1. In the Database Design markdown or pdf, provide the Data Definition Language (DDL) commands you used to create each of these tables in the database. Here's the syntax of the CREATE TABLE DDL command: CREATE TABLE table_name (column1 datatype, column2 datatype, column3 datatype,...);

Table Customers

```
CREATE TABLE customers (
  customer_id VARCHAR PRIMARY KEY,
  customer_zip_code_prefix INT,
  customer_city VARCHAR,
  customer_state VARCHAR
);
```

Table Geolocation

```
CREATE TABLE geolocation (
  geolocation_zip_code_prefix INT,
  geolocation_lat DECIMAL,
  geolocation_lng DECIMAL,
  geolocation_city VARCHAR,
  geolocation_state VARCHAR
);
```

Table order_orders

```
CREATE TABLE orders (
  order_id VARCHAR PRIMARY KEY,
  customer_id VARCHAR,
  order_status VARCHAR,
  order_purchase_timestamp TIMESTAMP,
  order_approved_at TIMESTAMP,
  order_delivered_carrier_date TIMESTAMP,
  order_delivered_customer_date TIMESTAMP,
  order_estimated_delivery_date TIMESTAMP
);
```

Table order_items

```
CREATE TABLE order_items (
  order_id VARCHAR,
  order_item_id INT,
  product_id VARCHAR,
  seller_id VARCHAR,
  shipping_limit_date TIMESTAMP,
  price DECIMAL,
```

```
  freight_value DECIMAL,
  PRIMARY KEY (order_id, order_item_id)
);
```

Table order_payments

```
CREATE TABLE order_payments (
  order_id VARCHAR,
  payment_sequential INT,
  payment_type VARCHAR,
  payment_installments INT,
  payment_value DECIMAL,
  PRIMARY KEY (order_id, payment_sequential)
);
```

Table order_reviews

```
CREATE TABLE order_reviews (
  review_id VARCHAR PRIMARY KEY,
  order_id VARCHAR,
  review_score INT,
  review_comment_title VARCHAR,
  review_comment_message VARCHAR,
  review_creation_date TIMESTAMP,
  review_answer_timestamp TIMESTAMP
);
```

Table Products

```
CREATE TABLE products (
  product_id VARCHAR PRIMARY KEY,
  product_category_name VARCHAR,
  product_name_lenght INT,
  product_description_lenght INT,
  product_photos_qty INT,
  product_weight_g INT,
  product_length_cm INT,
  product_height_cm INT,
  product_width_cm INT
);
```

Table Sellers

```
CREATE TABLE sellers (
  seller_id VARCHAR PRIMARY KEY,
  seller_zip_code_prefix INT,
  seller_city VARCHAR,
  seller_state VARCHAR
);
```

Table product_category_name_translation

```
CREATE TABLE product_category_name_translation (
  product_category_name VARCHAR PRIMARY KEY,
  product_category_name_english VARCHAR
);
```

ROWS COUNT

```
+-----------------------------------+-----------+
| table_name                        | row_count |
+-----------------------------------+-----------+
| olist_customers_dataset           |     99442 |
| olist_geolocation_dataset         |   1000164 |
| olist_order_items_dataset         |    102426 |
| olist_order_payments_dataset      |    112650 |
| olist_order_reviews_dataset       |     99224 |
| olist_orders_dataset              |     99442 |
| olist_products_dataset            |     32329 |
| olist_sellers_dataset             |      3096 |
| product_category_name_translation |        72 |
+-----------------------------------+-----------+
9 rows in set (4.21 sec)
```

ADVANCED Query:

Query 1 – Average Freight and Price by Seller (Join + Group By)

```
SELECT
    s.seller_id,
    s.seller_city,
    COUNT(oi.order_id) AS total_orders,
    AVG(oi.price) AS avg_price,
    AVG(oi.freight_value) AS avg_freight
FROM olist_order_items_dataset oi
JOIN olist_products_dataset p ON oi.product_id = p.product_id
JOIN olist_sellers_dataset s ON p.seller_id = s.seller_id
GROUP BY s.seller_id, s.seller_city;
Limit15;
```

seller_id	seller_city	total_orders	avg_price	avg_freight
0015a82c2db000af6aaaf3ae2ecb0532	santo andre	3	895	21.020000457763672
001cca7ae9ae17fb1caed9dfb1094831	cariacica	240	105.1626248995463	36.99858363866806
001e6ad469a905060d959994f1b41e4f	sao goncalo	1	250	17.940000534057617
002100f778ceb8431b7a1020ff7ab48f	franca	55	22.445454198663885	14.43018214485862
003554e2dce176b5555353e4f3555ac8	goiania	1	120	19.3799991607666
004c9cd9d87a3c30c522c48c4fc07416	ibitinga	170	115.95711768655217	20.889588170893052
00720abe85ba0859807595bbf045a33b	guarulhos	26	38.750000073359566	12.153076777091393
00ab3eff1b5192e5f1a63bcecfee11c8	sao paulo	1	98	12.079999923706055
00d8b143d12632bad99c0ad66ad52825	belo horizonte	1	86	51.099998474121094
00ee68308b45bc5e2660cd833c3f81cc	sao paulo	319	110.4404068471496	17.52097165734043
00fc707aaaad2d31347cf883cd2dfe10	maringa	848	84.6035264483038	15.271910347325623
010543a62bd80aa422851e79a3bc7540	sao paulo	2	708	15.97499942779541
010da0602d7774602cd1b3f5fb7b709e	sao bernardo do campo	5	169.89999389648438	46.04999923706055
011b0eaba87386a2ae96a7d32bb531d1	pompeia	2	49.9900016784668	14.59000015258789
01266d4c46afa519678d16a8b683d325	curitiba	3	30.083333651224773	15.743333498636881

15 rows in set (2.14 sec)

Query 2: Top States by Average Review Score (JOINs, GROUP BY, ORDER BY, and aggregation) WORKING

```
SELECT
    c.customer_state,
    COUNT(r.review_id) AS total_reviews,
    ROUND(AVG(r.review_score), 2) AS avg_review_score
FROM olist_order_reviews_dataset r
JOIN olist_orders_dataset o ON r.order_id = o.order_id
JOIN olist_customers_dataset c ON o.customer_id = c.customer_id
GROUP BY c.customer_state
ORDER BY avg_review_score DESC
LIMIT 15;
```

```
+----------------+---------------+------------------+
| customer_state | total_reviews | avg_review_score |
+----------------+---------------+------------------+
| AP             |            67 |             4.19 |
| AM             |           147 |             4.18 |
| PR             |          5038 |             4.18 |
| SP             |         41689 |             4.17 |
| MG             |         11625 |             4.14 |
| RS             |          5483 |             4.13 |
| MS             |           724 |             4.12 |
| RN             |           482 |             4.11 |
| TO             |           279 |             4.10 |
| MT             |           903 |             4.10 |
| SC             |          3623 |             4.07 |
| DF             |          2148 |             4.06 |
| AC             |            81 |             4.05 |
| RO             |           252 |             4.05 |
| ES             |          2016 |             4.04 |
+----------------+---------------+------------------+
15 rows in set (5.68 sec)
```

Query 3: Sellers with Low Freight Costs (Subquery)

This query finds sellers whose average freight value is greater than $50, using a subquery in the `WHERE` clause:

```
SELECT
    s.seller_id,
    s.seller_city,
    s.seller_state
FROM olist_sellers_dataset s
WHERE s.seller_id IN (
    SELECT p.seller_id
    FROM olist_order_items_dataset oi
    JOIN olist_products_dataset p ON oi.product_id = p.product_id
    GROUP BY p.seller_id
    HAVING AVG(oi.freight_value) < 10
)
LIMIT 15;
```

```
    -> LIMIT 15;
+----------------------------------+-----------------------+--------------+
| seller_id                        | seller_city           | seller_state |
+----------------------------------+-----------------------+--------------+
| seller_id                        | seller_city           | seller_state |
| 3442f8959a84dea7ee197c632cb2df15 | campinas              | SP           |
| e9e446d01bd10a97a8ffcfc4a3a20cb2 | sao paulo             | SP           |
| ec2e006556300a79a5a91e4876ab3a56 | sao paulo             | SP           |
| 05ca864204d09595ae591b93ea9cf93d | barueri               | SP           |
| 0692610d8abe24f287e9fae90ea0bbee | sao paulo             | SP           |
| ea00f977a203ff88adf7057cb7806998 | santos                | SP           |
| ffcfefa19b08742c5d315f2791395ee5 | curitiba              | PR           |
| c1dde11f12d05c478f5de2d7319ad3b2 | sao paulo             | SP           |
| bee36b4f9a2b9fdcaff6ec05df202ed0 | sao paulo             | SP           |
| baf15155e37ef5492731459bdc05be8a | sao paulo             | SP           |
| 71593c7413973a1e160057b80d4958f6 | sao paulo / sao paulo | SP           |
| f12d3c2a14729ae461b920c11fe20fdc | santo andre           | SP           |
| f0ec6a2adb05c62655a26dd347b8dede | sao paulo             | SP           |
| cc1f04647be106ba74e62b21f358af25 | sao paulo             | SP           |
+----------------------------------+-----------------------+--------------+
15 rows in set (1.65 sec)
```

Query 4: Compare Products in High vs Low Freight Brackets (SET OPERATOR: `UNION`)

```
(
    SELECT
        product_id,
        price,
        freight_value,
        'High Freight' AS freight_bracket
```

```
    FROM olist_order_items_dataset
    WHERE freight_value > 100
)
UNION
(
    SELECT
        product_id,
        price,
        freight_value,
        'Low Freight' AS freight_bracket
    FROM olist_order_items_dataset
    WHERE freight_value < 5
)
```

```
+----------------------------------+---------+---------------+-----------------+
| product_id                       | price   | freight_value | freight_bracket |
+----------------------------------+---------+---------------+-----------------+
| 43cc8e4d981bc04b9d78b12e8a908d41 |    1240 |        102.63 | High Freight    |
| a233df9a388d27dbdfd31731d4236db0 | 2649.99 |        134.17 | High Freight    |
| 8d4e92265a16e69a1e1d76e67e46d72f |    1350 |        294.76 | High Freight    |
| 63c4a70e0a12b4bd9475fca9e9937e76 |  122.99 |        164.98 | High Freight    |
| e303dfa61ada1f0823b4775f192606b3 |   148.5 |        165.32 | High Freight    |
| 660422061e06da17ca6101e9d6b7aae8 |     649 |        169.12 | High Freight    |
| ab495f166205a883ffe5ab0b5b55f867 |   629.9 |        117.35 | High Freight    |
| 90c1b4e040d1d1c45897ec2dad4a809d |  839.99 |        174.49 | High Freight    |
| ef854c7d98d5eba672287b0a9d37075b |    1990 |        125.05 | High Freight    |
| 72d0a38fe43ba7087d71e245d1b76c9e |     229 |        106.31 | High Freight    |
| 5640c59a8f6a08b3758272590693eec3 |  238.47 |        119.56 | High Freight    |
| 67a7c7243c0585ccc44471b5a5c115e9 |     322 |        186.38 | High Freight    |
| 65841ad29fc48cd40902e03da7511e05 |     849 |        174.95 | High Freight    |
| 1945afae0c93166dce1f186d00125695 |  517.99 |        107.87 | High Freight    |
| 8a443635fdf9759915c9be5be2e3b862 |    99.9 |        112.44 | High Freight    |
+----------------------------------+---------+---------------+-----------------+
15 rows in set (0.19 sec)
```

Query 5 (New): Products That Were Reviewed the Most with Highest Average Ratings (Multiple Joins, GROUP BY, HAVING with aggregation)

```
SELECT
    p.product_id,
    pct.product_category_name_english AS category,
    COUNT(r.review_id) AS total_reviews,
    ROUND(AVG(r.review_score), 2) AS avg_score
FROM olist_order_reviews_dataset r
JOIN olist_orders_dataset o ON r.order_id = o.order_id
```

JOIN olist_order_items_dataset oi ON o.order_id = oi.order_id
JOIN olist_products_dataset p ON oi.product_id = p.product_id
JOIN product_category_name_translation pct
  ON p.product_category_name = pct.product_category_name
GROUP BY p.product_id, pct.product_category_name_english
HAVING total_reviews > 5 AND avg_score >= 4.5
ORDER BY total_reviews DESC, avg_score DESC

```
*************************** 1. row ***************************
   product_id: 629e019a
 product_name: auto
total_reviews: 73
    avg_score: 4.56
*************************** 2. row ***************************
   product_id: f7a17d2c
 product_name: housewares
total_reviews: 64
    avg_score: 4.55
*************************** 3. row ***************************
   product_id: 5f504b3a
 product_name: cool_stuff
total_reviews: 63
    avg_score: 4.56
*************************** 4. row ***************************
   product_id: 6a8631b7
 product_name: electronics
total_reviews: 62
    avg_score: 4.71
*************************** 5. row ***************************
   product_id: fbc1488c
 product_name: perfumery
total_reviews: 58
    avg_score: 4.62
*************************** 6. row ***************************
   product_id: d696750e
 product_name: housewares
total_reviews: 57
    avg_score: 4.53
*************************** 7. row ***************************
   product_id: 73326828
 product_name: food
total_reviews: 54
    avg_score: 4.83
*************************** 8. row ***************************
   product_id: 130482ad
 product_name: computers_accessories
total_reviews: 52
    avg_score: 4.54
*************************** 9. row ***************************
   product_id: ed2067a9
 product_name: food
total_reviews: 52
    avg_score: 4.54
*************************** 10. row ***************************
   product_id: aa280035
 product_name: toys
total_reviews: 46
    avg_score: 4.63
*************************** 11. row ***************************
   product_id: d5b703c2
 product_name: toys
total_reviews: 44
    avg_score: 4.50
*************************** 12. row ***************************
   product_id: 57e089e3
 product_name: baby
total_reviews: 43
    avg_score: 4.63
*************************** 13. row ***************************
   product_id: f8b624d4
 product_name: housewares
total_reviews: 42
    avg_score: 4.69
*************************** 14. row ***************************
   product_id: aadff884
 product_name: health_beauty
total_reviews: 40
    avg_score: 4.65
*************************** 15. row ***************************
   product_id: bdb4be6c
 product_name: housewares
total_reviews: 40
    avg_score: 4.55
15 rows in set (5.23 sec)
```

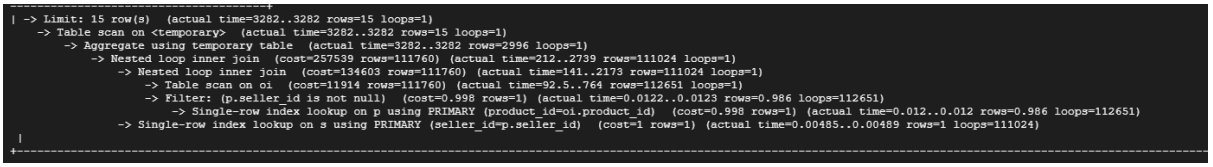
Part 2 Indexing:

QUERY 1

```
| -> Limit: 15 row(s)  (actual time=3424..3424 rows=15 loops=1)
    -> Table scan on <temporary>  (actual time=3424..3424 rows=15 loops=1)
        -> Aggregate using temporary table  (actual time=3424..3424 rows=2996 loops=1)
            -> Nested loop inner join  (cost=222711 rows=111760) (actual time=150..2959 rows=111024 loops=1)
                -> Nested loop inner join  (cost=99775 rows=111760) (actual time=115..2569 rows=111024 loops=1)
                    -> Table scan on oi  (cost=11702 rows=111760) (actual time=68.1..1246 rows=112651 loops=1)
                    -> Filter: (p.seller_id is not null)  (cost=0.688 rows=1) (actual time=0.0115..0.0116 rows=0.986 loops=112651)
                        -> Single-row index lookup on p using PRIMARY (product_id=oi.product_id)  (cost=0.688 rows=1) (actual time=0.0113..0.0113 rows=0.986 loops=112651)
                -> Single-row index lookup on s using PRIMARY (seller_id=p.seller_id)  (cost=1 rows=1) (actual time=0.00328..0.00331 rows=1 loops=111024)
 |
+--------------------------------------------------------------------------------------------------------------------------------
```

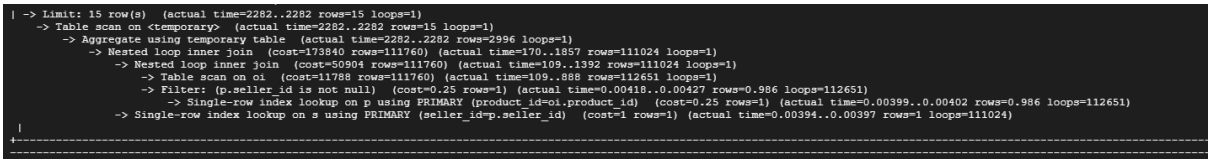
Index Choice:

CREATE INDEX idx_oi_price_freight ON olist_order_items_dataset(price, freight_value);



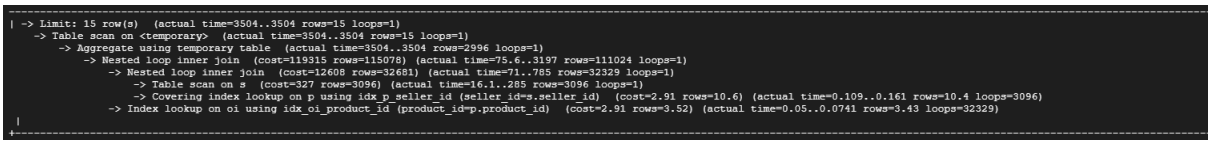
This index is designed to optimize the aggregation operations in the query by allowing the database to quickly access the 'price' and 'freight_value' columns used in the AVG functions. By having these columns indexed together, the system can reduce full table scans and speed up the computation of averages, especially when working with a large dataset.

CREATE INDEX idx_p_seller_id ON olist_products_dataset(seller_id);
JOINED ATTRIBUTE



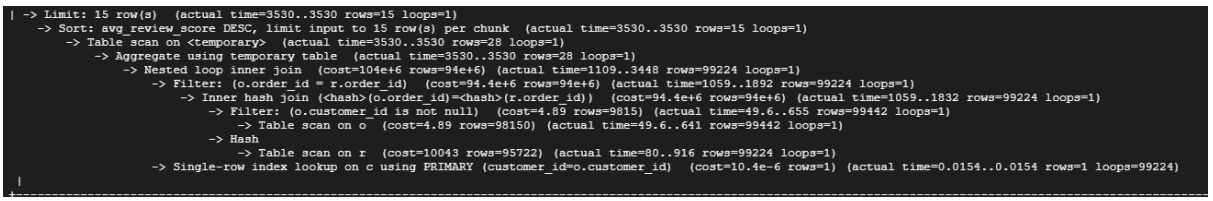
Index on olist_products_dataset(seller_id): This index improves the efficiency of the join between the products and sellers datasets. By indexing the seller_id in the products table, the database engine can rapidly match products to their corresponding sellers, reducing lookup time during the join process and ultimately enhancing the overall query performance.

CREATE INDEX idx_oi_product_id ON olist_order_items_dataset(product_id);



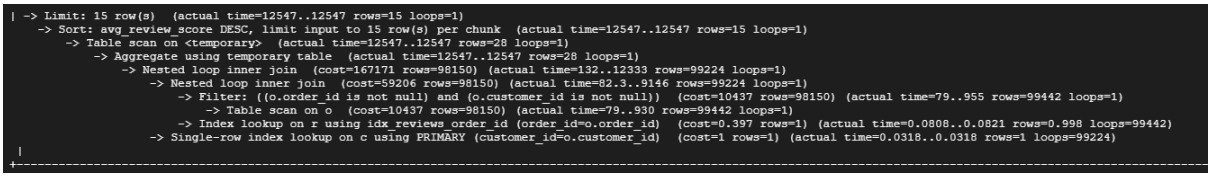
Index on olist_order_items_dataset(product_id): This index is established to speed up the join between order items and products. With the product_id column indexed in the order items dataset, the database can quickly locate matching records for each product, which minimizes the data scanned during the join operation and improves the execution speed of the query.

Query2: Before Indexing Performance:



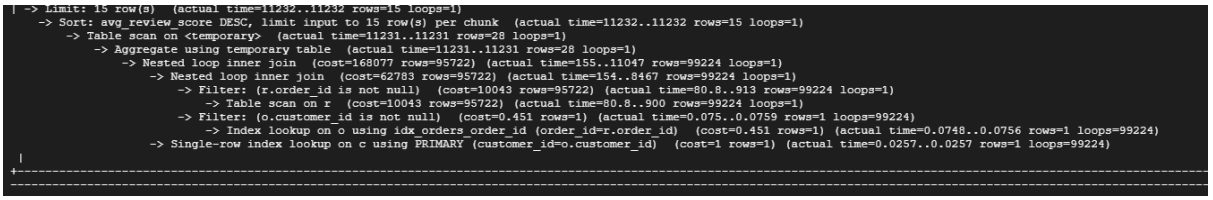
Index Choice:

1. CREATE INDEX idx_reviews_order_id ON olist_order_reviews_dataset(order_id);



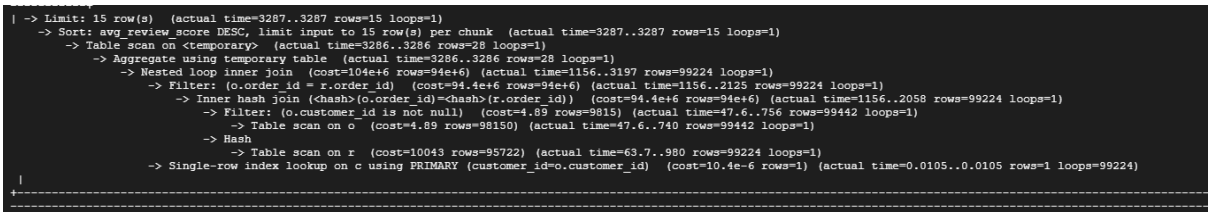
This index is designed to speed up the join between reviews and orders by allowing the database to quickly locate matching order_id values in the reviews table. This minimizes the need for full table scans during the join operation, significantly enhancing performance when working with large datasets.

2. CREATE INDEX idx_orders_order_id ON olist_orders_dataset(order_id);



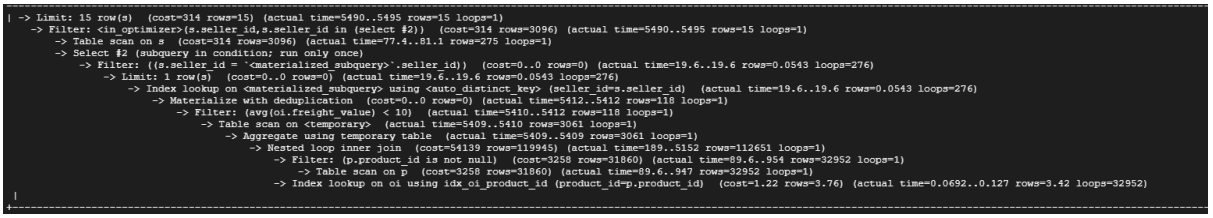
The index on order_id in the orders dataset optimizes the join process by efficiently retrieving corresponding order records for each review. By using this index, the database can quickly match order_id values between the reviews and orders tables, reducing lookup time and improving overall query performance.

3.CREATE INDEX idx_customers_state ON olist_customers_dataset(customer_state);



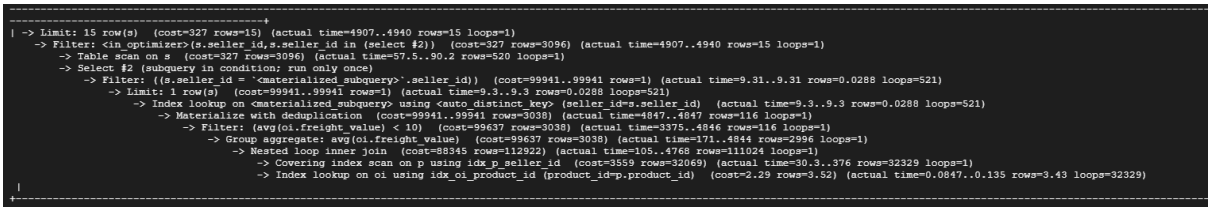
Indexing customer_state in the customers dataset is intended to improve the performance of the grouping and aggregation operations in the query. Since the query groups results by customer_state and computes aggregate functions like count and average, this index helps the database quickly sort and group rows by state, leading to faster execution even if the column has relatively few unique values.

Query 3 – Sellers with Low Freight Costs (Subquery)



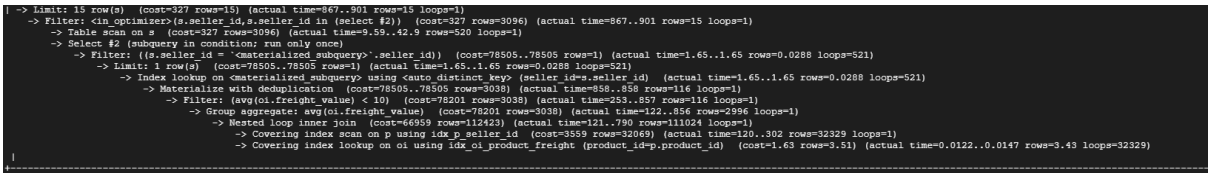
INDEX Choice:

1. CREATE INDEX idx_oi_product_id ON olist_order_items_dataset(product_id);



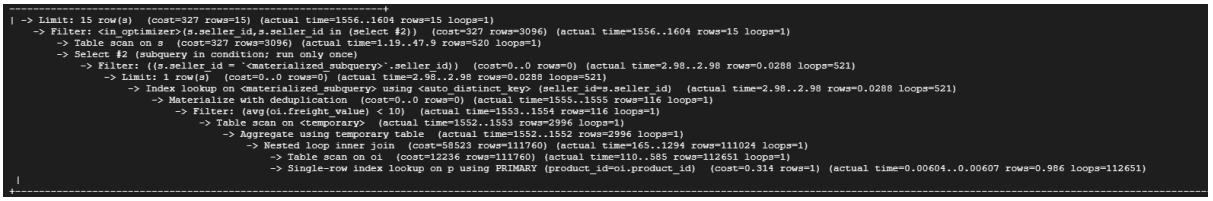
1. olist_order_items_dataset(product_id, freight_value)

CREATE INDEX idx_oi_product_freight ON olist_order_items_dataset(product_id, freight_value);



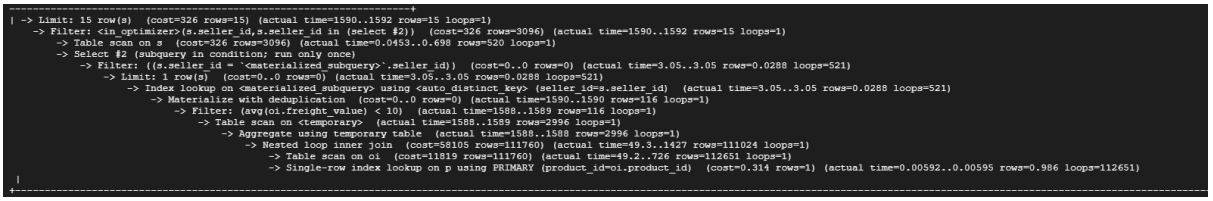
This composite index helps the subquery perform both the join on product_id and the aggregation on freight_value efficiently. It allows the database to quickly find records based on product_id while also retrieving freight_value for computing averages, reducing unnecessary scans. Optimizes join to products + improves AVG computation in HAVING.

2. olist_products_dataset(product_id, seller_id)



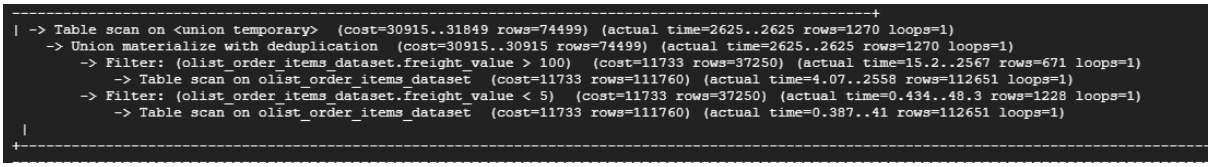
This index speeds up the join between order items and products by enabling fast lookup of seller_id using product_id. It makes grouping by seller_id in the subquery more efficient, which is crucial for filtering based on the average freight value. Helps with the join and allows efficient grouping by seller_id.

3. olist_sellers_dataset(seller_id)



This index ensures that the outer query can rapidly match seller records against the list of seller_ids produced by the subquery. It speeds up the filtering process in the main query by quickly locating the relevant sellers based on their seller_id. Helps with filtering in the outer query's WHERE ... IN (...).

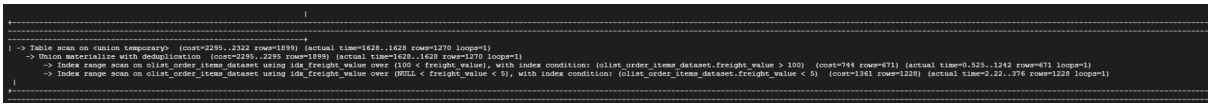
QUERY 4:



1. Single column index on freight_value

Best for optimizing the WHERE condition.

olist_order_items_dataset(freight_value);



This single-column index is designed to accelerate the filtering process in both parts of the UNION query, where freight_value is compared against 100 and 5. By indexing freight_value, the database engine can rapidly locate rows that meet the high or low freight conditions, thus minimizing full table scans. However, since the query also retrieves product_id and price, additional lookups may be necessary once the matching rows are identified.

2. Composite index on (freight_value, price)

Improves performance if you're analyzing price ranges or doing additional filtering/sorting by price.

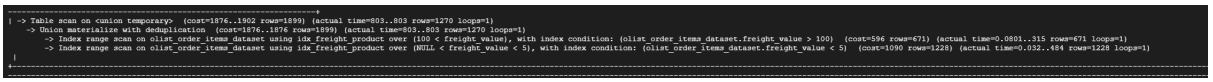
olist_order_items_dataset(freight_value, price);

This composite index enhances performance by starting with the freight_value column, which is used for filtering, and then including the price column, which is part of the SELECT list. The combination allows the engine to not only quickly identify rows with the desired freight_value criteria but also to more efficiently access the price data directly from the index. While this can reduce the number of additional lookups compared to a single-column index, the product_id column is not included, so the engine may still need to fetch that value separately.

3. Composite index on (freight_value, product_id)

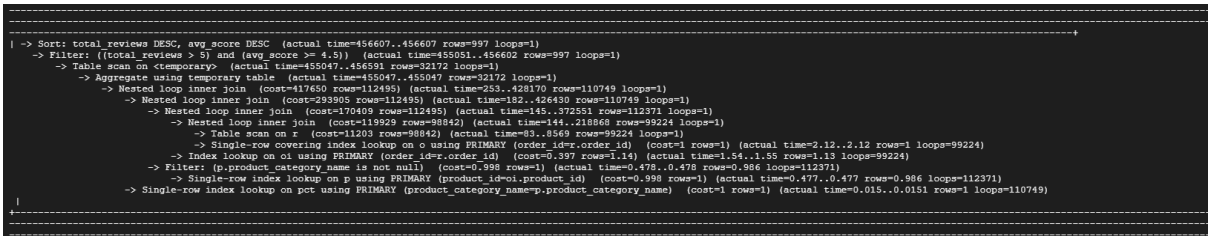
Useful if you later expand this query to group by or join on product_id.

olist_order_items_dataset(freight_value, product_id);



This composite index is optimized by pairing freight_value with product_id. It efficiently supports the WHERE clause by quickly filtering based on freight_value, while simultaneously speeding up access to product_id, which is crucial for the query's output. Although this index improves lookup performance for the filtering and one of the selected columns, the price column is omitted from the index, potentially necessitating extra lookups to retrieve its value.

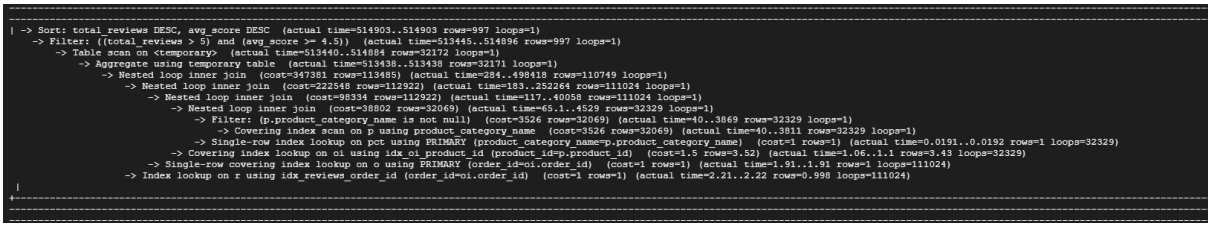
QUERY5



1. Index for joining reviews to orders:

olist_order_reviews_dataset(order_id);

Speeds up JOIN r ON r.order_id = o.order_id



This index is created to speed up the join between the reviews and orders datasets by allowing the database to quickly locate matching order_id values. Since the query starts by joining reviews to orders on order_id, this index reduces the amount of data scanned and accelerates the first step of the join chain.

2. Index for joining order items with products:

olist_order_items_dataset(product_id);

Optimizes JOIN oi ON oi.product_id = p.product_id

This index improves the join performance between order items and products by making product_id lookups more efficient. It allows the database to quickly find all order items for a given product, which is important for aggregating review scores and counts by product.

3. Index for joining products with category translation:

olist_products_dataset(product_category_name);

Speeds up JOIN pct ON p.product_category_name = pct.product_category_name

This index supports the final join in the query, where the products dataset is joined with the category translation table using product_category_name. By indexing this column, the database can more efficiently match each product with its English category name, streamlining the grouping and output formatting process.

```
EXPLAIN ANALYZE
SELECT
   p.product_id,
   pct.product_category_name_english AS category,
   COUNT(r.review_id) AS total_reviews,
   ROUND(AVG(r.review_score), 2) AS avg_score
FROM olist_order_reviews_dataset r
JOIN olist_orders_dataset o ON r.order_id = o.order_id
JOIN olist_order_items_dataset oi ON o.order_id = oi.order_id
JOIN olist_products_dataset p ON oi.product_id = p.product_id
JOIN product_category_name_translation pct
  ON p.product_category_name = pct.product_category_name
GROUP BY p.product_id, pct.product_category_name_english
HAVING total_reviews > 5 AND avg_score >= 4.5
ORDER BY total_reviews DESC, avg_score DESC;
```