Conducted based-analysis SQL to identify and address selection gaps on Amazon

This Project is performed using intermediate-level SQL queries to find the selection gaps using Amazon’s dataset.

This includes the examination of product performance metrics through SQL-based analysis to uncover insights. These insights enable the identification of selection gaps within product offerings, highlighting areas where customer demand may not be adequately met or where product performance falls short of expectations.

DATA -The data source for this project is <https://www.kaggle.com/datasets>. The data is cleaned using EXCEL.

TOOL USED- POSTGRE SQL /PgADMIN4

**Overview-**

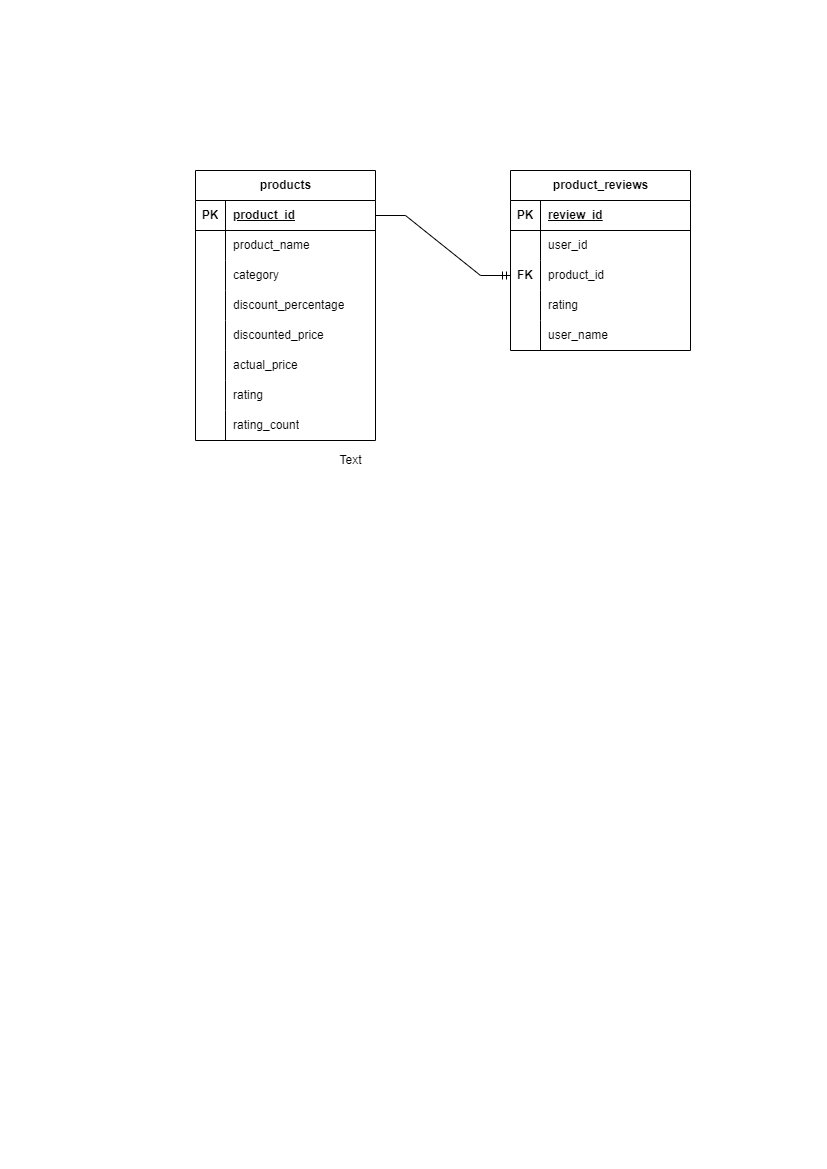
By delving into various aspects of product performance such as ratings, reviews, sales volume, revenue, and category distribution, we can gain insights into areas where improvements or adjustments may be needed. By addressing selection gaps, Amazon can enhance its product offerings This helps optimize the product portfolio by removing underperforming products or enhancing their quality.

**Problem statement** - To Identify and Address Selection Gaps through product performance analysis.

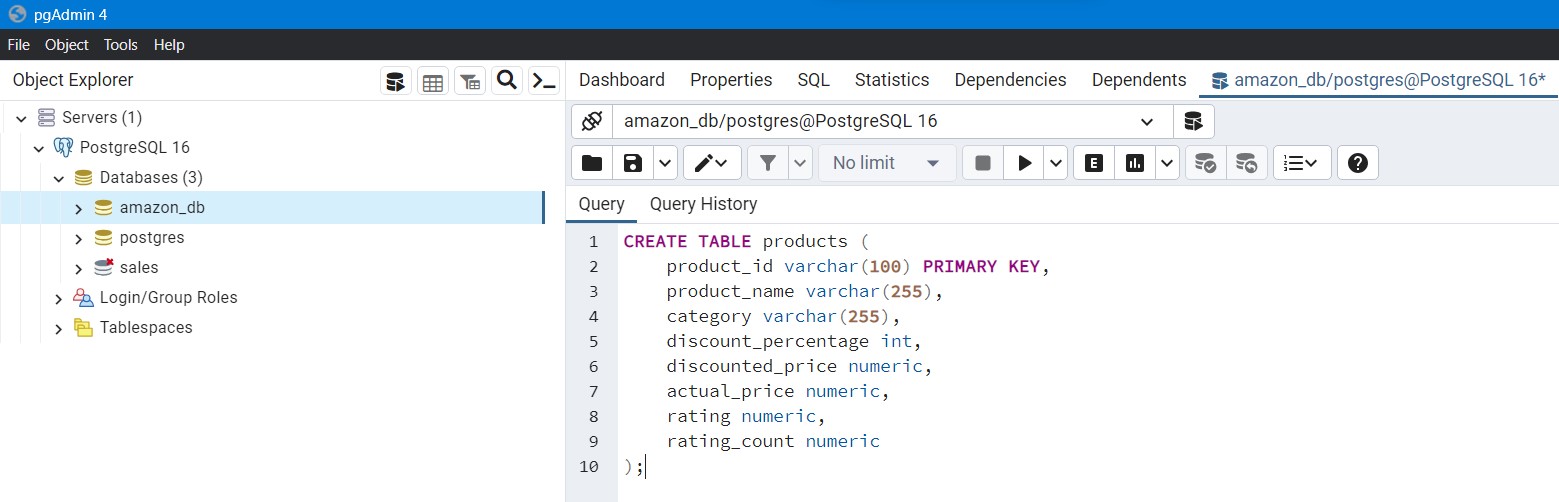
**Learnings –**Understanding the area of improvements within the product offering to optimize the product portfolio.

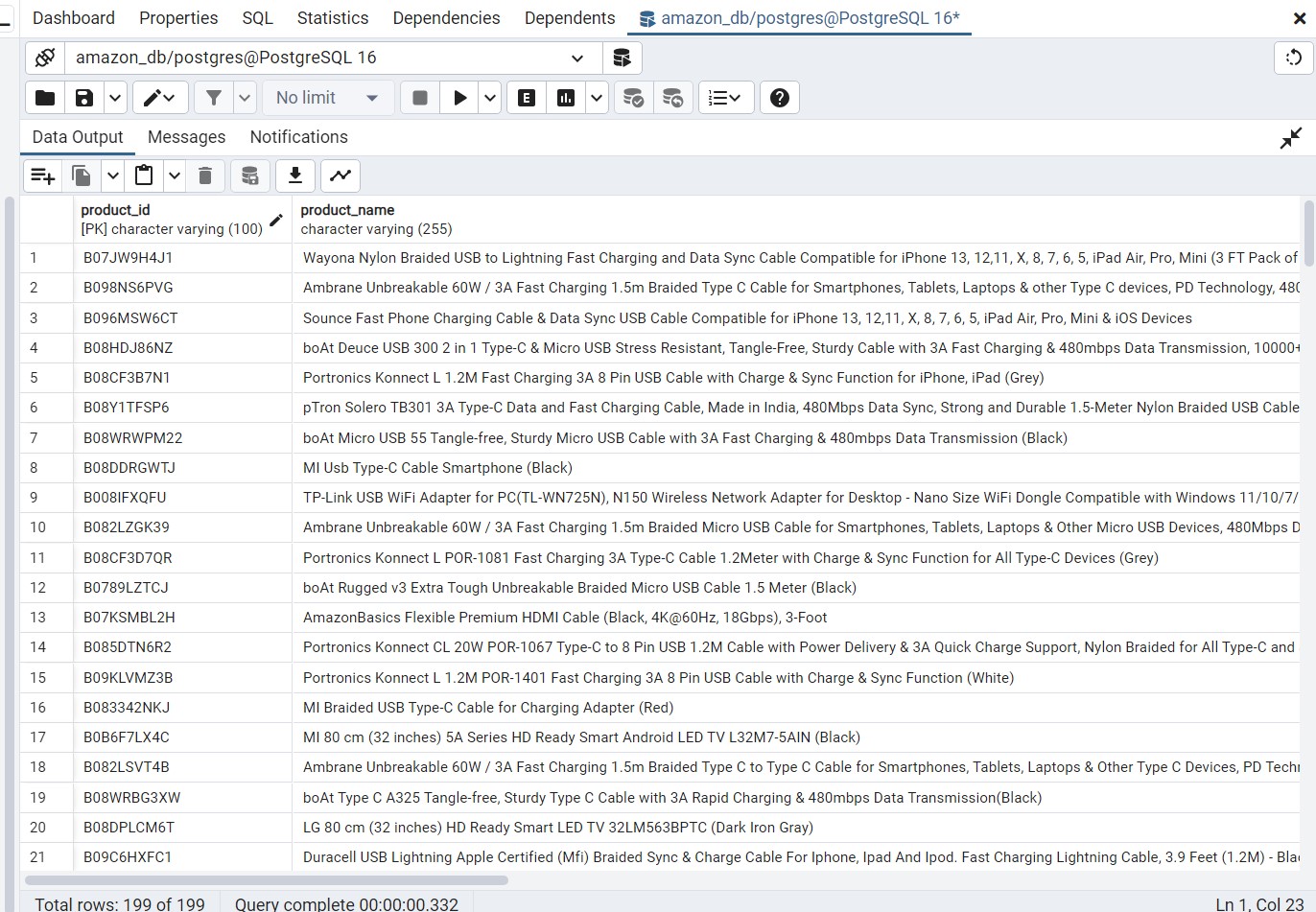
The detailed explanation of each query along with its purpose and learnings is explained at the end of each query

SCHEMA

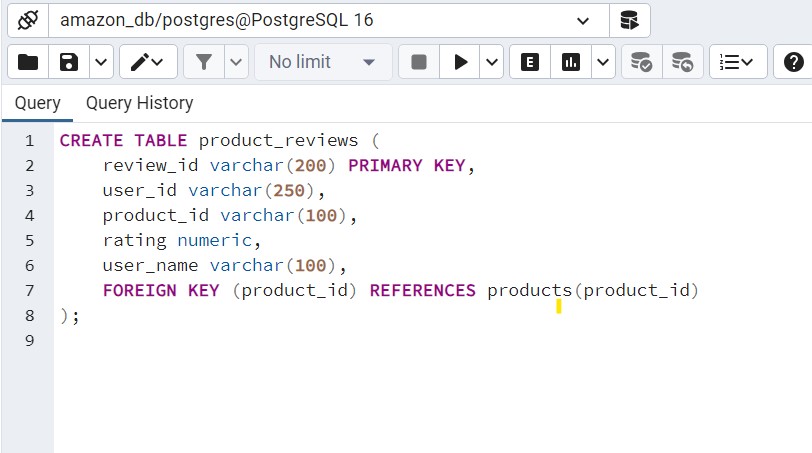


CREATING A TABLE products IN DATABASE amazon\_db

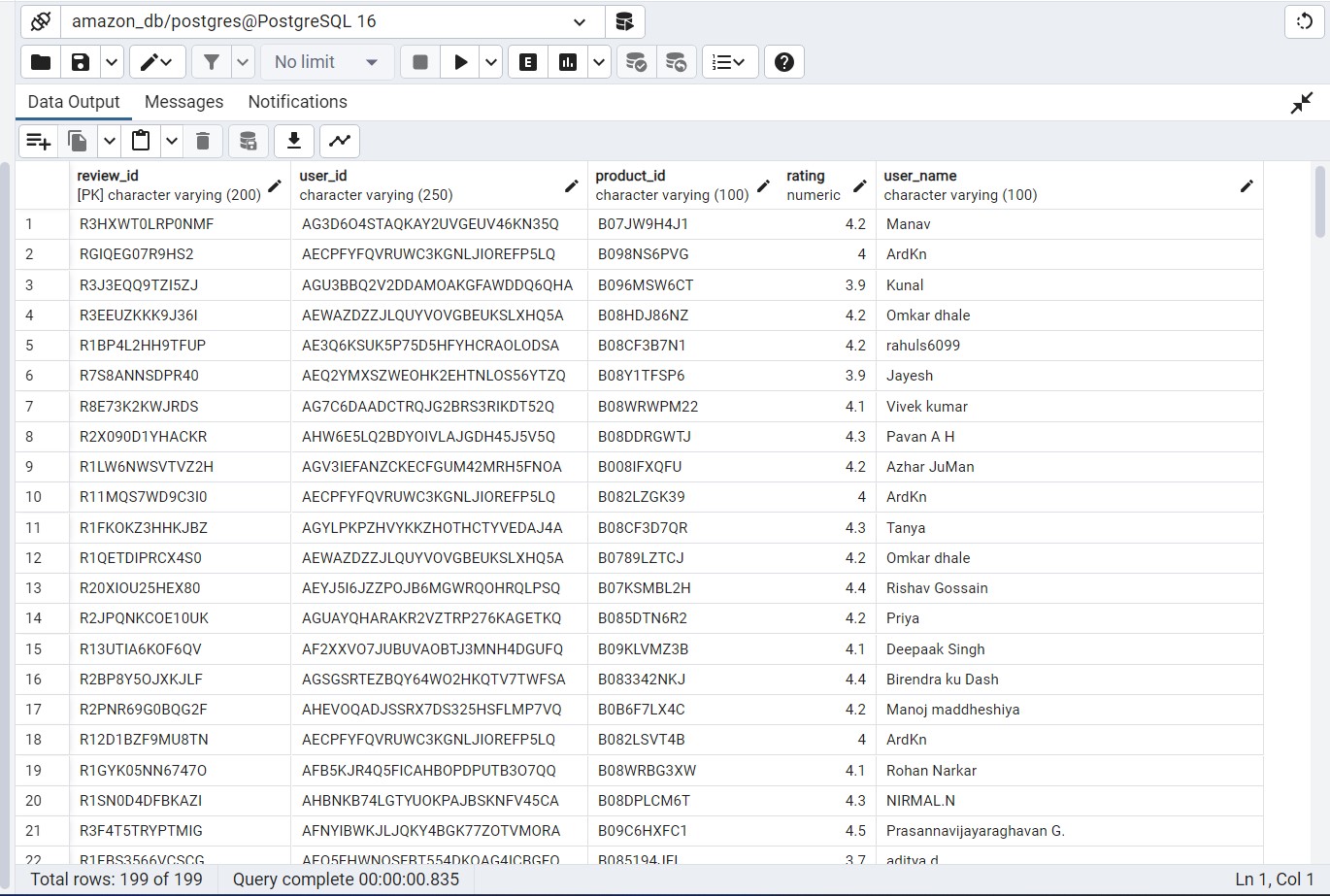


The result after the data is imported in table amazon\_db 

CREATING A TABLE products\_reviews in database amazon\_db



The result after the data is imported in TABLE products\_reviews



These tables are created to store information about products and their reviews in the database amazon\_db. Here's an explanation of each table and how they help in writing the queries:

Products Table:

product\_id: Unique identifier for each product.

product\_name: Name of the product.

category: Category to which the product belongs.

discount\_percentage: Percentage of discount offered on the product.

discounted\_price: Price of the product after discount.

actual\_price: Original price of the product.

rating: Average rating of the product.

rating\_count: Number of ratings the product has received.

Purpose: This table stores detailed information about each product, including its name, category, pricing, and ratings. It serves as the foundation for analyzing various aspects of product performance such as ratings, revenue, discounts, and category distribution.

Product Reviews Table:

review\_id: Unique identifier for each review.

user\_id: Identifier for the user who provided the review.

product\_id: Identifier for the product being reviewed (foreign key referencing product\_id in the Products table).

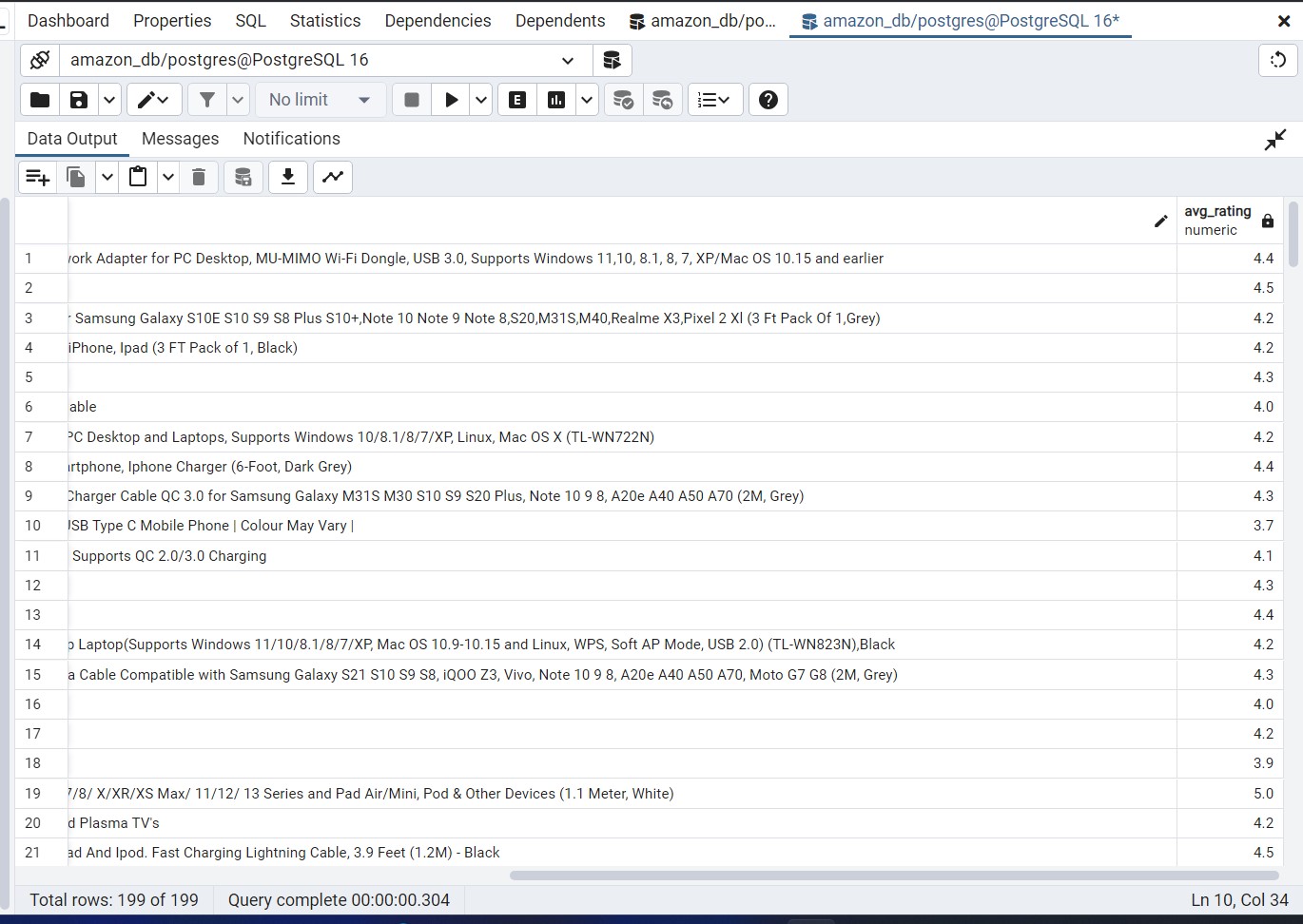
rating: Rating provided by the user for the product.

user\_name: Name of the user who provided the review.

Purpose: This table stores user reviews for products, including the rating given by each user. It allows us to analyze customer feedback based on reviews and sentiment towards products, which is crucial for understanding customer satisfaction and identifying areas for improvement.

 1. QUERY FOR AVERAGE RATING OF EACH PRODUCT

Results for average rating for each product



SELECT p.product\_id, p.product\_name, ROUND(AVG(pr.rating), 1) AS avg\_rating: This part of the query selects three columns:

p.product\_id: The unique identifier for each product.

p.product\_name: The name of each product.

ROUND(AVG(pr.rating), 1) AS avg\_rating: It calculates the average rating for each product by using the AVG function to find the average of the rating column in the product\_reviews table, rounded to one decimal place. The result is aliased as avg\_rating.

FROM products p: This specifies the table products as the source for retrieving data about products. It aliases this table as p for brevity.

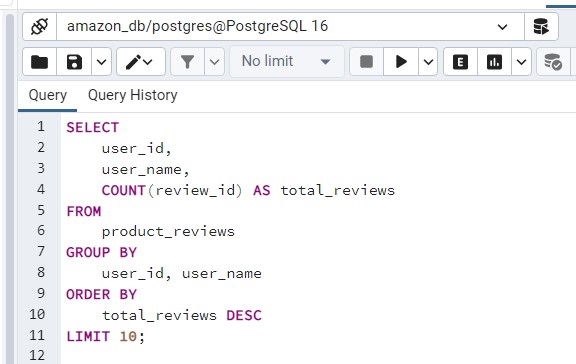
LEFT JOIN product\_reviews pr ON p.product\_id = pr.product\_id: This performs a left join between the products table (p) and the product\_reviews table (pr) based on the product\_id column. It ensures that all rows from the products table are included in the result set, regardless of whether there is a matching row in the product\_reviews table. This join allows the retrieval of product reviews for each product.

GROUP BY p.product\_id, p.product\_name: This groups the result set by product\_id and product\_name. It ensures that the aggregation function (AVG) is applied to the ratings of each product separately.

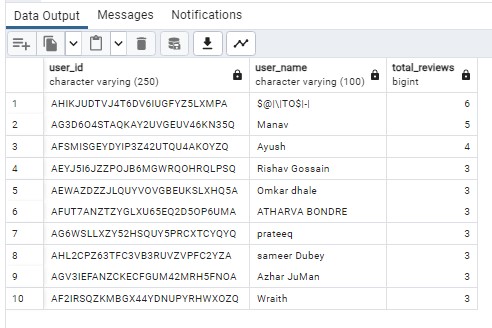
Problem: Understanding the average rating of each product helps in identifying which products are performing well in terms of customer satisfaction.

Solution: This query calculates the average rating for each product, allowing us to identify products with consistently high or low ratings.

2. TOP 10 USERS WHO HAVE PROVIDED THE MOST REVIEWS



Results for 10 top users with most reviews



SELECT user\_id, user\_name, COUNT(review\_id) AS total\_reviews: This part of the query selects three columns:

user\_id: The unique identifier for each user who wrote a review.

user\_name: The name of each user who wrote a review.

COUNT(review\_id) AS total\_reviews: It counts the number of reviews written by each user and aliases the count as total\_reviews.

FROM product\_reviews: This specifies the product\_reviews table as the source for retrieving data about product reviews.

GROUP BY user\_id, user\_name: This groups the result set by user\_id and user\_name. It ensures that each user's reviews are grouped together.

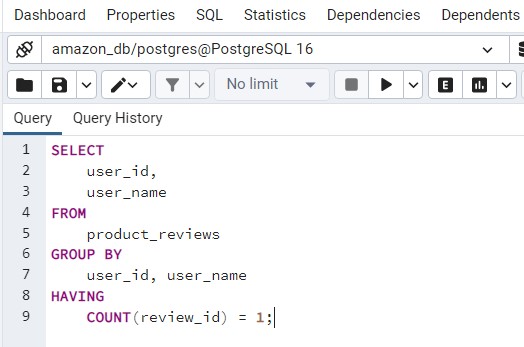
ORDER BY total\_reviews DESC: This orders the result set by the total number of reviews (total\_reviews) in descending order, meaning the users with the highest number of reviews will appear first.

LIMIT 10: This limits the result set to only the top 10 users with the highest number of reviews.

Problem: Identifying the most active users can provide insights into which products are being reviewed frequently.

Solution: This query retrieves the top 10 users who have provided the most reviews, helping us understand customer engagement levels

3. USERS WITH ONLY ONE REVIEW:



Results users with one review



SELECT user\_id, user\_name: This part of the query selects two columns:

user\_id: The unique identifier for each user who wrote a review.

user\_name: The name of each user who wrote a review.

FROM product\_reviews: This specifies the product\_reviews table as the source for retrieving data about product reviews.

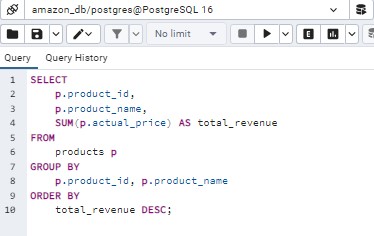
GROUP BY user\_id, user\_name: This groups the result set by user\_id and user\_name. It ensures that each user's reviews are grouped together.

HAVING COUNT(review\_id) = 1: This clause filters the grouped results to only include users who have written exactly one review. The COUNT(review\_id) function counts the number of reviews for each user, and HAVING is used to apply the condition to the grouped results.

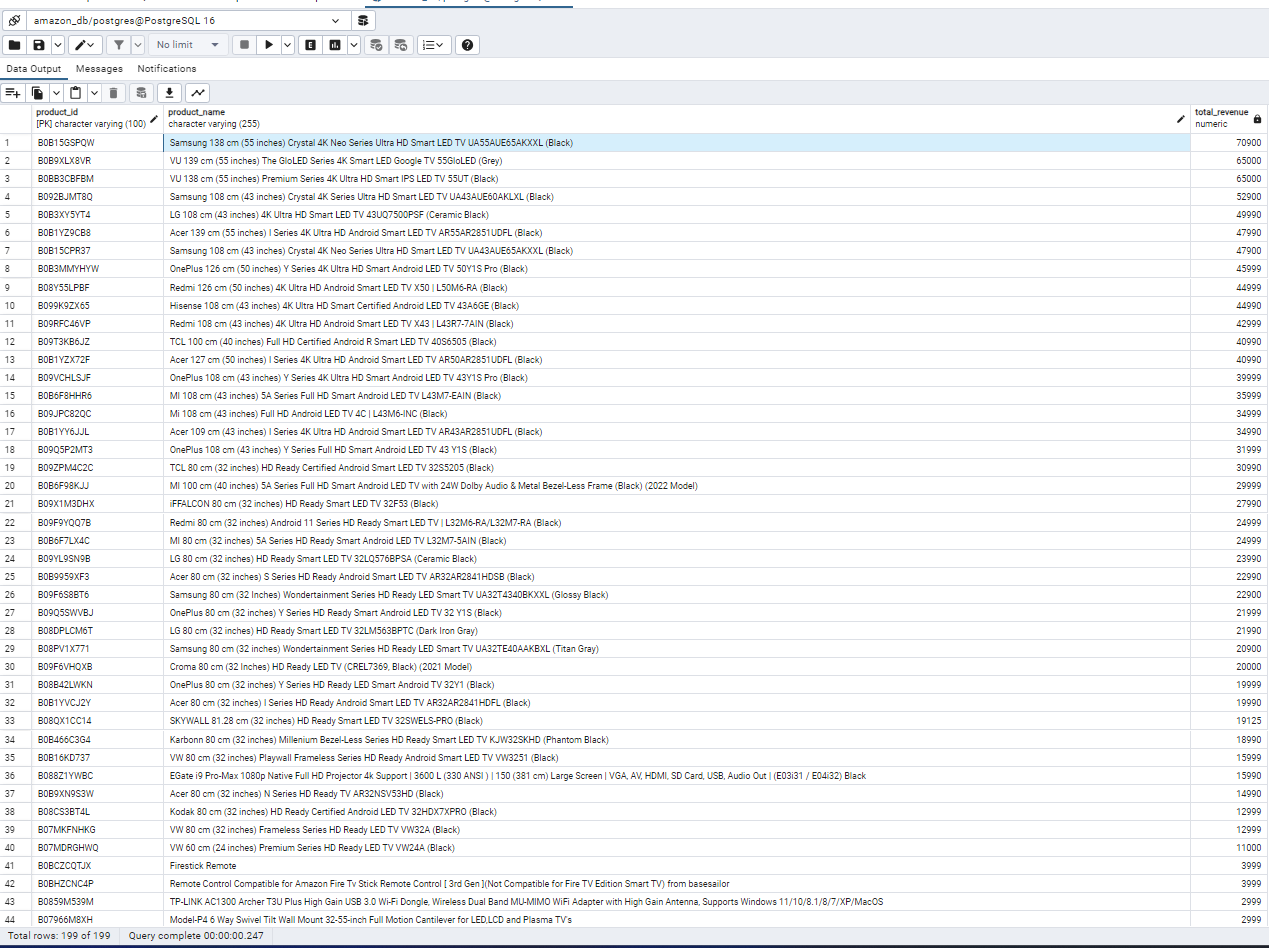
Problem: Users who have provided only one review might indicate products that are not engaging enough for repeat purchases.

Solution: This query identifies users who have provided only one review, highlighting potential areas where product selection could be improved to encourage repeat purchases.

4.TOTAL REVENUE GENERATED BY EACH PRODUCT (BASED ON ACTUAL PRICE)



Total revenue generated by each product (based on actual price)



SELECT p.product\_id, p.product\_name, SUM(p.actual\_price) AS total\_revenue: This part of the query selects three columns:

p.product\_id: The unique identifier for each product.

p.product\_name: The name of each product.

SUM(p.actual\_price) AS total\_revenue: It calculates the total revenue generated by each product by summing up the values in the actual\_price column for that product. The result is aliased as total\_revenue.

FROM products p: This specifies the products table as the source for retrieving data about products. It aliases this table as p for brevity.

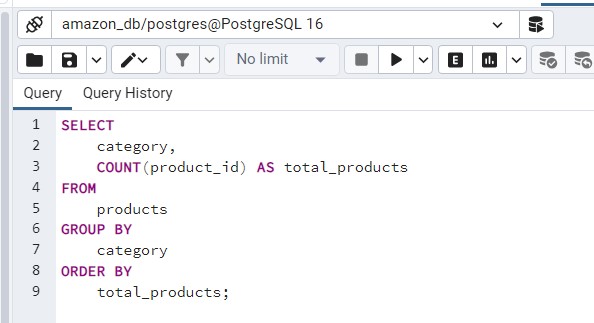
GROUP BY p.product\_id, p.product\_name: This groups the result set by product\_id and product\_name. It ensures that each product's revenue is calculated separately.

ORDER BY total\_revenue DESC: This orders the result set by the total revenue (total\_revenue) in descending order, meaning the products with the highest total revenue will appear first.

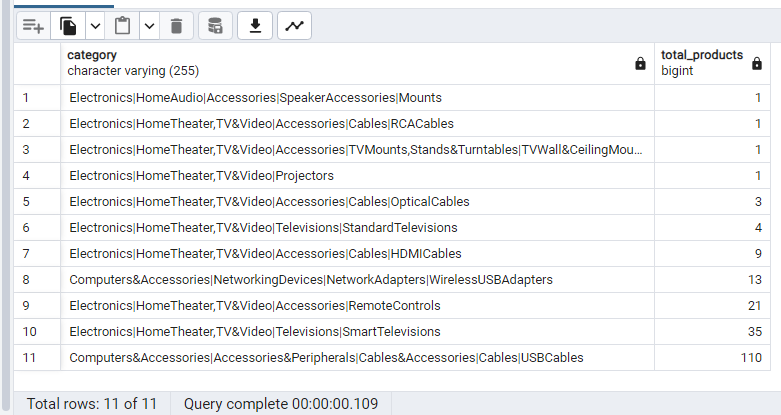
Problem: Analyzing revenue generated by each product helps in understanding their overall profitability.

Solution: This query calculates the total revenue generated by each product based on its actual price, allowing us to prioritize products that contribute the most to revenue.

5.QUERY TO IDENTIFY CATEGORIES WITH FEWER PRODUCTS:



Results of categories with fewer products



SELECT category, COUNT(product\_id) AS total\_products: This part of the query selects two columns:

category: The category of the products.

COUNT(product\_id) AS total\_products: It counts the number of products within each category and aliases the count as total\_products.

FROM products: This specifies the products table as the source for retrieving data about products.

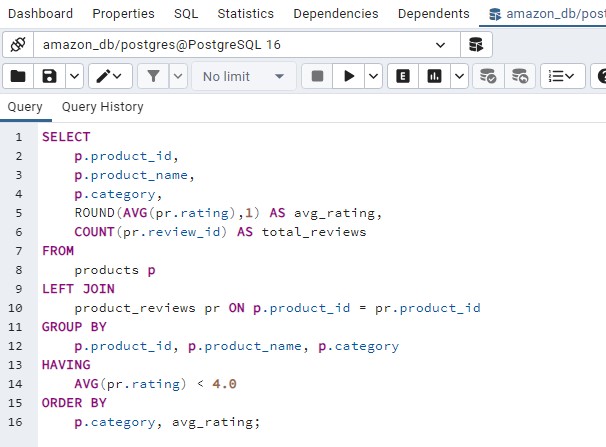
GROUP BY category: This groups the result set by the category column. It ensures that the count of products is calculated for each unique category.

ORDER BY total\_products;: This orders the result set by the total number of products (total\_products) in ascending order. Categories with fewer products will appear first.

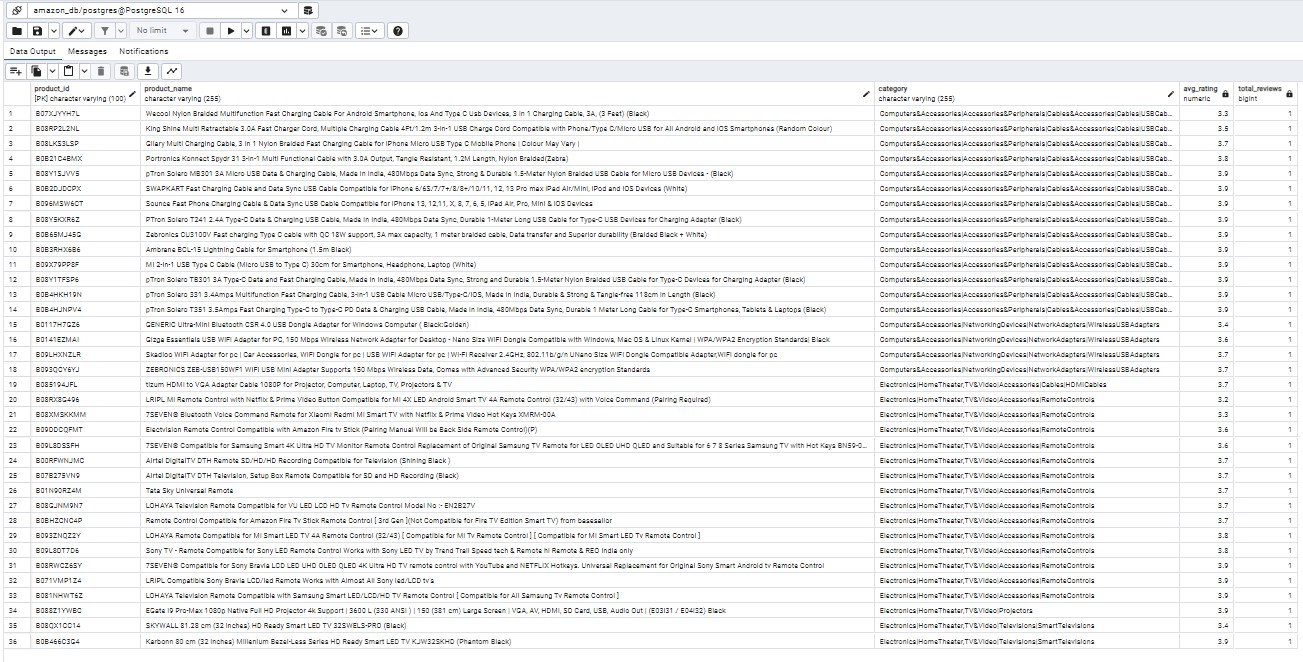
Problem: Categories with fewer products might indicate areas where product selection is lacking.

Solution: This query identifies categories with fewer products, helping us focus on expanding product offerings in those category

6. LOWEST RATED PRODUCTS WITHIN EACH CATEGORY



Lowest rated products within each category



SELECT p.product\_id, p.product\_name, p.category, AVG(pr.rating) AS avg\_rating, COUNT(pr.review\_id) AS total\_reviews: This part of the query selects five columns:

p.product\_id: The unique identifier for each product.

p.product\_name: The name of each product.

p.category: The category of each product.

AVG(pr.rating) AS avg\_rating: It calculates the average rating for each product by utilizing the AVG function to find the average of the rating column in the product\_reviews table. The result is aliased as avg\_rating.

COUNT(pr.review\_id) AS total\_reviews: It counts the total number of reviews for each product. This count includes even products with no reviews, as the join is a left join.

FROM products p LEFT JOIN product\_reviews pr ON p.product\_id = pr.product\_id: This performs a left join between the products table (p) and the product\_reviews table (pr) based on the product\_id column. It ensures that all products are included in the result set, even if they have no reviews.

GROUP BY p.product\_id, p.product\_name, p.category: This groups the result set by product\_id, product\_name, and category. It ensures that the calculations (average rating and total reviews) are performed for each unique combination of product ID, name, and category.

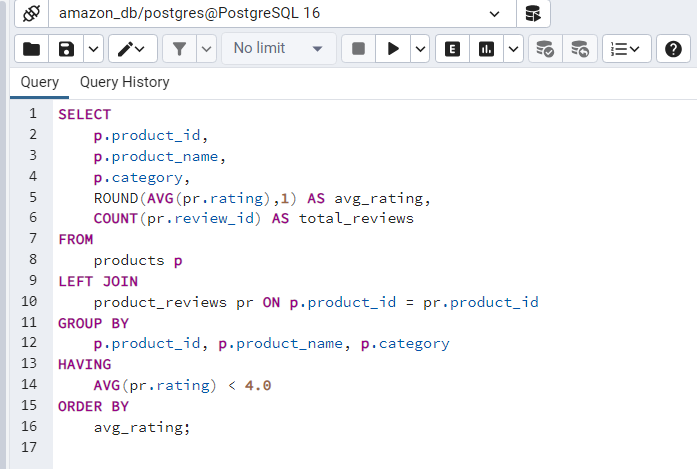
HAVING AVG(pr.rating) < 4.0: This filters the grouped results to only include products with an average rating less than 4.0. The HAVING clause is used here because it filters the results based on aggregated values (in this case, the average rating), rather than individual row values.

ORDER BY p.category, avg\_rating: This orders the result set first by category (p.category) in ascending order and then by average rating (avg\_rating) within each category in ascending order. This arrangement ensures that the lowest-rated products within each category appear first.

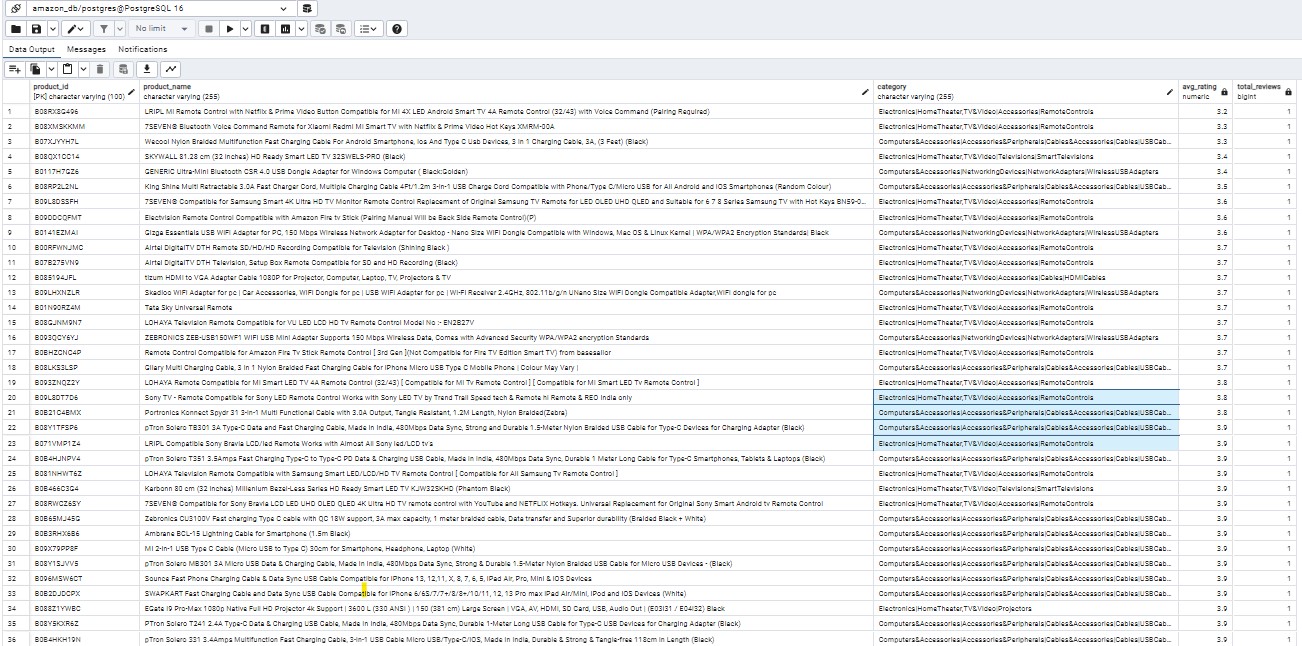
Problem: Identifying the lowest rated products within each category helps in pinpointing areas where product selection or quality may be lacking.

Solution: This query retrieves the lowest rated products within each category, enabling us to address quality issues or gaps in product selection.

7. LOWEST RATED PRODUCTS OVERALL



The result of the low-rated products overall



SELECT p.product\_id, p.product\_name, p.category, AVG(pr.rating) AS avg\_rating, COUNT(pr.review\_id) AS total\_reviews: This part of the query selects five columns:

p.product\_id: The unique identifier for each product.

p.product\_name: The name of each product.

p.category: The category of each product.

AVG(pr.rating) AS avg\_rating: It calculates the average rating for each product by utilizing the AVG function to find the average of the rating column in the product\_reviews table. The result is aliased as avg\_rating.

COUNT(pr.review\_id) AS total\_reviews: It counts the total number of reviews for each product. This count includes even products with no reviews, as the join is a left join.

FROM products p LEFT JOIN product\_reviews pr ON p.product\_id = pr.product\_id: This performs a left join between the products table (p) and the product\_reviews table (pr) based on the product\_id column. It ensures that all products are included in the result set, even if they have no reviews.

GROUP BY p.product\_id, p.product\_name, p.category: This groups the result set by product\_id, product\_name, and category. It ensures that the calculations (average rating and total reviews) are performed for each unique combination of product ID, name, and category.

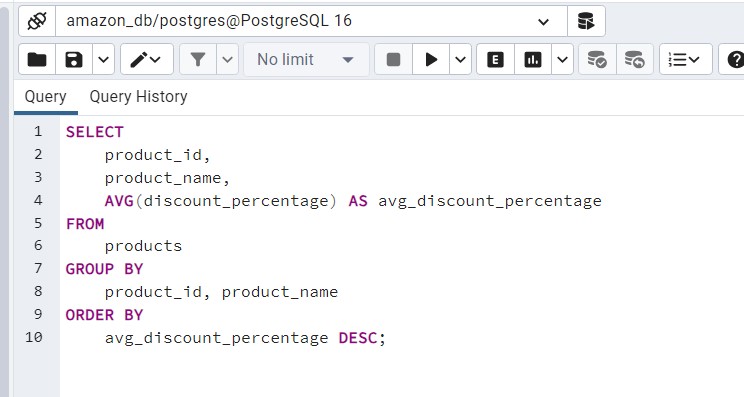
HAVING AVG(pr.rating) < 4: This filters the grouped results to only include products with an average rating less than 4. The HAVING clause is used here because it filters the results based on aggregated values (in this case, the average rating), rather than individual row values.

ORDER BY avg\_rating: This orders the result set by the average rating (avg\_rating) in ascending order. This arrangement ensures that the lowest-rated products appear first.

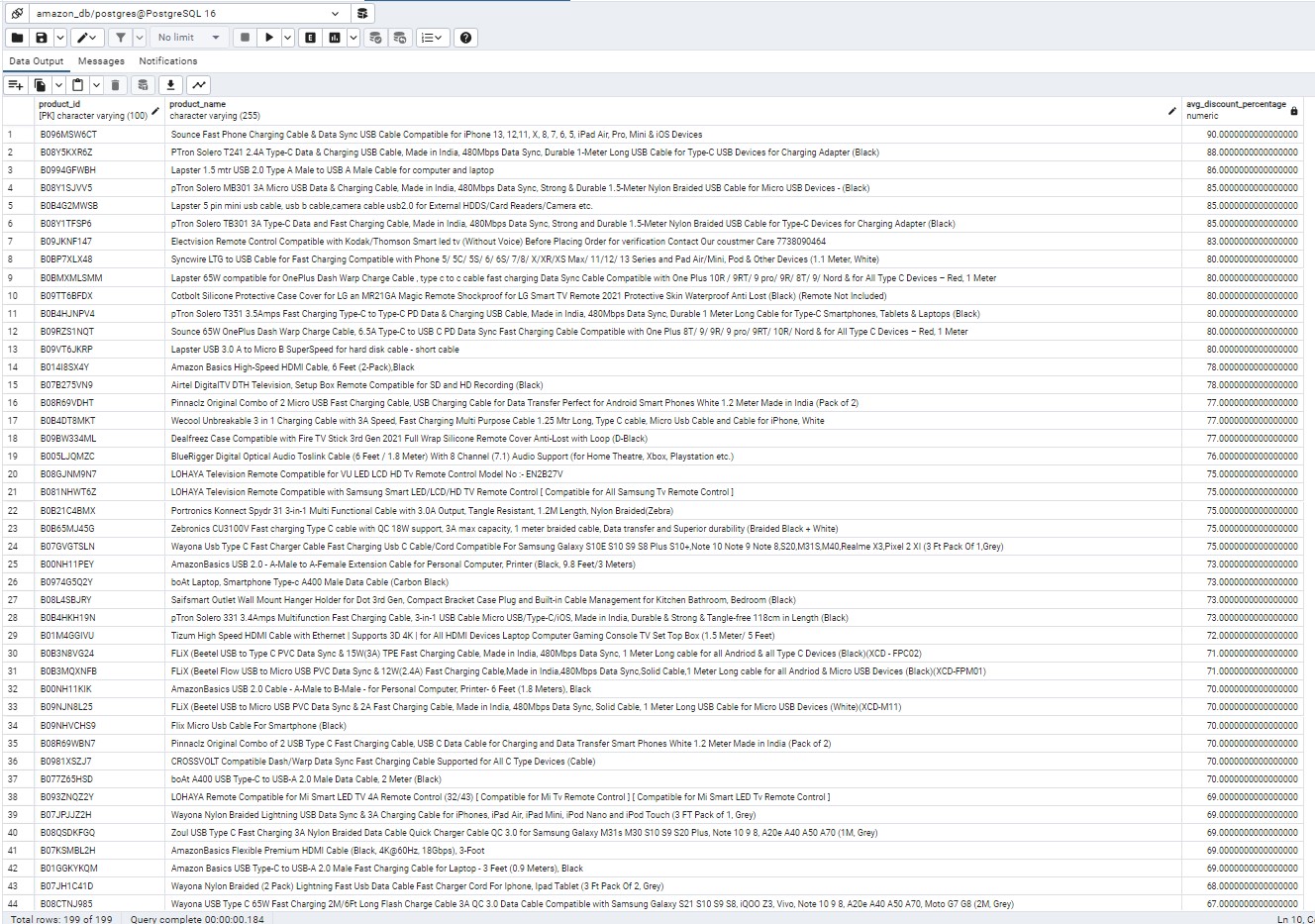
Problem: Products with consistently low ratings across all categories may need immediate attention.

Solution: This query identifies the lowest rated products overall, allowing us to prioritize improvements or consider removing them from our offerings.

8. PRODUCTS WITH THE HIGHEST AVERAGE DISCOUNT PERCENTAGE:



Result of Products with the highest average discount percentage



SELECT product\_id, product\_name, AVG(discount\_percentage) AS avg\_discount\_percentage:

This part of the query selects three columns:

product\_id: The unique identifier for each product.

product\_name: The name of each product.

AVG(discount\_percentage) AS avg\_discount\_percentage: It calculates the average discount percentage for each product by utilizing the AVG function to find the average of the discount\_percentage column in the products table. The result is aliased as avg\_discount\_percentage.

FROM products: This specifies the products table as the source for retrieving data about products.

GROUP BY product\_id, product\_name: This groups the result set by product\_id and product\_name. It ensures that the calculations (average discount percentage) are performed for each unique combination of product ID and name.

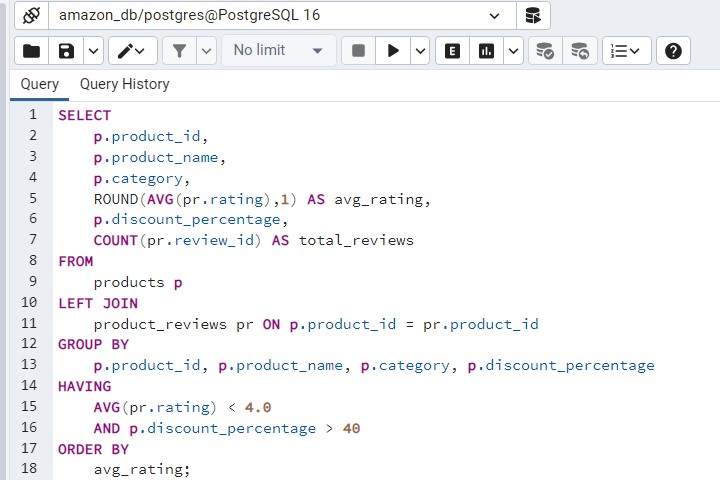
ORDER BY avg\_discount\_percentage DESC: This orders the result set by the average discount percentage (avg\_discount\_percentage) in descending order. This arrangement ensures that the products with the highest average discount percentages appear first.

Overall, the query calculates the average discount percentage for each product and presents the products with the highest average discount percentages first. This information is useful for identifying products that typically offer the highest discounts.

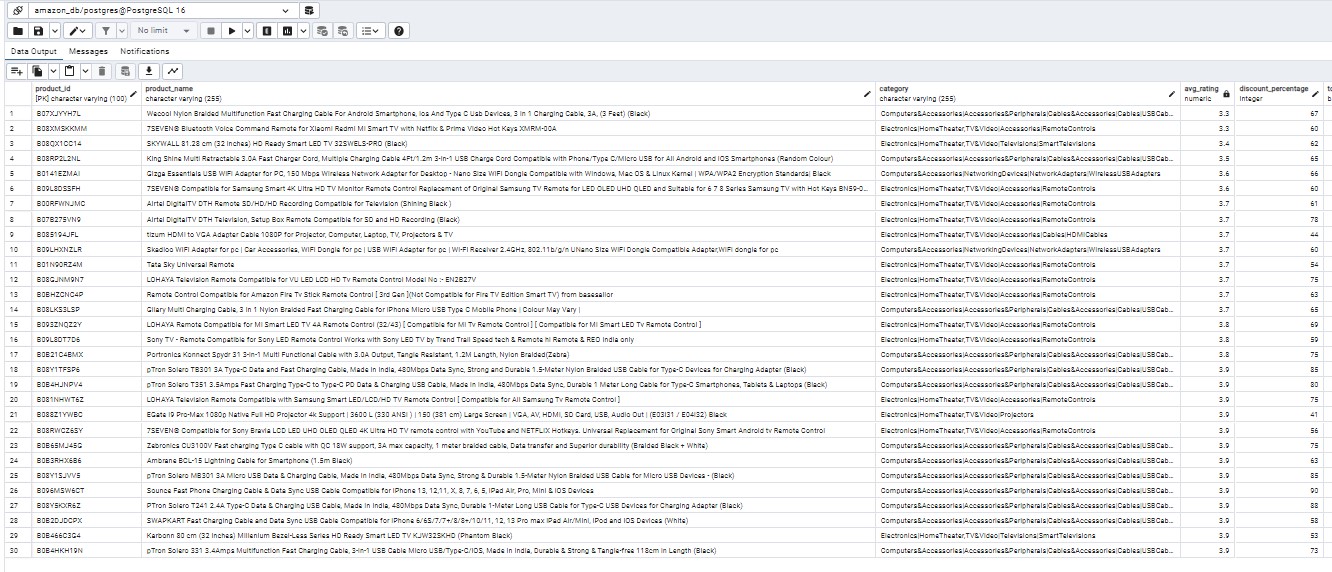
Problem: Products with high discount percentages might indicate areas where pricing strategies need adjustment.

Solution: This query identifies products with the highest average discount percentage, helping us evaluate pricing strategies and profitability.

9.LOW-RATED PRODUCTS WITH HIGH DISCOUNT PERCENTAGE



Result of Low -rated products with high discount percentage



SELECT p.product\_id, p.product\_name, p.category, AVG(pr.rating) AS avg\_rating, p.discount\_percentage, COUNT(pr.review\_id) AS total\_reviews: This part of the query selects six columns:

p.product\_id: The unique identifier for each product.

p.product\_name: The name of each product.

p.category: The category of each product.

AVG(pr.rating) AS avg\_rating: It calculates the average rating for each product by utilizing the AVG function to find the average of the rating column in the product\_reviews table. The result is aliased as avg\_rating.

p.discount\_percentage: The discount percentage for each product.

COUNT(pr.review\_id) AS total\_reviews: It counts the total number of reviews for each product. This count includes even products with no reviews, as the join is a left join.

FROM products p LEFT JOIN product\_reviews pr ON p.product\_id = pr.product\_id: This performs a left join between the products table (p) and the product\_reviews table (pr) based on the product\_id column. It ensures that all products are included in the result set, even if they have no reviews.

GROUP BY p.product\_id, p.product\_name, p.category, p.discount\_percentage: This groups the result set by product\_id, product\_name, category, and discount\_percentage. It ensures that the calculations (average rating and total reviews) are performed for each unique combination of product ID, name, category, and discount percentage.

HAVING AVG(pr.rating) < 4.0 AND p.discount\_percentage > 40: This filters the grouped results to only include products with an average rating less than 4.0 and a discount percentage greater than 40. The HAVING clause is used here because it filters the results based on aggregated values (in this case, the average rating), rather than individual row values.

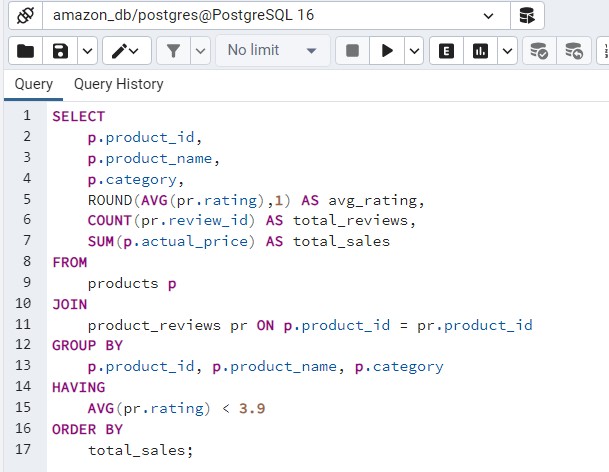
ORDER BY avg\_rating: This orders the result set by the average rating (avg\_rating) in ascending order. This arrangement ensures that the low-rated products appear first.

Overall, the query identifies low-rated products with a high discount percentage by calculating the average rating for each product and filtering the results to include only those with an average rating less than 4.0 and a discount percentage greater than 40. It then orders the results by average rating to present the low-rated products first.

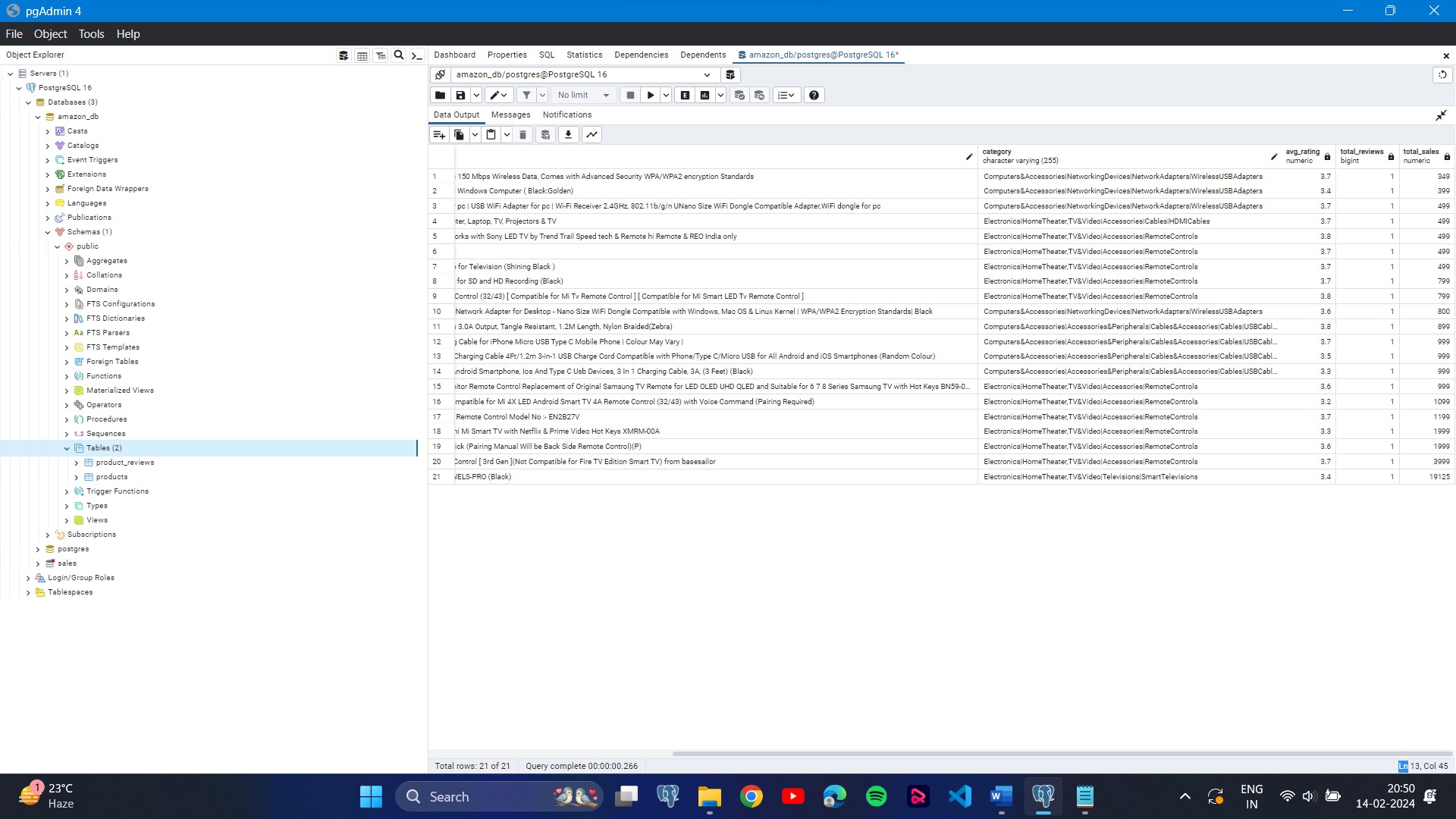
Problem: Low-rated products with high discount percentages may not be attracting customers despite price reductions.

Solution: This query identifies low-rated products with high discount percentages, enabling us to reassess pricing and marketing strategies for these products.

10. LOW-RATED PRODUCTS WITH LOW SALES VOLUME:



Result of low - rated products with low sales volume



SELECT p.product\_id, p.product\_name, p.category, AVG(pr.rating) AS avg\_rating, COUNT(pr.review\_id) AS total\_reviews, SUM(p.actual\_price) AS total\_sales: This part of the query selects six columns:

p.product\_id: The unique identifier for each product.

p.product\_name: The name of each product.

p.category: The category of each product.

AVG(pr.rating) AS avg\_rating: It calculates the average rating for each product by utilizing the AVG function to find the average of the rating column in the product\_reviews table. The result is aliased as avg\_rating.

COUNT(pr.review\_id) AS total\_reviews: It counts the total number of reviews for each product.

SUM(p.actual\_price) AS total\_sales: It calculates the total sales volume for each product by summing up the actual\_price column in the products table. The result is aliased as total\_sales.

FROM products p JOIN product\_reviews pr ON p.product\_id = pr.product\_id: This performs an inner join between the products table (p) and the product\_reviews table (pr) based on the product\_id column. It ensures that only products with reviews are included in the result set.

GROUP BY p.product\_id, p.product\_name, p.category: This groups the result set by product\_id, product\_name, and category. It ensures that the calculations (average rating, total reviews, and total sales) are performed for each unique combination of product ID, name, and category.

HAVING AVG(pr.rating) < 3.9: This filters the grouped results to only include products with an average rating less than 3.9. The HAVING clause is used here because it filters the results based on aggregated values (in this case, the average rating), rather than individual row values.

ORDER BY total\_sales: This orders the result set by the total sales volume (total\_sales) in ascending order. This arrangement ensures that the products with the lowest sales volume appear first.

Overall, the query identifies low-rated products with low sales volume by calculating the average rating, total reviews, and total sales volume for each product, filtering the results to include only those with an average rating less than 3.9, and ordering the results by total sales volume to present the products with the lowest sales volume first.

Problem: Low-rated products with low sales volume may indicate poor product-market fit or quality issues.

Solution: This query identifies low-rated products with low sales volume, allowing us to evaluate whether these products should be improved, marketed differently, or removed from our offerings.