A logo of a university

AI-generated content may be incorrect.A logo of a university

AI-generated content may be incorrect.Subject: Information Management 1

Course/Year: BSIS 2B Professor: Mr. Guillermo Red

**Reflection on Phase 5: Database Optimization and Performance Tuning**

The query aims to calculate the monthly revenue per item type from the purchases table. The EXPLAIN output reveals that MySQL will perform a full table scan (type: ALL) because there are no suitable indexes (possible\_keys: NULL, key: NULL). This means it will read every row in the purchases table. Additionally, MySQL will need to create a temporary table (Using temporary) to handle the GROUP BY operation and perform a filesort (Using filesort) to satisfy the ORDER BY clause, as it cannot use an index for sorting. Consequently, this query is likely to be inefficient and slow, especially as the purchases table grows. To improve performance, consider adding a composite index on (item\_type, purchase\_date) or (YEAR (purchase\_date), MONTH (purchase\_date), item\_type). This query aims to find the top 5 selling items by quantity from the purchases table. The EXPLAIN output shows that MySQL will perform a full table scan (type: ALL) because no suitable indexes are available (possible\_keys: NULL, key: NULL). It will then create a temporary table (Using temporary) to perform the GROUP BY operation and finally use filesort (Using filesort) to order the results by the total quantity sold before applying the LIMIT 5. This indicates an inefficient query, especially for a large purchases table, as it has to process and sort all rows before returning the top 5. To improve performance, consider adding a composite index on (item\_type, item\_id). While an index on quantity might seem relevant for the ORDER BY, it's unlikely to be directly used on the aggregated SUM(quantity). The filesort is likely unavoidable in this scenario without significant query restructuring. This query aims to retrieve the total transactions and total sales for each username by joining fact\_purchases and dim\_users. The EXPLAIN output indicates a full table scan (type: ALL) on the fact\_purchases table, despite the existence of a potential index on the join column (user\_id). Subsequently, it uses a more efficient index lookup (type: ref, key: PRIMARY) on the dim\_users table based on the user\_id from fact\_purchases. However, the initial full scan on fact\_purchases is a significant performance bottleneck. Additionally, the query uses a temporary table (Using temporary) for the GROUP BY operation and performs a filesort (Using filesort) to satisfy the ORDER BY total\_sales DESC clause. In essence, while the join with dim\_users is optimized, the inefficient full scan of fact\_purchases and the subsequent temporary table and filesort suggest that this query's performance could be significantly improved by ensuring the user\_id index on fact\_purchases is utilized.