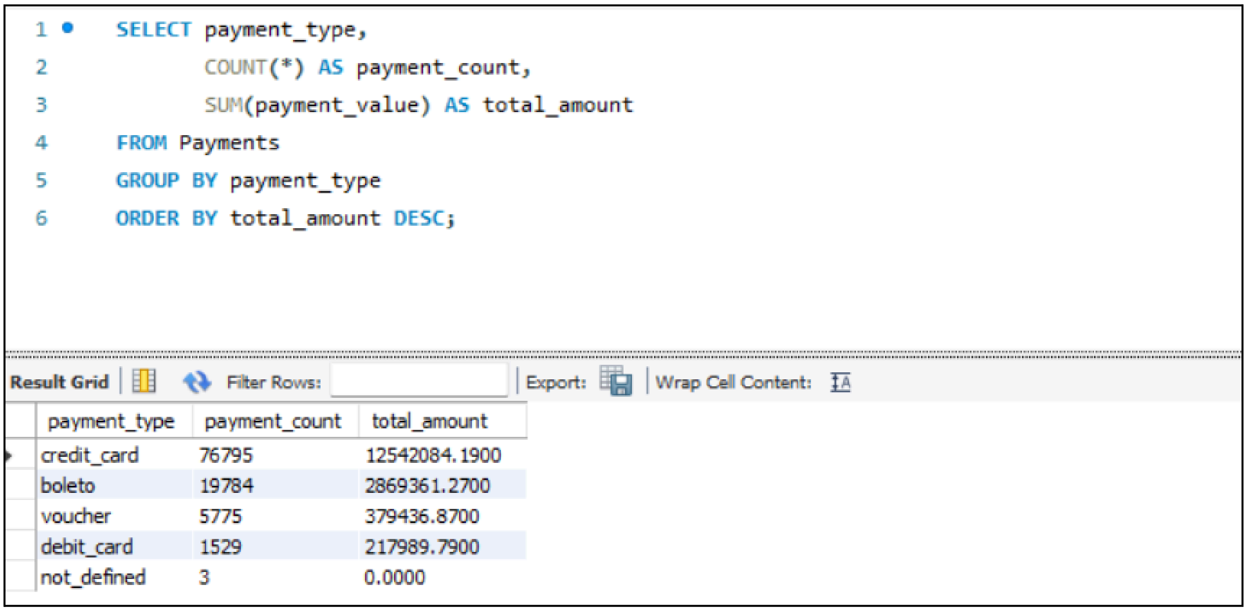
4. Customer Retention Rate



```sql
SELECT COUNT(DISTINCT CASE WHEN order_count > 1 THEN customer_id END) * 100.0 / COUNT(DISTINCT customer_id) AS retention_rate
FROM (
    SELECT customer_id, COUNT(order_id) AS order_count
    FROM Orders
    GROUP BY customer_id
) AS customer_orders;
```

retention_rate
0.00000

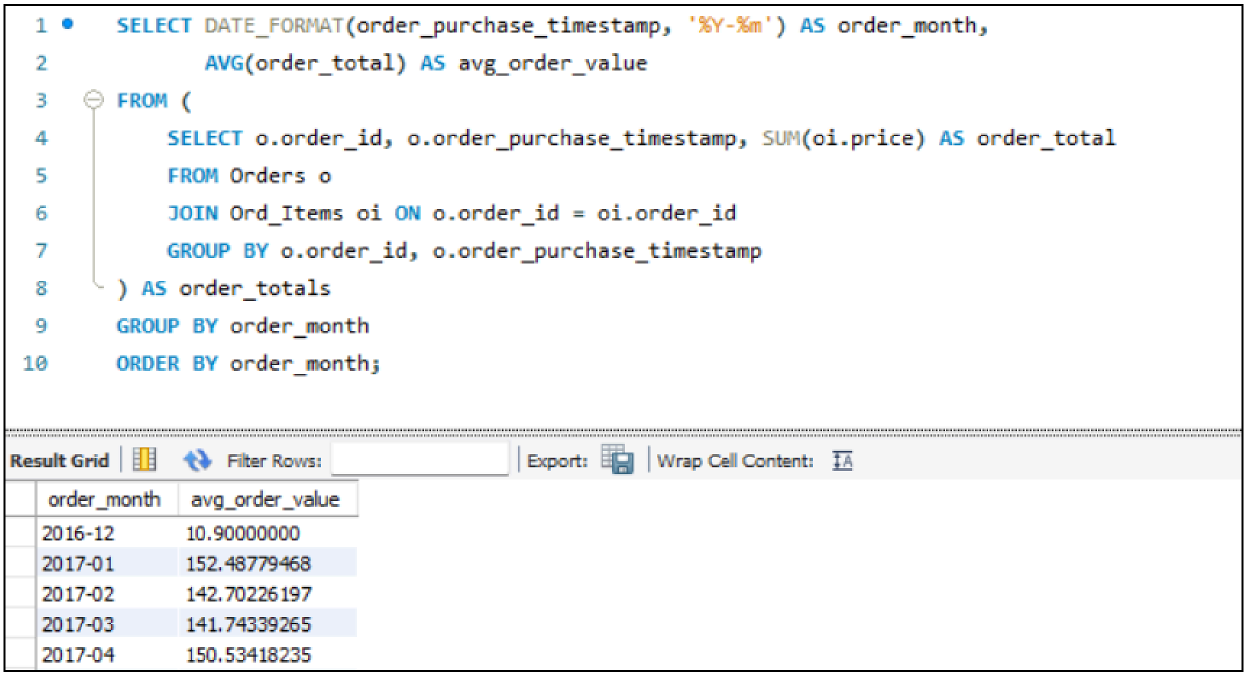
5. Payment Distribution by Type



```sql
SELECT payment_type,
       COUNT(*) AS payment_count,
       SUM(payment_value) AS total_amount
FROM Payments
GROUP BY payment_type
ORDER BY total_amount DESC;
```

payment_type	payment_count	total_amount
credit_card	76795	12542084.1900
boleto	19784	2869361.2700
voucher	5775	379436.8700
debit_card	1529	217989.7900
not_defined	3	0.0000

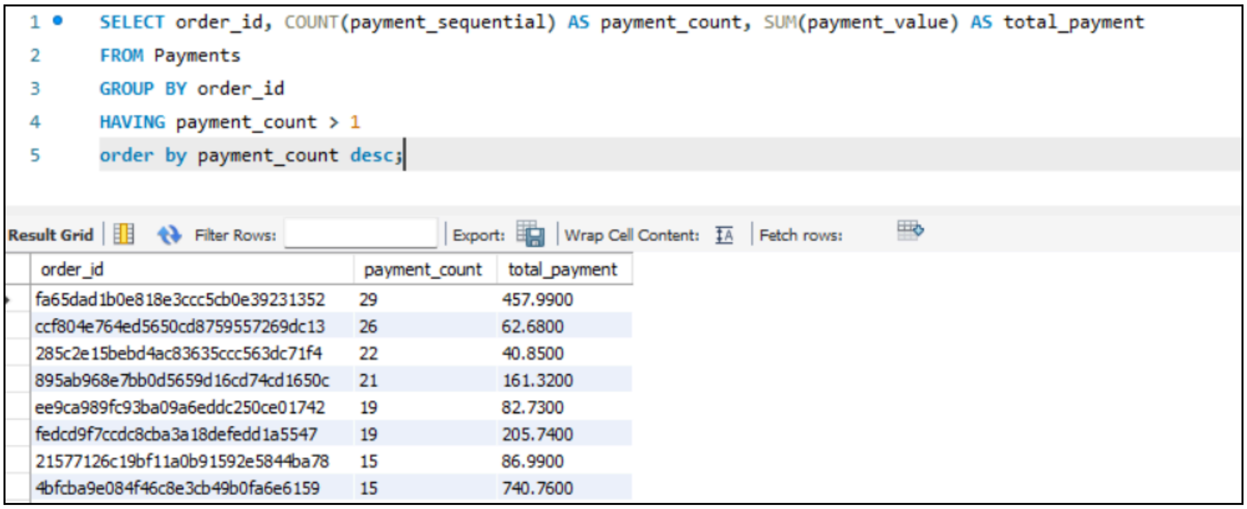
6. Average Order Value by Month



```sql
SELECT DATE_FORMAT(order_purchase_timestamp, '%Y-%m') AS order_month,
       AVG(order_total) AS avg_order_value
FROM (
    SELECT o.order_id, o.order_purchase_timestamp, SUM(oi.price) AS order_total
    FROM Orders o
    JOIN Ord_Items oi ON o.order_id = oi.order_id
    GROUP BY o.order_id, o.order_purchase_timestamp
) AS order_totals
GROUP BY order_month
ORDER BY order_month;
```

order_month	avg_order_value
2016-12	10.90000000
2017-01	152.48779468
2017-02	142.70226197
2017-03	141.74339265
2017-04	150.53418235

7. Fraudulent Payment Detection (Orders with Multiple Payments)



```sql
SELECT order_id, COUNT(payment_sequential) AS payment_count, SUM(payment_value) AS total_payment
FROM Payments
GROUP BY order_id
HAVING payment_count > 1
order by payment_count desc;
```

order_id	payment_count	total_payment
fa65dad1b0e818e3ccc5cb0e39231352	29	457.9900
ccf804e764ed5650cd8759557269dc13	26	62.6800
285c2e15bebd4ac83635ccc563dc71f4	22	40.8500
895ab968e7bb0d5659d16cd74cd1650c	21	161.3200
ee9ca989fc93ba09a6eddc250ce01742	19	82.7300
fedcd9f7ccdc8cba3a18defedd1a5547	19	205.7400
21577126c19bf11a0b91592e5844ba78	15	86.9900
4bfcba9e084f46c8e3cb49b0fa6e6159	15	740.7600

3. Conclusion

This e-commerce database is designed to efficiently store and analyze business data. The structured indexing, normalized schema, and analytical SQL queries allow for insightful business decision-making, from revenue trends to customer retention and fraudulent payment detection.