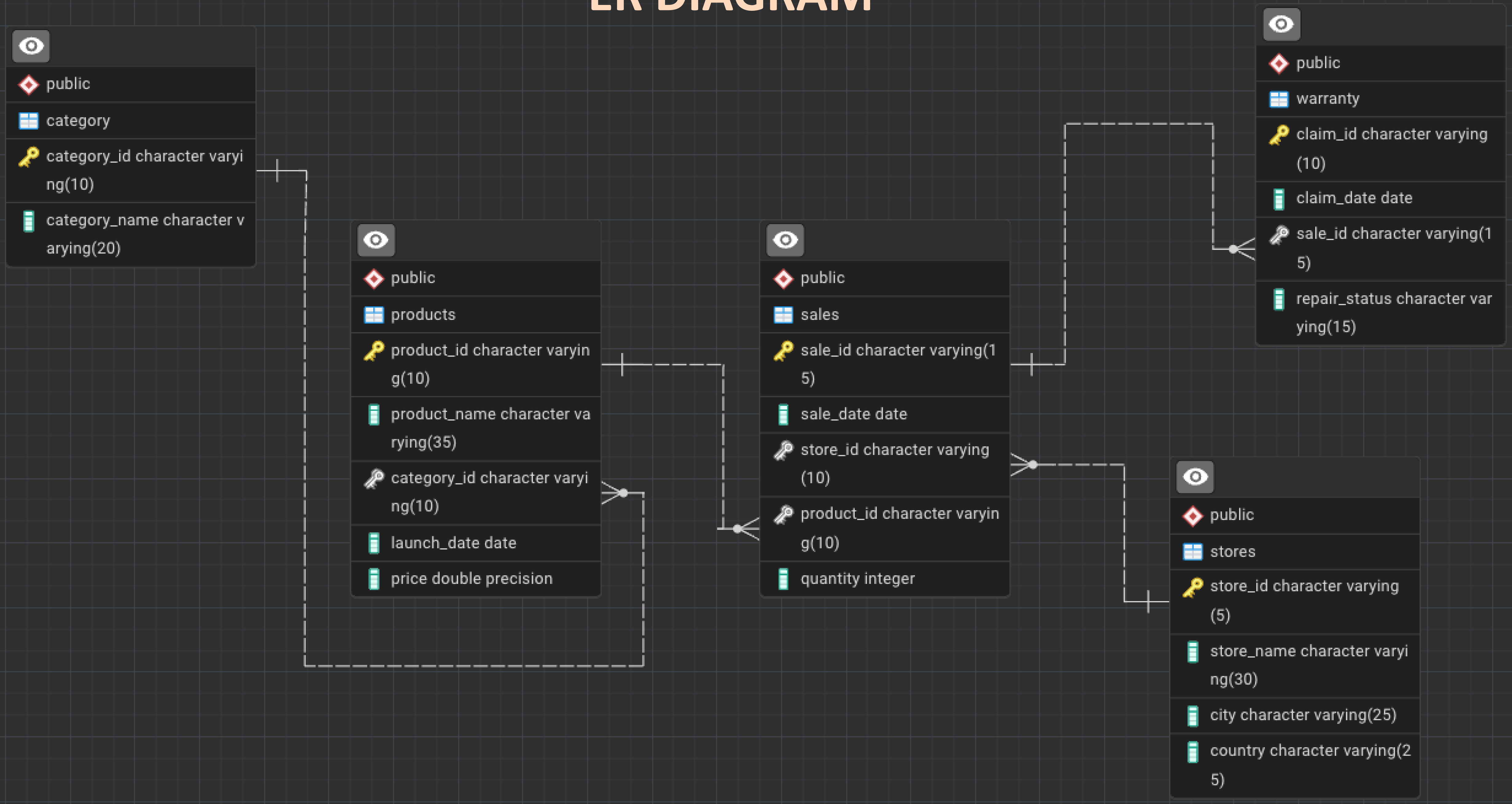
Apple Retail Sales Analysis SQL Project - Analyzing 1 Million Row Data

This project was ideal for me as an aspiring data analyst looking to enhance my SQL skills by working with a large-scale dataset and solving real-world business questions.

ER DIAGRAM



public
category
category_id character varying(10)
category_name character varying(20)

public
products
product_id character varying(10)
product_name character varying(35)
category_id character varying(10)
launch_date date
price double precision

public
sales
sale_id character varying(15)
sale_date date
store_id character varying(10)
product_id character varying(10)
quantity integer

public
warranty
claim_id character varying(10)
claim_date date
sale_id character varying(15)
repair_status character varying(15)

public
stores
store_id character varying(5)
store_name character varying(30)
city character varying(25)
country character varying(25)

Improving Query Performance Through Indexing

Before Indexing

Execution Time: 117.743 ms

Planning Time: 0.09 ms

```
EXPLAIN ANALYSE
SELECT * FROM sales WHERE product_id='P-33
```

After Indexing

Execution Time: 21.867 ms

Planning Time: 0.111 ms

```
EXPLAIN ANALYSE
SELECT * FROM sales WHERE product_id='P-33'
```

The following indexes were created to improve query performance:

```
CREATE INDEX sales_product_id ON sales(product_id);
CREATE INDEX sales_sale_id ON sales(sale_id);
CREATE INDEX sales_sale_date ON sales(sale_date);
```

As shown in the results, indexing reduced the execution time by approximately 81%, demonstrating significant performance improvement.

Store Analysis Queries

Q1. Find the number of stores in each country

```
SELECT country, COUNT(*) AS total_stores
FROM stores
GROUP BY country;
```

Q2. Calculate the total number of units sold by each store

```
SELECT s.store_id, s.store_name, SUM(sa.quantity) AS total_units_sold
FROM stores s
JOIN sales sa ON s.store_id = sa.store_id
GROUP BY s.store_id, s.store_name;
```

Q6. Identify which store had the highest total units sold in the last year

```
SELECT s.store_id, s.store_name, SUM(sa.quantity) AS total_units_sold
FROM stores s
JOIN sales sa ON s.store_id = sa.store_id
WHERE sa.sale_date >= '2024-01-01'
GROUP BY s.store_id, s.store_name
ORDER BY total_units_sold DESC
LIMIT 1;
```

Sales Analysis Queries

Q3. Identify how many sales occurred in December 2023

```
SELECT COUNT(*) AS december_2023_sales
FROM sales
WHERE sale_date BETWEEN '2023-12-01' AND '2023-12-31';
```

Q7. Count the number of unique products sold in the last year

```
SELECT COUNT(DISTINCT product_id) AS unique_products_sold
FROM sales
WHERE sale_date >= '2024-01-01';
```

Q10. For each store, identify the best-selling day based on highest quantity sold

```
SELECT * FROM
(SELECT
store_id,TO_CHAR(sale_date, 'Day') as day_name,
SUM(quantity) as total_unit_sold,
RANK() OVER(PARTITION BY store_id ORDER BY SUM (quantity) DESC) as rank
FROM sales
GROUP BY 1, 2
)as t1
WHERE rank = 1
```

Product and Category Analysis

Q8. Find the average price of products in each category

```
SELECT p.category_id,
c.category_name,AVG(p.price) AS avg_price
FROM products as p JOIN category c
ON p.category_id=c.category_id
GROUP BY 1,2
ORDER BY 3 DESC;
```

Q14. Identify the product category with the most warranty claims filed in the last two years

```
SELECT c.category_name, COUNT(*) AS total_claims
FROM warranty w
JOIN sales s ON w.sale_id = s.sale_id
JOIN products p ON s.product_id = p.product_id
JOIN category c ON p.category_id = c.category_id
WHERE w.claim_date >= CURRENT_DATE - INTERVAL '2 years'
GROUP BY c.category_name
ORDER BY total_claims DESC
LIMIT 1;
```

Warranty Claims Analysis

Claim Frequency

Analyze how often warranty claims are filed across different time periods

Claim Status

Examine the percentage of claims that are approved, rejected, or pending

Product Correlation

Identify which products and categories generate the most warranty claims

Q4. Determine how many stores have never had a warranty claim filed

```
SELECT COUNT (DISTINCT s.store_id) AS stores_without_warranty_claims
FROM stores s
LEFT JOIN sales sa ON s.store_id = sa.store_id
LEFT JOIN warranty w ON sa.sale_id = w.sale_id
WHERE w.sale_id IS NULL;
```

Q5. Calculate the percentage of warranty claims marked as "Rejected"

```
SELECT ROUND((SELECT COUNT(*)
FROM warranty
WHERE repair_status = 'Rejected') * 100.0/(SELECT COUNT(*) FROM warranty), 2) AS Rejected_percentage;
```

Time-Based Warranty Analysis

Q9. How many warranty claims were filed in 2020?

```
SELECT COUNT(*) AS warranty_claims_2020
FROM warranty
WHERE claim_date BETWEEN '2020-01-01' AND '2020-12-31';
```

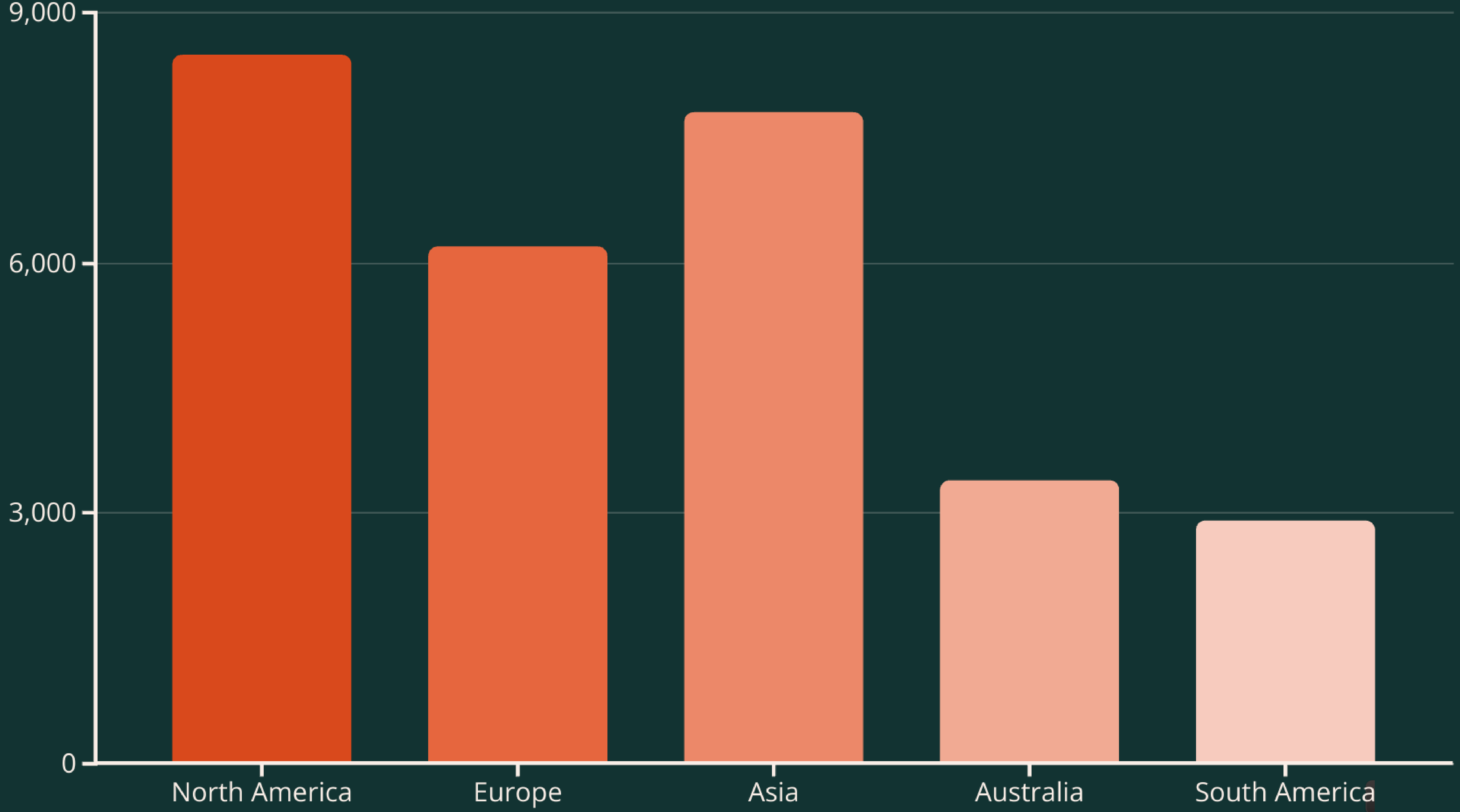
Q11. Calculate how many warranty claims were filed within 180 days of a product sale

```
SELECT COUNT (*) AS claims_within_180_days
FROM warranty w
JOIN sales s ON w.sale_id = s.sale_id
WHERE w.claim_date <= s.sale_date + INTERVAL '180 days';
```

Q12. Determine how many warranty claims were filed for products launched in the last two years

```
SELECT COUNT (*) AS claims_for_recent_products
FROM warranty w
JOIN sales s ON w.sale_id = s.sale_id
JOIN products p ON s.product_id = p.product_id
WHERE p.launch_date >= CURRENT_DATE - INTERVAL '2 years';
```

Regional Sales Performance



Product Performance by Region

Q15. Identify the least selling product in each country for each year based on total units sold

```
WITH ProductSales AS (
SELECT
st.country,
EXTRACT(YEAR FROM s.sale_date) AS sale_year,
s.product_id,
SUM(s.quantity) AS total_units
FROM sales s
JOIN stores st ON s.store_id = st.store_id
GROUP BY st.country, sale_year, s.product_id
),RankedProducts AS (
SELECT
country,
sale_year,
product_id,
total_units,
RANK() OVER (PARTITION BY country, sale_year ORDER BY total_units ASC) AS rank
FROM ProductSales
)
SELECT country, sale_year, product_id, total_units
FROM RankedProducts
WHERE rank = 1;
```

Query Performance Best Practices

Create Appropriate Indexes

Add indexes on frequently queried columns to improve search performance

Optimize Query Structure

Use EXPLAIN ANALYZE to identify and resolve performance bottlenecks

Regular Maintenance

Perform routine database maintenance to ensure optimal performance

Monitor Query Execution

Track execution times and implement improvements where needed

As demonstrated in the first section, proper indexing can dramatically improve query performance. The execution time for our sample query was reduced from 117.743 ms to 21.867 ms after implementing appropriate indexes on the sales table. This represents an 81% improvement in query performance, highlighting the importance of database optimization techniques in business intelligence applications.

THANK YOU

If you're hiring or know someone who is - Let's connect!