**Data Validation Queries**: These queries are used to validate the accuracy of data. They might include complex joins, aggregations, and filters to ensure the data in the dashboard matches the source data. For example, queries that compare aggregates like sums and averages across different tables.

SELECT SUM(table1.column) AS sum\_table1,

SUM(table2.column) AS sum\_table2

FROM table1

INNER JOIN table2 ON table1.id = table2.id

WHERE table1.date = '2023-01-01'

GROUP BY table1.category;

**Performance Testing Queries**: To test how the dashboard performs under load, queries that simulate complex data retrieval scenarios are used. These may include subqueries, complex joins, and queries over large datasets.

SELECT a.\*, b.\*

FROM large\_table a

JOIN another\_large\_table b ON a.id = b.foreign\_id

WHERE a.date BETWEEN '2023-01-01' AND '2023-12-31';

**Security and Access Control Queries**: These queries ensure that data access permissions and security rules are properly implemented. They might involve testing different user roles to ensure that each role has appropriate access to data.

-- Assuming role-based access control

SELECT \*

FROM dashboard\_data

WHERE user\_role = 'standard\_user' AND department = 'Sales';

**Data Consistency and Integrity Queries**: To ensure that the data displayed is consistent and maintains integrity, queries that check for duplications, data mismatches, or orphan records are used.

SELECT id, COUNT(\*)

FROM table

GROUP BY id

HAVING COUNT(\*) > 1; -- finding duplicate records

**Time-based Data Retrieval Queries**: These are important for dashboards that display time-sensitive data. Queries might involve complex date and time functions to retrieve data within specific time frames or to compare data across different periods.

SELECT \*

FROM sales

WHERE sale\_date > CURRENT\_DATE - INTERVAL '30 days';

**Data Formatting and Transformation Queries**: To verify that data is properly formatted and transformed for dashboard display, queries involving string functions, case statements, and data type conversions are used.

SELECT UPPER(name) AS name\_upper,

CAST(price AS DECIMAL(10,2)) AS price\_formatted

FROM products;

**Cross-Database or Cross-Schema Queries**: In environments where data is spread across multiple databases or schemas, queries that join this data are crucial to ensure the dashboard can integrate and display it correctly.

SELECT \*

FROM database1.schema1.table

JOIN database2.schema2.table

ON schema1.table.id = schema2.table.id;

**Historical Data Queries**: For dashboards that display trends over time, queries that retrieve and compare historical data are used. These may involve window functions and complex time-based calculations.

SELECT date,

SUM(sales) OVER (ORDER BY date ROWS BETWEEN 6 PRECEDING AND CURRENT ROW) AS rolling\_7\_day\_total

FROM daily\_sales – rolling 7 day total of sales

* **SUM(sales) OVER (...) AS rolling\_7\_day\_total**: This is the core of the query. It calculates the sum of **sales** for each row, along with the sales of the 6 preceding days. This is known as a rolling or moving sum.
* **ORDER BY date**: This orders the data in the window function by the **date** column. This is crucial because the rolling sum is calculated based on this order.
* **ROWS BETWEEN 6 PRECEDING AND CURRENT ROW**: This defines the window for the rolling sum. For each row in the table, it sums the **sales** of that row and the 6 rows that come before it (based on the date order).

**Predictive Analysis Queries**: If the dashboard includes predictive analytics, queries involving statistical functions or machine learning models might be used to test these features.

-- Example of a simple linear regression model in SQL

SELECT

REGR\_SLOPE(y, x) AS slope,

REGR\_INTERCEPT(y, x) AS intercept

FROM

(SELECT sales AS y, day\_of\_week AS x FROM sales\_data);

**Data Aggregation and Grouping Queries**: These are used to test dashboards that display summarized data. Queries may involve GROUP BY clauses, aggregate functions like COUNT, SUM, AVG, etc., and having clauses for filtering.

SELECT category, COUNT(\*) AS num\_products,

AVG(price) AS average\_price

FROM products

GROUP BY category

HAVING COUNT(\*) > 10;