**1. When the performance of the DB quires starts degrading, what can be done so that the DB query response received in a reasonable time**

**ANS**:-Potential Reasons for Degraded Performance:-

1)Lack of Indexing

2)Increased Data Volume: As the database grows in size, queries may take longer to process due to the increased data to scan

3)Inefficient Query Structure: Poorly optimized queries or inefficient JOINs can result in longer execution times.

4)Data Model Complexity

**There are several actions you can take to improve the response time and overall performance of the database.**

1)Optimize Query Structure

2)Add Indexes: Indexing can significantly speed up query execution, especially for large datasets. Properly chosen indexes can make searching and filtering operations faster.

3)\*\*Avoid SELECT \*\*\*: Retrieve only the required columns instead of using SELECT \* as it might fetch unnecessary data and impact performance.

4)Caching: Implement caching mechanisms to store the results of frequently executed queries. This can reduce the need to fetch data from the database repeatedly.

**2. Database indexes**

Database indexes are data structures used to improve the retrieval speed of data from a database table.

They work like a table of contents or an index in a book, allowing the database management system (DBMS) to quickly find the location of specific data within a table, without having to scan the entire table.

Indexes play a crucial role in optimizing query performance, especially for large datasets,

by reducing the time it takes to execute SELECT, UPDATE, DELETE, and JOIN operations.

**3. Type of indexes**

1)**Clustered index** is an index in which the physical order of data in the table is based on the indexed columns, allowing for efficient retrieval of data in a specific order.

2)**Non-clustered** index is an index that stores a separate copy of data sorted based on the indexed columns, with pointers to the actual data rows, enabling faster searching and filtering operations.

3)**Unique index** enforces the uniqueness of values in the indexed column(s), preventing duplicate entries.

4)**Filtered index** is a selective index that includes only a subset of rows from a table based on a specified condition, optimizing performance for queries on a specific data subset.

5)**Columnstore index** is an index designed for large-scale data warehousing, storing data column-wise instead of row-wise, leading to significant compression and faster analytical queries.

6)**Hash index** uses a hash function to generate a hash code for each indexed value, providing rapid access to exact match searches, but not suitable for range queries or comparison operations.

4. When will indexes fail, and why?

Indexes in MySQL can fail when the following conditions are met:

* **The index is not used by the query.** This can happen if the query does not use the columns that are indexed. For example, a query that only selects the name column of a table will not use the id index, even if the id column is indexed.
* **The index is fragmented.** A fragmented index is an index that is spread out over multiple data pages. This can happen if a lot of data has been inserted or deleted from the table. A fragmented index can be slower than a non-fragmented index.
* **The index is too large.** A large index can be slower than a smaller index. This is because the MySQL server has to read more data from disk to access a large index.
* **The index is not used often.** If an index is not used often, the MySQL server may drop it in order to save space. This can happen if the index is not used by any of the queries that are running against the table.