

1.

Indexes are crucial in e-commerce search systems to improve query performance and provide faster results. For example, an online store with thousands of products allows users to search by name, category, or price. Without indexes, every search query would require scanning the entire product table, causing slow response times. By creating indexes on commonly searched columns, such as `product_name` or `category_id`, the database can quickly locate relevant rows. Composite indexes can further optimize searches combining multiple columns. Indexes reduce I/O operations, allowing the system to return search results efficiently. In addition, proper indexing supports filtering, sorting, and pagination, enhancing the user experience. Poorly designed indexes, however, can increase storage and slow down inserts or updates. Therefore, analyzing query patterns and designing appropriate indexes is essential for maintaining high performance in e-commerce platforms, ensuring users find products quickly and the system remains scalable under heavy traffic.

2.

A composite index is an index that includes multiple columns from a database table, improving query performance when filtering or sorting by more than one column. For example, an index on (`category_id`, `price`) helps queries searching for products in a category and sorting by price. Composite indexes allow the database to locate rows efficiently, reducing full table scans. They are particularly useful in complex queries with multiple conditions or `ORDER BY` clauses. Properly designed composite indexes improve read performance while balancing storage and maintenance costs. They are widely used in e-commerce, reporting, and analytical systems to optimize data retrieval speed and system responsiveness.

3.

In MySQL Workbench, I created an index on the `products` table for `product_name` and `category_id`. I then analyzed a query searching for products by category and name using `EXPLAIN` to observe the execution plan. The index allowed the query to quickly locate matching rows without scanning the entire table. Testing showed a significant improvement in

performance for large datasets. Writing and analyzing the query helped me understand how indexes guide the database engine and reduce I/O operations. This hands-on exercise emphasized the importance of indexing strategies, query patterns, and database optimization for efficient, scalable applications in real-world scenarios.

4.

The flowchart illustrates the steps in query optimization. It starts with receiving a SQL query, followed by parsing and syntax checking. The database engine then generates a query execution plan and evaluates multiple strategies using cost estimation. Index usage, join methods, and filtering are analyzed to select the most efficient plan. The selected plan is executed, and results are returned to the user. Feedback from execution can inform future optimizations. This visual representation helps understand how databases improve query performance, minimize resource usage, and ensure efficient retrieval. Flowcharts make optimization processes easier to design, communicate, and maintain.

