**Interview Questions for Indexes**:

Sure, Talha! I’ll go over **interview questions related to indexes** and provide detailed answers for each, covering various index types, their use cases, and the underlying concepts. I'll ensure you're well-prepared for any indexing-related questions in interviews!

**1. What are indexes and how do they improve query performance?**

**Answer:**  
Indexes are database structures that allow faster access to data. They function like the index in a book, where you don’t need to read every page to find the information you’re looking for. Instead, you can directly go to the relevant section using the index.

Indexes **improve query performance** by reducing the number of rows the database needs to scan. When a query is executed, the database can use an index to quickly locate rows that meet the query’s conditions, significantly speeding up data retrieval, especially in large tables.

For example:

SELECT \* FROM Rides WHERE ride\_id = 100;

* **With Index**: The database can jump directly to the row with ride\_id = 100.
* **Without Index**: The database will scan the entire table to find the matching row.

**2. Can a table have multiple clustered indexes?**

**Answer:**  
No, a table can have **only one clustered index** because a clustered index determines the physical order of the data in the table. Since data can only be ordered in one way, there can be only one clustered index per table.

**Why?**  
The data rows themselves are stored in the **order of the clustered index**. If you had multiple clustered indexes, the data would need to be sorted in multiple ways, which isn’t possible.

**3. When would you use a composite index?**

**Answer:**  
A **composite index** is used when you frequently run queries that involve **multiple columns** in the **WHERE clause**. Instead of creating multiple single-column indexes, a composite index can be created to optimize queries that filter by multiple columns.

**Example Use Case:**

For a table Rides, if you often search using both passenger\_id and fare:

SELECT \* FROM Rides WHERE passenger\_id = 'P02' AND fare = 300;

You can create a composite index like this:

CREATE INDEX idx\_passenger\_fare ON Rides(passenger\_id, fare);

This allows the database to **quickly filter** based on both columns together, improving query performance.

**4. How do clustered and non-clustered indexes differ?**

**Answer:**

* **Clustered Index:**
  + The data in the table is **physically ordered** by the clustered index. There can only be **one clustered index per table**.
  + The **primary key** is usually the clustered index.
  + Example: If a table is ordered by ride\_id, the rows will be physically stored in order of ride\_id.
* **Non-clustered Index:**
  + The index structure is **separate** from the data table. It contains a **pointer** to the actual data rows.
  + You can have **multiple non-clustered indexes** on a table.
  + Example: You can create an index on fare without rearranging the actual data rows in the table.

**When to use**:

* **Clustered index** is used when you need to **order data** by a key.
* **Non-clustered index** is useful for frequently queried columns that are not the primary key.

**5. What is a bitmap index and when would you use it?**

**Answer:**  
A **bitmap index** is used for columns with **low cardinality**, i.e., columns with a **limited number of distinct values** (e.g., "active" vs "inactive"). The index uses a **bitmap** (a sequence of bits) to represent the presence or absence of each distinct value.

**When to use:**

* **Low cardinality columns**: Gender, status (active/inactive), etc.
* **Efficient for queries**: When the query involves **AND/OR** operations between multiple columns with low cardinality.

**Example Use Case:**

If you have a Rides table with a status column that has only two values: active and inactive. A bitmap index on status would allow the database to quickly determine which rows are active or inactive without scanning the entire table.

**6. How does a B-tree work in indexing?**

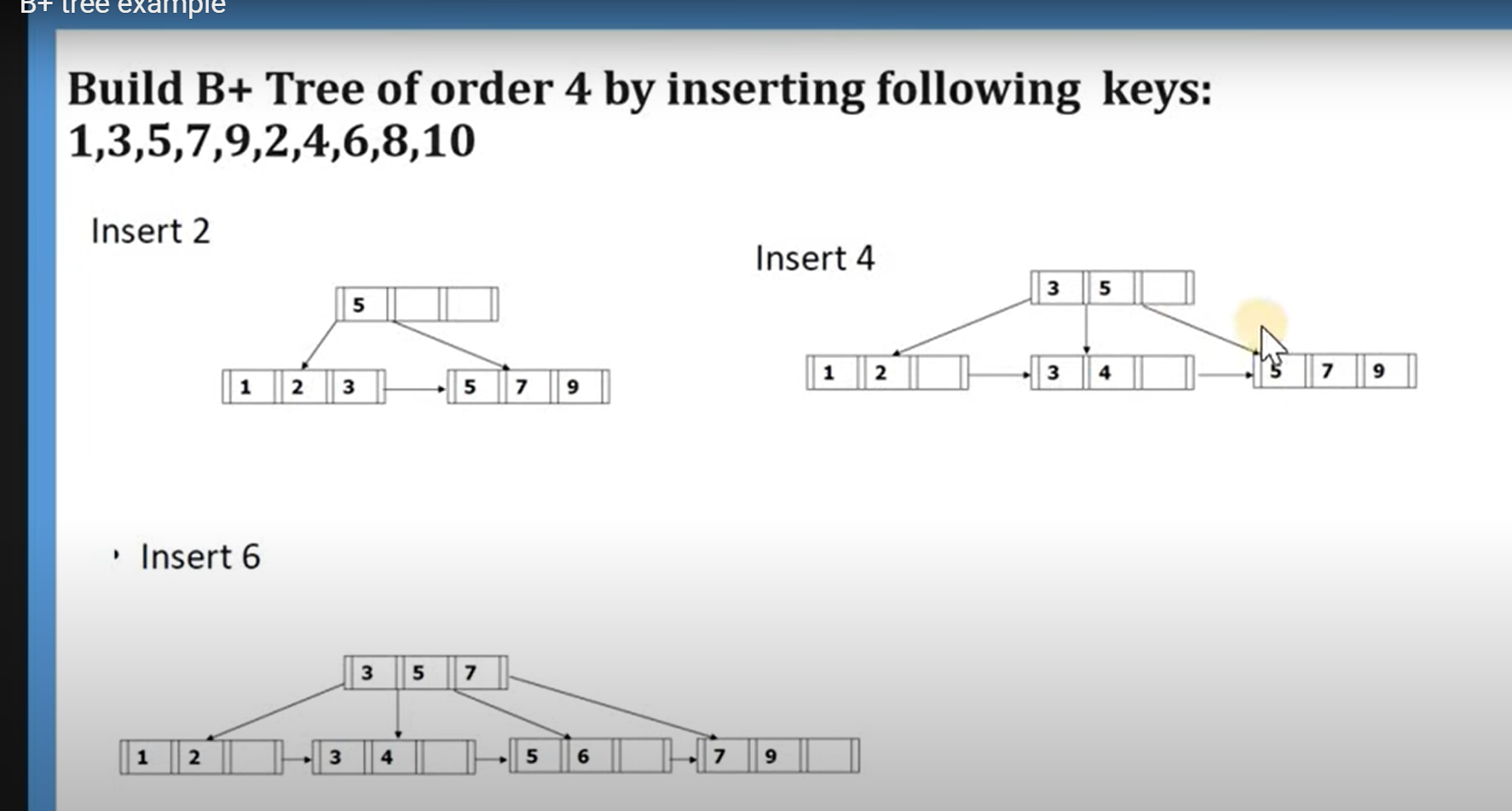
**Answer:**  
A **B-tree** (Balanced Tree) is the most common data structure used for indexing. It is a **self-balancing tree** structure that ensures **sorted data** and allows for **efficient searching, insertion, and deletion** operations.

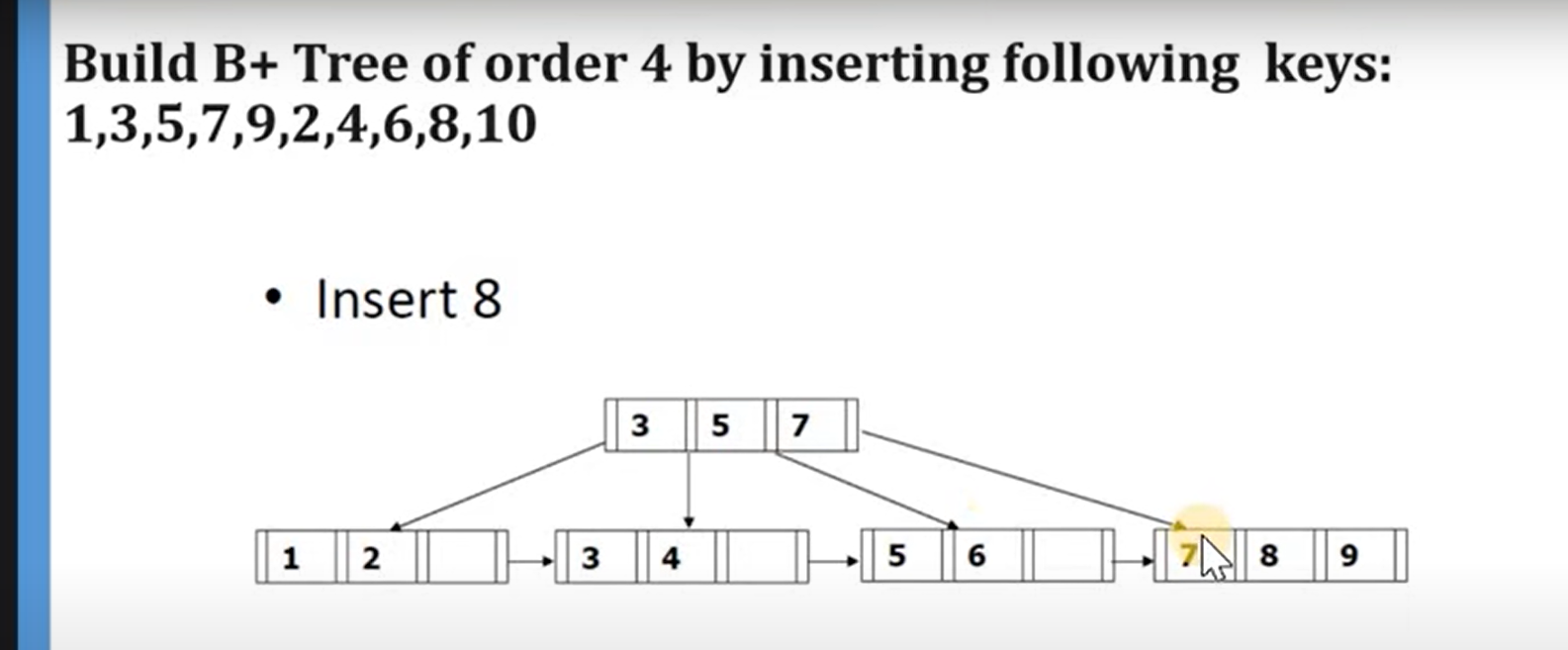
* Each **node** in the B-tree represents a range of values.
* The **leaf nodes** contain pointers to the actual data rows in the table.
* **Searching** in a B-tree is logarithmic, i.e., O(log n), which allows quick lookups.

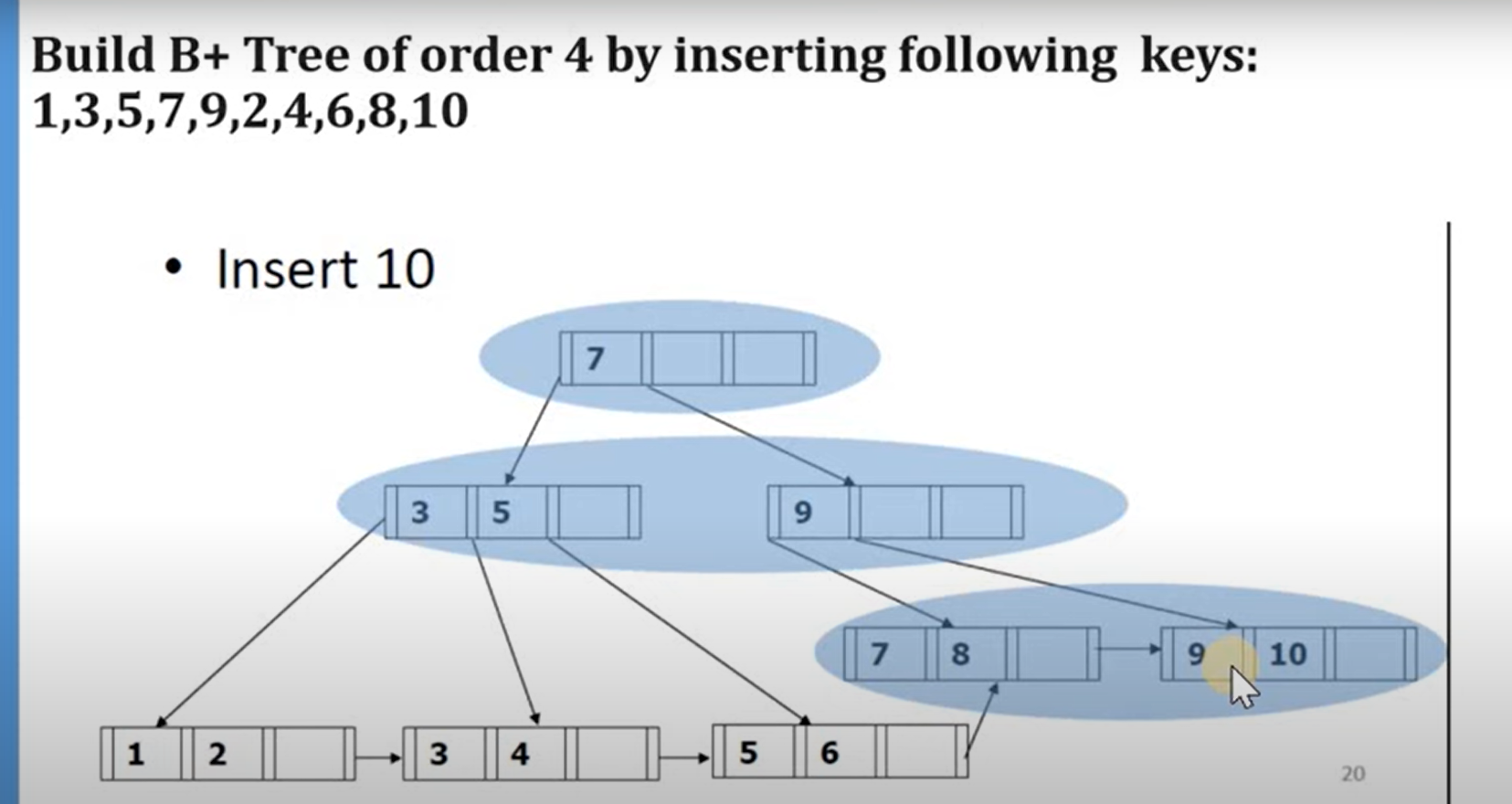
**Example:**

If you search for a value in the index, it will start from the **root node**, move through **branches** to reach the **leaf node**, and then get the **pointer** to the data row.

**Use Case**: When you create an index on a column like ride\_id, the database uses a B-tree to search efficiently for rows.







**7. What’s the difference between a full-text index and a regular index?**

**Answer:**  
A **full-text index** is designed for **text searching** on large text fields like descriptions or comments. It works by breaking the text into **tokens** (words) and storing them in an optimized way to speed up search queries like **MATCH** and **AGAINST** in MySQL.

**Difference:**

* **Full-text index** allows efficient searches for words or phrases, even when the words are **not exact matches**.
* **Regular index** (e.g., B-tree) is typically used for exact matches or range queries.

**Example Query:**

SELECT \* FROM Rides WHERE MATCH(comment) AGAINST('fast');

**When to use:**

* **Full-text indexes** are useful for search applications that need to search for keywords within a large text field.

**8. What is the impact of indexes on write performance?**

**Answer:**  
Indexes **speed up read operations**, but they **slow down write operations** (INSERT, UPDATE, DELETE) because:

* When data is inserted, the index must be updated as well.
* When a column that’s indexed is updated, the index also needs to be updated.
* For DELETE, the corresponding index entries must also be removed.

**Example:**

If a table has multiple indexes, each insert will take longer because the system needs to update all the indexes.

**Tip for interview:** If the table has frequent inserts or updates, it’s important to evaluate whether indexing is necessary or if you can **avoid indexing on columns that don’t need frequent querying**.

**9. What is the difference between TRUNCATE, DELETE, and DROP?**

**Answer:**

1. **TRUNCATE**:
   * **Deletes all rows** in a table without logging each row deletion.
   * **Faster** than DELETE because it doesn't log individual row deletions and doesn't trigger foreign key constraints.
   * Does not fire triggers.
   * Cannot be rolled back in some databases (like MySQL).
2. TRUNCATE TABLE Rides;
3. **DELETE**:
   * Deletes rows **one by one**, and **can be rolled back**.
   * Triggers **foreign key constraints** and **fires triggers**.
   * Slower compared to TRUNCATE.
4. DELETE FROM Rides WHERE fare < 300;
5. **DROP**:
   * Removes the **entire table** (structure and data) from the database.
   * **Cannot be rolled back**.
   * Does not remove indexes separately (they are removed automatically when the table is dropped).
6. DROP TABLE Rides;

**Difference with example:**

* **TRUNCATE**: Removes all rows but keeps the structure. Fast.
* **DELETE**: Removes rows but can be rolled back and impacts performance more.
* **DROP**: Removes both the data and the table.

**10. What is a Surrogate Key?**

**Answer:**  
A **Surrogate Key** is a system-generated key that uniquely identifies rows in a table. It usually has **no business meaning** and is often an auto-incremented number.

**Why use a Surrogate Key?**

* **No semantic meaning**: You can have a surrogate key like driver\_id instead of using the driver’s name as a primary key.
* **Performance**: It is smaller and easier to index, improving query performance.

**Example**:  
A Drivers table might use a surrogate key driver\_id:

CREATE TABLE Drivers (

driver\_id INT AUTO\_INCREMENT PRIMARY KEY,

name VARCHAR(100)

);

This is more efficient than using name as the primary key, especially when names can repeat.