Advanced Database

Khaula Molapo

20240001

Week 4

1.

A retail company uses a business intelligence system to study customer buying habits and sales trends. The database stores millions of records, including customer details, products, transactions, and dates. Managers complain that reports take too long to load, especially when filtering data by date or product category. The database administrator investigates and finds that the main problem is slow search performance because the tables are very large and not properly indexed. Every time a report runs, the system scans the whole table, which wastes time and resources. To fix this, the administrator decides to add indexes on commonly searched columns like product ID, transaction date, and customer ID. They also remove unused indexes that slow down data insertion. After tuning the database, report loading becomes much faster, and users can generate insights quickly. This improves business decisions because managers no longer wait long for data. The scenario shows that good indexing and performance tuning are important for business intelligence systems that work with large datasets and frequent reporting queries.

2.

A composite index is an index that includes more than one column in a table. Instead of creating separate indexes for each column, a composite index combines them to make searching faster when queries use multiple conditions together. For example, a database may have a table of sales with columns for customer ID and order date. If most queries search for both columns at the same time, a composite index on those two fields improves performance. Composite indexes help reduce the amount of data the database must scan when running queries. They are useful in reporting systems, business intelligence tools, and applications that filter data using multiple columns. However, developers must plan carefully because too many indexes can slow down insert and update operations. When used correctly, composite indexes improve query speed and make databases more efficient.

3.

SQL Example:

```
CREATE INDEX idx_customer_date ON Orders(CustomerID, OrderDate);
SELECT CustomerID, OrderDate, TotalAmount FROM Orders WHERE CustomerID = 101
AND OrderDate > '2025-01-01';
```

Reflection:

During the simulation, it became clear that indexing helps reduce the number of records

searched, which improves performance. Even though the C++ program is only a simulation, it shows how optimized searching is faster than scanning all data. Using indexes in real databases helps reduce server load and improves user experience. However, indexes must be used carefully because they increase storage and may slow down updates. Overall, the practice showed how indexing improves speed and efficiency in database systems.

4.

The tuning and indexing diagram uses simple square shapes to show how data flows through a database system. The first square represents the user query. The second square shows the database engine receiving the request. A third square shows the index lookup process, where the database checks indexed columns to find matching records quickly. Another square shows the filtered data results, and the last square represents the final output returned to the user. The diagram helps explain how indexing improves performance because the system does not need to scan the entire table. Instead, it uses indexed data to locate records faster. Using simple shapes makes the process easy to understand for beginners. The diagram is useful for presentations and learning because it visually explains how database tuning and indexing reduce search time and improve efficiency.