1. Total number of interactions per day.

SELECT DATE(timestamp) AS interaction\_date, COUNT(\*) AS total\_interactions

FROM interactions

GROUP BY DATE(timestamp)

ORDER BY interaction\_date;

optimization : CREATE INDEX idx\_timestamp ON interactions(timestamp);

Indexing: Create an index on the timestamp column if it's not already indexed.

This can significantly speed up queries involving date-based aggregations.

Partitioning:Though I have used only 10,000 records ,if dataset is really large, we can consider partitioning the interactions table by date range.

Partitioning helps in managing large amounts of data more efficiently by dividing the data into smaller, manageable chunks based on

specified criteria (e.g., by date).

2) Top 5 users by the number of interactions.

SELECT user\_id, COUNT(\*) AS total\_interactions

FROM interactions

GROUP BY user\_id

ORDER BY total\_interactions DESC

LIMIT 5;

optimization :: we can create an index on the user\_id column to speed up grouping and sorting operations.

CREATE INDEX idx\_user\_id ON interactions(user\_id);

Materialized Views: If the query performance is critical and the data doesn't change frequently, consider creating a materialized view that precomputes the

top users.

This way, querying the materialized view would be faster than executing the aggregation query repeatedly.

3) Most interacted products based on the number of

interactions.

SELECT product\_id, COUNT(\*) AS total\_interactions

FROM interactions

GROUP BY product\_id

ORDER BY total\_interactions DESC;

Similar to the top users query, we can create a materialized view for the most interacted products if performance is critical and the

data is relatively static.

Other strategy of optimizing query is using query execution plan .

Query Execution Plan: we can use EXPLAIN command provided by database system to analyze the query execution plan. This helps in

identifying potential bottlenecks and optimizing query performance.