

Makara Ramoabi

20240045

Database

1. Scenario Analysis

An e-commerce website stores thousands of products in a MySQL database. Customers frequently search for items using keywords, filter by category, and sort by price. Without indexes, the database must scan every row in the products table to find matches, which slows down performance as the product catalog grows. To improve search speed, the database administrator creates indexes on commonly searched columns such as `product_name`, `category_id`, and `price`. When a user searches for “gaming laptop” in the electronics category, MySQL uses the index to quickly locate matching records instead of scanning the entire table. This significantly reduces query execution time and improves user experience. Indexes are especially useful during peak traffic periods, such as sales events, where thousands of customers search simultaneously. Proper indexing ensures faster response times, efficient resource usage, and better scalability as the platform continues to grow.

2. Concept Research

A composite index is an index that includes two or more columns in a table. It is useful when queries frequently filter or sort data using multiple columns together. For example, in an orders table, a composite index on (`customer_id`, `order_date`) helps queries that search for a specific customer’s orders within a date range. The order of columns in a composite index matters because MySQL uses the index from left to right. Composite indexes improve query performance by reducing the number of rows scanned and supporting more efficient filtering and sorting operations in complex queries.

3. Tool Practice

Example SQL Commands:

```
CREATE INDEX idx_category_price  
ON products (category_id, price);
```

```
EXPLAIN SELECT *
```

FROM products

WHERE category_id = 3

AND price < 500;

Reflection

Creating the index in MySQL Workbench was straightforward using the SQL editor. After running the CREATE INDEX command, I used the EXPLAIN statement to analyze how MySQL executed the query. Before indexing, the query performed a full table scan, which was inefficient. After creating the composite index, the execution plan showed that MySQL used the new index to filter rows quickly. The number of examined rows decreased significantly, and the query type improved. This practice helped me understand how indexes directly impact performance and how the EXPLAIN tool helps evaluate optimization effectiveness in real database environments.

4. Flowchart Design

The flowchart begins with a user submitting a query. The system then checks whether relevant indexes exist for the filtered or sorted columns. If indexes are available, the query optimizer evaluates possible execution plans and selects the most efficient one. If no suitable index exists, the database performs a full table scan. The optimizer executes the query and returns results to the user. After execution, performance metrics such as execution time and rows examined are reviewed. If performance is slow, the database administrator analyzes the query and may create or modify indexes. This cycle ensures continuous performance improvement and efficient database operation.