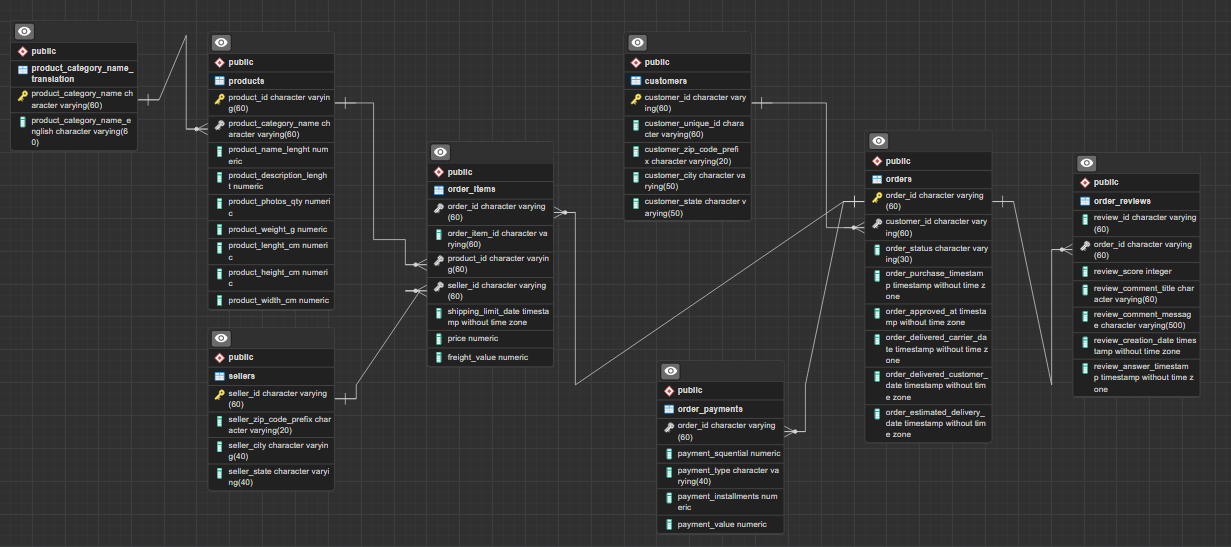
**SQL OLIST PROJE ÇÖZÜMÜ**



In the ERD diagram, there are one-to-many relationships between the "orders" table and the "order\_items", "order\_payments" and "order\_reviews" tables. "order\_id" is located as the primary key in the "orders" table and this field is "order\_items", “ It is used as a foreign key in the “order\_payments” and “order\_reviews” tables.

There is also a one-to-many relationship between the "customers" table and the "orders" table. While “customer\_id” is found as the primary key in the “customers” table, it is used as a foreign key in the “orders” table.

There is also a one-to-many relationship between the “sellers” table and the “order\_items” table. While “seller\_id” is used as the primary key in the “sellers” table, it is used as a foreign key in the “order\_items” table.

There is a one-to-many relationship from the “products” table to the “order\_items” table. While “product\_id” is used as the primary key in the “products” table, it is used as a foreign key in the “order\_items” table.

Finally, from the “product\_category\_name\_translation” table to the “products” table

There is a one-to-many relationship. While “product\_category\_name” is found as the primary key in the “product\_category\_name” table, it is used as a foreign key in the “products” table.

**Case 1: Order Analysis**

**Question 1:**

Analyze the monthly distribution of orders. “The order\_approved\_at” field should be used for date data.

**SQL Query:**

**SELECT (date\_trunc('month', order\_approved\_at))::date AS month,**

**count(\*) AS orders\_count**

**FROM orders**

**GROUP BY month**

**ORDER BY month;**

**Explanation:**

* The date\_trunc function converts the dates in the “order\_approved\_at” column to the first day of the month and is converted to date format with the expression ‘::date’.
* AS month gives the name “month” to the date converted to the first day of the month obtained in the previous step.
* COUNT(\*) counts the number of orders for each month.
* GROUP BY month groups results by month.
* ORDER BY month sorts the query result by month.

When the chart is examined, the lowest orders were realized in September and December 2016. The number of orders started to increase in January 2017. The highest number of orders was realized in November 2017.

**Question 2:**

Examine the order numbers in the order status breakdown on a monthly basis. Visualize the output of the query with Excel. Are there months where there is a dramatic decline or rise? Examine and interpret the data.

**SQL Query:**

**SELECT**

**(date\_trunc('month', order\_approved\_at))::date AS month,**

**order\_status,**

**COUNT(order\_id) AS orders\_count**

**FROM orders**

**WHERE order\_status='delivered'**

**GROUP BY month,order\_status**

**ORDER BY month;**

**Explanation:**

* This query divides the dates in the order\_approved\_at column into months and counts how many orders there are according to order\_status with each month.
* The date\_trunc function converts the dates in the "order approved at" column to the first day of the month and is converted to date format with the expression '::date'.
* AS month gives the name “month” to the date converted to the first day of the month obtained in the previous step.
* COUNT(order\_id) counts the number of order\_id for each month and order\_status combination.
* With the WHERE order\_status statement, it only returns the number of delivered orders.
* GROUP BY month,order\_status groups the results by month and order\_status.
* ORDER BY month sorts the query result by month.

As seen in the chart, the number of orders increased and reached 715 in January 2017. In the following period, in November 2017, there was an increase of 61% compared to the number of orders in the previous month. In December 2017, it decreased again by 21%. In January 2018, it increased again by 23%. When the chart is examined, the highest number of orders was reached in November 2017.

**SQL QUERY**

**SELECT**

**(date\_trunc('month', order\_approved\_at))::date AS month,**

**order\_status,**

**COUNT(order\_id) AS orders\_count**

**FROM orders**

**WHERE order\_status='canceled' OR order\_status='unavailable'**

**GROUP BY month,order\_status**

**ORDER BY month;**

**Explanation:**

* This query divides the dates in the order\_approved\_at column into months and counts how many orders there are according to order\_status with each month.
* The date\_trunc function converts the dates in the "order approved at" column to the first day of the month and is converted to date format with the expression '::date'.
* AS month gives the name “month” to the date converted to the first day of the month obtained in the previous step.
* COUNT(order\_id) counts the number of order\_id for each month and order\_status combination.
* Returns the number of canceled and unavailable orders with the WHERE order status statement.
* GROUP BY month,order\_status groups the results by month and order\_status.
* ORDER BY month sorts the query result by month.

In this chart, we see the number of failed orders by month. The more orders are placed, the more likely the order will fail.

**Question 3:**

Examine the order numbers in the product category breakdown. What are the prominent categories on special days? For example, New Year's Eve, Valentine's Day...

**SQL Query:**

**SELECT**

**product\_category\_name,**

**(date\_trunc('month', orders.order\_approved\_at))::date AS order\_month,**

**COUNT(orders.order\_id)**

**FROM orders**

**LEFT JOIN order\_items ON orders.order\_id=order\_items.order\_id**

**LEFT JOIN products ON order\_items.product\_id=products.product\_id**

**WHERE**

**products.product\_category\_name IS NOT NULL**

**GROUP BY product\_category\_name,order\_month**

**ORDER BY order\_month**

**Explanation:**

This query calculates the total number of orders by product category and order month.

It is used to examine the total number of orders by product category and months by combining the “orders” table with the “order\_items” and “products” tables.

* LEFT JOIN is used to establish relationships between tables.
* The “orders” table and the “order\_items” table are combined via the “order\_id” column.
* The “order\_items” table and the “products” table are combined via the “product\_id” column. A connection is established between the “products” and “orders” tables through the “order\_items” table.
* products.product\_category\_name IS NOT NULL returns non-NULL values ​​from “product\_category\_name ”.
* GROUP BY product\_category\_name, order\_month groups results by product category and order month.
* ORDER BY order\_month sorts the query result by order months.

The chart shows the top 10 categories with the most orders.

Highest ordered categories:

1. cama\_mesa\_banho 11.105
2. beleza\_saude 9.670
3. esporte\_lazer 8.641

**Special days celebrated in Brazil;**

* New Year's Day (Ano Novo) - 1 January
* Valentine’s Day – 14 February
* All Souls Day (Dia de Finados) - 2 November
* Black Friday - 11 November
* Noel (Natal) - 25 December

The first 3 categories that stand out on Special Days are;

**Valentine’s Day:**

02.2017

1. moveis\_decoracao 253
2. beleza\_saude 172
3. cama\_mesa\_banho 162

02.2018

1. informatica\_acessorios 991
2. cama\_mesa\_banho 708
3. beleza\_saude 663

In this regard, we may consider organizing discounts or promotions for the following February and increasing the stocks of these categories. This can be a strategic step to meet customer demand and increase sales.

**New Year**

12.2017

1. cama\_mesa\_banho 568
2. esporte\_lazer 509
3. beleza\_saude 483

12.2018

1. cama\_mesa\_banho 568
2. esporte\_lazer 509
3. beleza\_saude 483

In this regard, we may consider organizing discounts or promotions for the following December and increasing the stocks of these categories. This can be a strategic step to meet customer demand and increase sales.

**Black Friday And All Souls Day**

11.2017

1. cama\_mesa\_banho 956
2. moveis\_decoracao 776
3. esporte\_lazer 604

In this regard, we may consider organizing discounts or promotions for next October and increasing the stocks of these categories. This can be a strategic step to meet customer demand and increase sales.

**Question 4:**

- Examine the order numbers on the basis of days of the week (Monday, Thursday, ....) and month days (such as the 1st, 2nd of the month). Create a visual in Excel with the output of the query you wrote and interpret it.

**1. QUERY Weekdays;**

**SELECT**

**to\_char(order\_approved\_at,'day')AS day\_name,**

**COUNT(order\_id) AS order\_count**

**FROM orders**

**GROUP BY day\_name**

**ORDER BY order\_count DESC**

**Explanation:**

This query counts how many orders there are for each day by converting the dates in the order\_approved\_at column to days of the week.

* The to\_char function converts the dates in the “order\_approved\_at” column as day names according to the “day” parameter. In other words, it is used to obtain which day the dates are.
* AS day\_name defines the day name obtained in the previous step with the name “day\_name”.
* COUNT(order\_id) counts the number of order\_id for each day name.
* GROUP BY day\_name groups results by day name.
* ORDER BY order\_count DESC sorts the query result in descending order by order count.

When the days of the week are examined, the day with the most sales is Tuesday. The day with the least sales is Sunday.

**2.QUERY Month Day**

**SELECT**

**to\_char(order\_approved\_at,'DD') AS month\_day,**

**COUNT(order\_id) AS order\_count**

**from orders**

**GROUP BY month\_day**

**Explanation:**

This query counts how many orders there are for each day by converting the dates in the order approved at column to days of the month.

• The to\_char function converts the dates in the “order\_approved\_at” column to the day of the month based on the “DD” parameter. In other words, it is used to obtain which day the dates are.

• AS month\_day identifies the day number obtained in the previous step with the name “month\_day”.

• COUNT(order\_id) counts the number of order\_id for each day number.

• GROUP BY month\_day groups results by day number.

* There are 160 null values.

When the days of the month are examined, the 24th day is the first day with the most orders, followed by the 5th day. The minimum order day is the 31st day. This may be related to salary receipt dates.

**Case 2: Customer Analysis**

**Question 1:**

- In which cities do customers shop more? Determine the customer's city as the city from which they place the most orders and perform the analysis accordingly.

**SQL QUERY:**

**WITH most\_order\_city AS**

**(WITH customer\_with\_city AS**

**(SELECT**

**c.customer\_id ,**

**customer\_city,**

**COUNT(DISTINCT order\_id) AS order\_count**

**FROM customers c**

**LEFT JOIN orders o ON c.customer\_id=o.customer\_id**

**GROUP BY 1,2**

**ORDER BY order\_count desc),**

**row\_num AS**

**(**

**SELECT**

**customer\_id,**

**customer\_city,**

**order\_count,**

**ROW\_NUMBER () OVER (PARTITION BY customer\_id ORDER BY order\_count DESC ) AS rn\_order\_count**

**FROM**

**customer\_with\_city**

**)**

**SELECT \* FROM row\_num**

**WHERE rn\_order\_count=1)**

**SELECT c.customer\_city,**

**COUNT(order\_id) AS order\_count**

**FROM customers c**

**LEFT JOIN orders o ON c.customer\_id=o.customer\_id**

**LEFT JOIN most\_order\_city on most\_order\_city.customer\_id=c.customer\_id**

**GROUP BY c.customer\_city**

**ORDER BY order\_count DESC**

**Explanation:**

This query analyzes the number of customers' orders by city by joining the "customers" and "orders" tables. In terms of performance, it is more convenient to use WITH.

1. With the first WITH statement, a table named “customer\_with\_city” is created. This table combines the “customers” table and the “orders” table via customer\_id to find the cities of the customers and the number of orders in these cities. It counts orders with the COUNT function and DISTINCT expression. The DISTINCT statement returns only non-duplicate values.
2. With the GROUP BY statement, it groups according to customer\_id and city\_name and sorts them in descending order according to the order\_count column.
3. With the second WITH statement, a table named “row\_num” is created. This table divides the data from the “customer\_with\_city” table by customer\_id, creating a sequential numbering (rn\_order\_count) based on the number of orders for each customer. In this way, it finds the cities with the highest number of orders for each customer.
4. In the main query, the data from the table named “most\_order\_city” and the “customers” and “orders” tables are combined. While information about all customers is obtained using LEFT JOIN, the cities with the highest number of orders are matched according to the customer IDs in the "most\_order\_city" table. The number of orders in each city is calculated using the COUNT function. It is grouped by city with the GROUP BY statement and sorted in descending order by the number of orders with DESC.

The chart shows the top 10 cities where customers place the most orders. Sao Paulo was the city with the most orders. Different order suggestions may be offered depending on the city the person is located in.

**Case 3: Sales Analysis**

**Question 1:**

- Who are the sellers who deliver orders to customers in the fastest way? Bring top 5. Examine and comment on the order numbers of these sellers and the comments and ratings on their products.

**SQL QUERY:**

**WITH day\_diffrence AS**

**(SELECT s.seller\_id, order\_purchase\_timestamp , order\_delivered\_customer\_date,**

**order\_delivered\_customer\_date::date-order\_purchase\_timestamp::date AS day\_difference,**

**ROW\_NUMBER()OVER(PARTITION BY s.seller\_id ORDER BY order\_delivered\_customer\_date::date-order\_purchase\_timestamp::date ASC ) AS rn**

**FROM orders o**

**LEFT JOIN order\_items oi ON o.order\_id=oi.order\_id**

**LEFT JOIN sellers s ON oi.seller\_id=s.seller\_id**

**ORDER BY day\_difference)**

**SELECT \* FROM day\_diffrence**

**WHERE rn=1**

**LIMIT 5**

**Explanation:**

As a result of this query, the day difference between each vendor's delivery dates and order dates is calculated and the 5 vendors with the lowest value are selected.

This query calculates the day difference between the delivery dates and order dates of all orders containing sellers by combining the "orders", "order\_items" and "sellers" tables.

1. With the first WITH statement, a table named “day\_difference” is created. This table combines the “orders”, “order\_items” and “sellers” tables and calculates the day difference between each seller's delivery dates and order dates. The “order\_items” table is used to join the “orders” and “sellers” tables. Using the ROW\_NUMBER function, the day difference value for each vendor is numbered in ASC (increasing) order. The query result is sorted by seller IDs and day differences.
2. In the main query, data from the “day\_difference” table is used. It selects sellers with rn (row number) value 1 (i.e. those with the lowest day difference). With the LIMIT 5 statement, it retrieves only the first 5 results.

As a result of this query, the day difference between each vendor's delivery dates and order dates is calculated and the 5 vendors with the lowest value are selected.

**NOTE:** The row\_number function was used to prevent the same seller from appearing in the top 5 twice with more than one fast sale.

| **Top 5 Seller\_id** |
| --- |
| 46dc3b2cc0980fb8ec44634e21d2718e |
| 67bf6941ba2f1fa1d02c375766bc3e53 |
| 5e063e85d44b0f5c3e6ec3131103a57e |
| 5a8e7d5003a1f221f9e1d6e411de7c23 |
| 5011f0d93373a4c5753adf58ca77af8d |

Examine and comment on the order numbers of these sellers and the comments and ratings on their products.

**ORDER COUNTS**

**WITH top\_5\_order\_count AS**

**(WITH day\_diffrence AS**

**(SELECT s.seller\_id,order\_purchase\_timestamp , order\_delivered\_customer\_date,**

**order\_delivered\_customer\_date::DATE-order\_purchase\_timestamp ::DATE AS day\_diffrence,**

**ROW\_NUMBER()OVER(PARTITION BY S.SELLER\_ID ORDER BY order\_delivered\_customer\_date::DATE-order\_purchase\_timestamp ::DATE ASC ) AS rn,**

**COUNT( o.order\_id) OVER (PARTITION BY s.seller\_id) AS order\_count FROM orders o**

**LEFT JOIN order\_items oi ON o.order\_id= oi.order\_id**

**LEFT JOIN sellers s ON oi.seller\_id=s.seller\_id**

**ORDER BY day\_diffrence)**

**SELECT \* FROM day\_diffrence**

**WHERE rn=1**

**LIMIT 5)**

**SELECT seller\_id,**

**day\_diffrence,**

**rn,**

**order\_count**

**FROM top\_5\_order\_count**

**Explanation:**

This SQL code creates a temporary table named “top\_5\_order\_count”. This table depends on another temporary table called “day\_diffrence”.

The “day\_diffrence” table joins the data by joining the “sellers” table with the “order\_items” and “orders” tables. It groups sellers using the “seller\_id” field and creates a result set that includes the date of the first order received by each seller, delivery date, day difference, order (rn) and number of orders. Results are sorted by day difference.

The “top\_5\_order\_count” table contains only the first 5 rows of the “day\_diffrence” table.

Finally, it selects the “seller\_id”, “day\_diffrence”, “rn” and “order\_count” fields from the “top\_5\_order\_count” table.

**2. SQL QUERY**

**WITH top\_5\_order\_count AS**

**(**

**WITH day\_diffrence AS**

**(**

**SELECT s.seller\_id, order\_purchase\_timestamp, order\_delivered\_customer\_date,**

**order\_delivered\_customer\_date::DATE - order\_purchase\_timestamp::DATE AS day\_diffrence,**

**ROW\_NUMBER() OVER (PARTITION BY s.seller\_id ORDER BY order\_delivered\_customer\_date::DATE - order\_purchase\_timestamp::DATE ASC) AS rn,**

**COUNT(o.order\_id) OVER (PARTITION BY s.seller\_id) AS order\_count**

**FROM orders o**

**LEFT JOIN order\_items oi ON o.order\_id = oi.order\_id**

**LEFT JOIN sellers s ON oi.seller\_id = s.seller\_id**

**ORDER BY day\_diffrence**

**)**

**SELECT \* FROM day\_diffrence WHERE rn = 1 LIMIT 5**

**)**

**SELECT tc.seller\_id, tc.order\_count, review\_score,review\_comment\_message,**

**CASE**

**WHEN review\_score = 1 THEN 'cok\_dusuk\_skor'**

**WHEN review\_score = 2 THEN 'dusuk\_skore '**

**WHEN review\_score = 3 THEN 'orta\_skor'**

**WHEN review\_score = 4 THEN 'yuksek\_skor'**

**WHEN review\_score = 5 THEN 'cok\_yuksek\_skor'**

**END AS segment**

**FROM top\_5\_order\_count tc**

**LEFT JOIN order\_items oi ON tc.seller\_id = oi.seller\_id**

**LEFT JOIN order\_reviews orw ON oi.order\_id = orw.order\_id**

**WHERE review\_score BETWEEN 1 AND 5;**

**Explanation:**

It was written to examine sellers' order numbers and product comments and ratings. Thanks to this query, sellers can be examined according to their review\_scores and customer comments.

• First, a subquery called inner WITH day\_diffrence is defined. This subquery calculates the difference between the delivery dates of orders and data vendors by combining the orders, order\_items, and sellers tables. Sellers are grouped by seller\_id and select orders with the smallest difference. Sorting is done for each seller with the rn field and only the 5 sellers with the smallest difference are selected.

• A second main query called WITH top\_5\_order\_count is defined. This query selects only the top 5 merchants with column rn 1, based on the results returned from the day\_diffrence subquery.

• Next, the main query joins the top\_5\_order\_count table with the order\_items and order\_reviews tables. Returns sellers' order counts and related review\_score, review\_comment\_message and segment values.

• The CASE WHEN structure assigns the appropriate segment value for each review\_score based on the review\_score values. For example, if review\_score is 1, the segment is assigned as 'very low score'.

• Finally, it only selects data with a review\_score between 1 and 5, with a WHERE condition.

In this way, the query returns order counts and corresponding comments from the top 5 sellers who have the shortest delivery time and belong to certain rating segments.

**Question 2:**

* Which sellers sell products from more categories? Do sellers with many categories also have a high number of orders?

**SQL QUERY:**

**SELECT s.seller\_id,**

**COUNT(DISTINCT product\_category\_name) AS category\_count,**

**COUNT( oi.order\_id) AS order\_count**

**FROM order\_items oi**

**LEFT JOIN products P ON oi.product\_id = p.product\_id**

**LEFT JOIN sellers s ON oi.seller\_id = s.seller\_id**

**GROUP BY s.seller\_id**

**ORDER BY category\_count DESC**

**LIMIT 5**

**Açıklama:**

In this query, first the order\_items, products and sellers tables are combined.

With LEFT JOIN, the order\_items table is joined with the products table based on the product\_id field and gets the product\_category\_name field.

With LEFT JOIN, the order\_items table is joined with sellers table based on seller\_id field and the seller information is obtained.

s.seller\_id is retrieved in the SELECT statemen.

With the expression COUNT(DISTINCT product\_category\_name) AS category\_count, the number of different categories is calculated for each seller. That's why DISTINCT is used.

COUNT(oi.order\_id) AS order\_count ifadesiyle her bir satıcı için sipariş sayısı hesaplanıyor.

The number of orders for each seller is calculated with COUNT(oi.order\_id) AS order\_count expression.

With the GROUP BY s.seller\_id statement, the results are grouped by seller.

With the ORDER BY category\_count DESC statement, descending order is made according to the number of categories.

With the LIMIT 5 statement, only the first 5 rows are retrieved.

As a result, this query calculates how many products sellers have in different categories and how many orders they received, listing the top 5 sellers by number of categories.

|  | **seller\_id** | **category\_count** | **order\_count** |
| --- | --- | --- | --- |
| 1. | b2ba3715d723d245138f291a6fe42594 | 27 | 363 |
| 2. | 4e922959ae960d389249c378d1c939f5 | 23 | 454 |
| 3. | 955fee9216a65b617aa5c0531780ce60 | 23 | 1499 |
| 4. | 1da3aeb70d7989d1e6d9b0e887f97c23 | 21 | 328 |
| 5. | f8db351d8c4c4c22c6835c19a46f01b0 | 19 | 724 |

The table above shows the seller\_ids of the top five sellers selling in the most categories, the number of categories they sell in and the number of orders they receive.

According to the table, although the seller who ranked 1st made sales in 27 categories, he remained in the 4th place in the group because the number of orders was 363. Although the seller ranked 3rd sold in fewer categories with 23 categories, it was the seller with the highest number of orders with 1499 orders.

**Case 4: Payment Analysis**

**Question 1:**

- Which region do the users with the highest number of installments live in? Interpret this output.

**SQL QUERY**

**WITH excess\_installment AS**

**(SELECT c.customer\_id,**

**payment\_installments,**

**customer\_state**

**FROM orders o**

**LEFT JOIN order\_payments op ON o.order\_id = op.order\_id**

**LEFT JOIN customers C ON o.customer\_id = c.customer\_id**

**WHERE payment\_installments > 12)**

**SELECT customer\_state,**

**COUNT(customer\_id) AS customer\_count**

**FROM excess\_installment**

**GROUP BY customer\_state**

**ORDER BY customer\_count DESC**

**Explanation:**

First, a temporary table (WITH excess\_installment AS) named excess\_installment is created.

This table combines orders, order\_payments, and customers tables.

With LEFT JOIN, orders table joins the order\_payments table based on the order\_id field and gets the payment\_installments field.

With LEFT JOIN, orders table joins customers table based on the customer\_id field and obtains the customer information.

Selects records with the number of installments greater than 12, provided that payment\_installments > 12.

Retrieves customer\_state field and customer\_id in the SELECT statement.

GROUP BY customer\_state statement groups the results by the customer\_state field.

Calculates the number of customers for each customer case with the expression COUNT(customer\_id) AS customer\_count.

ORDER BY customer count DESC performs a descending order according to the number of customers.

As a result, this query groups the states (customer\_state) of customers with installments greater than 12 and lists them sorted from highest to lowest according to the number of customers.

As seen in the chart, the sellers who use installments the most when making payments live in the SP region.

**Question 2:**

- Calculate the number of successful orders and total successful payment amount according to payment type. Rank them in order from the most used payment type to the least.

**SQL QUERY**

**SELECT payment\_type, COUNT(o.order\_id) AS succ\_order\_count ,**

**SUM(price) AS sum\_succ\_price**

**FROM orders o**

**LEFT JOIN order\_items oi ON o.order\_id=oi.order\_id**

**LEFT JOIN order\_payments op ON o.order\_id=op.order\_id**

**WHERE order\_status='delivered'**

**GROUP BY payment\_type**

**ORDER BY succ\_order\_count DESC**

**Açıklama:**

Payment types in this query; "not\_defined","boleto","debit\_card","voucher","credit\_card" is selected. Successful payment is received as 'delivered'.

In this query, first the orders, order\_items and order\_payments tables are combined.

With LEFT JOIN, the orders table is joined with the order\_items table based on the order\_id field and the product price is obtained.

With LEFT JOIN, the orders table is joined with the order payments table based on the order\_id field and receives the payment type.

Delivered orders are selected with the condition order\_status='delivered' in the WHERE statement.

In the SELECT statement, the payment\_type field calculates the number of successful orders for each payment type with the expression COUNT(o.order\_id) AS succ\_order\_count.

Calculates the total price of successful orders for each payment type with the expression SUM(price) AS sum\_succ\_price.

The GROUP BY payment\_type statement groups the results by payment type.

With the ORDER BY succ\_order\_count DESC statement, descending order is made according to the number of successful orders.

As a result, this query lists the number of successful orders (succ\_order\_count) and the total price of successful orders (sum\_succ\_price) for each payment type, grouping invoiced orders by payment type.

| **order\_status** | **order\_status\_count** |
| --- | --- |
| delivered | 115038 |
| shipped | 1245 |
| canceled | 745 |
| unavailable | 650 |
| processing | 375 |
| invoiced | 373 |
| created | 5 |
| approved | 3 |

This table sorts order\_status by numbers. 1. Next is delivered.

In the chart, the number of delivered orders is shown as a column chart, while the total amounts of delivered orders are shown as a line chart. Most purchases were made by credit card. At least one purchase was made with a debit card. The payment type with the highest payment volume is credit card with 77%.

**Question 3 :**

- Make a category-based analysis of orders paid in one shot and in installments. In which categories is payment in installments used most?

**SQL QUERY:**

**SELECT product\_category\_name,**

**COUNT(CASE WHEN payment\_installments<1 THEN 'singlepayment' END) AS single\_payment,**

**COUNT(CASE WHEN payment\_installments>0 THEN 'installements' END) AS installements**

**FROM orders o**

**LEFT JOIN order\_items oi ON o.order\_id=oi.order\_id**

**LEFT JOIN order\_payments op ON o.order\_id=op.order\_id**

**LEFT JOIN products p on oi.product\_id=p.product\_id**

**WHERE product\_category\_name IS NOT NULL**

**GROUP BY product\_category\_name**

**ORDER BY taksitli DESC**

**Explanation:**

* First, the orders table is joined to the order\_items, order\_payments and products tables with a LEFT JOIN. This enables the consolidation of relevant data of orders, order items, payment information and products.
* By filtering the rows that do not have a product category with the WHERE product\_category\_name IS NOT NULL statement, only rows with product category information are retrieved.
* Grouping is done by product category with the GROUP BY product\_category\_name statement. This is used to calculate the number of payments for each product category.
* COUNT(CASE WHEN payment\_installments<1 THEN ‘singlepayment’ END) AS single\_payment expression calculates the number of single payment payments. When the payment\_installments column is less than 1, it is considered a single check and counts this case.
* COUNT(CASE WHEN payment\_installments>0 THEN ‘installments’ END) AS installments calculates the number of installment payments. When the payment\_installments column is greater than 0, it is considered installment and counts this status.
* Finally, with the ORDER BY installment DESC expression, the order is made from largest to smallest according to the number of installment payments.

As a result, this query counts payment types by product category, listing the number of single payment ('singlepayment') and installment ('installment') payments for each product category.

The chart shows the top 10 product categories where installment payment is most used. Installment payment is most commonly used in the 'cama\_mesa\_banho' category. As a suggestion, the number of installments can be increased for products belonging to these categories.

**Case 5: RFM Analysis**

* Perform RFM analysis using the data set in the e\_commerce\_data\_.csv file below. When calculating recency, use the last order date as basis, not today's date.

**SQL QUERY:**

**WITH recency AS**

**(WITH li\_date AS**

**(SELECT**

**customer\_id,**

**MAX(invoice\_date\_2) AS last\_invoice\_date**

**FROM e\_commerce\_2**

**WHERE customer\_id IS NOT null**

**GROUP BY customer\_id)**

**SELECT**

**customer\_id,**

**'"2011-12-09"'::DATE -last\_invoice\_date AS recency,**

**CASE**

**WHEN '"2011-12-09"'::DATE - last\_invoice\_date <= 10 THEN 5**

**WHEN '"2011-12-09"'::DATE - last\_invoice\_date <= 30 THEN 4**

**WHEN '"2011-12-09"'::DATE - last\_invoice\_date <= 90 THEN 3**

**WHEN '"2011-12-09"'::DATE - last\_invoice\_date <= 180 THEN 2**

**ELSE 1**

**END AS recency\_score**

**FROM li\_date**

**),**

**frequency AS**

**(SELECT customer\_id,**

**COUNT(DISTINCT invoice\_no) AS frequency,**

**CASE**

**WHEN COUNT(DISTINCT invoice\_no) BETWEEN 0 AND 50 THEN 1**

**WHEN COUNT(DISTINCT invoice\_no) BETWEEN 51 AND 100 THEN 2**

**WHEN COUNT(DISTINCT invoice\_no) BETWEEN 101 AND 150 THEN 3**

**WHEN COUNT(DISTINCT invoice\_no) BETWEEN 151 AND 200 THEN 4**

**WHEN COUNT(DISTINCT invoice\_no) BETWEEN 201 AND 250 THEN 5**

**END AS frequency\_score**

**FROM e\_commerce\_2**

**WHERE customer\_id IS NOT NULL**

**GROUP BY customer\_id**

**),**

**monetory AS**

**(SELECT**

**customer\_id,**

**SUM(unit\_price) AS monetory,**

**CASE**

**WHEN SUM( unit\_price) BETWEEN 0 AND 1000 THEN 1**

**WHEN SUM( unit\_price) BETWEEN 1001 AND 10000 THEN 2**

**WHEN SUM( unit\_price) BETWEEN 10001 AND 20000 THEN 3**

**WHEN SUM( unit\_price) BETWEEN 20001 AND 30000 THEN 4**

**WHEN SUM( unit\_price) BETWEEN 30001 AND 50000 THEN 5**

**END AS monetory\_score**

**FROM e\_commerce\_2**

**WHERE customer\_id IS NOT null**

**GROUP BY customer\_id**

**)**

**SELECT**

**r.customer\_id,**

**recency,**

**recency\_score,**

**frequency,**

**frequency\_score,**

**monetory,**

**monetory\_score,**

**(recency\_score+frequency\_score+monetory\_score )/3 AS rfm\_score**

**FROM recency r**

**JOIN frequency f ON r.customer\_id=f.customer\_id**

**JOIN monetory m ON r.customer\_id=m.customer\_id**

**Explanation:**

This query performs customer segmentation by making the necessary calculations for RFM (Recency, Frequency, Monetary) analysis. It calculates customers' recency, frequency and monetory values ​​and creates segment scores based on these values.

* The Recency value calculates the time from the customer's last purchase date to “2011-12-09”. The closer the most recent transaction within a given period of time, the lower the recency value. Based on this value, a recency score is assigned (between 1-5).
* The Frequency value calculates the number of different invoice numbers of the customer. Customers who shop more frequently receive a higher frequency value. A frequency score is assigned based on the number of invoices in a certain range (between 1-5).
* Monetary value calculates the customer's total purchase amount. Higher coin amounts get a higher monetory score. A monetory score is assigned based on the amount of money in a certain range (between 1-5).

It calculates recency, recency score, frequency, frequency score, monetary, monetory\_score and total RFM score (rfm\_score) for each customer.

**Customers with an RFM score of 1:** Low value (wake-up)

It represents the customer group that is about to be lost.

**Customers with an RFM score of 2:** Potential customer

It represents a customer group with potential.

**Customers with an RFM score of 3:** Loyal customer

It represents a group of loyal customers who shop regularly.

**Customers with an RFM score of 4:** Valuable customer

It represents the group of customers who shop frequently and for high amounts.

**Customers with an RFM score of 5:** Super customer

It represents the most important customer group that makes purchases very frequently and for very high amounts.