SESSION WILL BE DIVIDED INTO

1. What is Optimization in SQL?
2. How to Achieve Optimization?
3. Partition in SQL
    a) What is Partition in SQL?
    b) Types of Partitioning in SQL
    c) Implementation
4. Indexing in SQL
5. Partitioning vs Indexing

SECTION 1

Optimization In SQL

Concept of Optimization in SQL

- **Optimization** means making SQL queries run **faster**, **efficiently**, and **using fewer resources** (CPU, Memory, Disk).
- Goal: Retrieve or manipulate data **quickly**, **without scanning unnecessary data**.
- Important because data volumes are **growing rapidly** (millions to billions of records).

SECTION 2

How To Achieve Optimization

- **How to achieve optimization in SQL?**
  - There are several techniques used to optimize performance:
    a) **Partitioning the Database**
       1. Break a large table into **smaller, manageable partitions**.
       2. Queries can **access only the necessary partition ➔ faster data retrieval**.
    b) **Indexing**
       1. Create **special lookup structures** (Indexes) to **quickly find data** without scanning the whole table.
       2. Works like a **Table of Contents** in a book.
    c) **Other Techniques**
       1. **Proper Query Writing** (e.g., avoiding SELECT *)
       2. **Using Joins carefully** (e.g., choosing correct join types)
       3. **Materialized Views** (precomputed query results)
       4. **Caching** (storing frequent query results temporarily)
       5. **Using Proper Data Types** (smaller, efficient types)

SECTION 3

Partition In SQL

3. A) What is Partitioning?

- Dividing a large table **horizontally** into **smaller sections**.
- Each section is called a **Partition**.
- From the user's view, the table is still **one single table**.
- Optimizes **read/write performance**, and **maintenance** becomes easier.

SECTION 3

Partition In SQL

3. B) Types of Partitioning?

Type	Description	Use Case Example
Range Partitioning	Based on a range of values	Orders partitioned by year
List Partitioning	Based on a list of values	Customers partitioned by country
Hash Partitioning	Based on hashing function	Transactions distributed randomly
Composite Partitioning	Combination (Range + Hash)	Sales partitioned by year and customer ID

SECTION 3

Partition In SQL

3. c) Implementation of Partition

- **Range Partitioning**

```
CREATE TABLE table_name (
    column1 datatype,
    column2 datatype,
    ...
)
PARTITION BY RANGE (column_name) (
    PARTITION partition_name1 VALUES LESS THAN (value1),
    PARTITION partition_name2 VALUES LESS THAN (value2),
    ...
    PARTITION partition_max VALUES LESS THAN (MAXVALUE)
);
```

SECTION 3

Partition In SQL

3. c) Implementation of Partition

- **List Partitioning**

```
CREATE TABLE table_name (
    column1 datatype,
    column2 datatype,
    ...
)
PARTITION BY LIST (column_name) (
    PARTITION partition_name1 VALUES (value1, value2, ...),
    PARTITION partition_name2 VALUES (value3, value4, ...),
    ...
);
```

SECTION 3

Partition In SQL

3. c) Implementation of Partition

- **Hash Partitioning**

```
CREATE TABLE table_name (
    column1 datatype,
    column2 datatype,
    ...
)
PARTITION BY HASH (column_name)
PARTITIONS number_of_partitions;
```

SECTION 3

Partition In SQL

3. d) Summary

Type	Syntax Keyword	Best for
Range Partitioning	PARTITION BY RANGE	Dates, numbers
List Partitioning	PARTITION BY LIST	Specific values like countries
Hash Partitioning	PARTITION BY HASH	Random spread
Composite Partitioning	PARTITION BY RANGE + SUBPARTITION BY HASH	Large, complex tables

SECTION 4

Indexing In SQL

3. A) What is Indexing in SQL?

1. **What is Indexing?**
   - Indexing creates **a small, efficient data structure** to **locate rows quickly** without scanning the entire table.
   - Works **like a book's index**: find topics instantly without reading every page.

2. **Benefits of Indexing**
   - **Speeds up SELECT queries** (especially WHERE conditions and JOINs).
   - **Improves sorting and grouping**.
   - **Faster searches** on large tables.

SECTION 4

Indexing In SQL

3. B) Implementation of Indexing

```
CREATE INDEX idx_customer_id
ON Customers (CustomerID);
```