What is Partitioning and Sharding in DBMS || DB Optimisation

1. WHAT IS PARTITIONING?

A big problem becomes easy when it is broken into smaller parts.
Partitioning follows this exact principle in databases.

Partitioning divides a very large database table into smaller, manageable pieces called partitions.
These partitions are still part of the same table and SQL queries work on them normally, without any change.

Why this helps:
DDL operations (ALTER, DELETE, UPDATE) work faster on smaller partitions instead of one huge table.

Real-world example:
Instead of keeping all students' files in one huge cupboard, we store them class-wise in separate drawers.

2. PARTITIONING ACROSS MULTIPLE SERVERS

Partitioning can also be used to divide database objects across multiple servers.
This improves performance, control, and scalability.

Problem:
Relational databases are hard to scale horizontally (adding servers) because relations must be maintained.

Solution:
If the database is already spread across servers, partitioning helps divide data cleanly among them.

Real-world example:
Instead of one shop handling all customers, multiple branches handle nearby customers.

3. VERTICAL PARTITIONING (COLUMN-WISE SPLIT)

Key Points:

1. Relation is sliced vertically (by columns).
2. Different columns are stored on different servers.
3. To get a full record (tuple), data must be fetched from multiple servers.

Example:

Student table:

- Server 1 → Roll No, Name
- Server 2 → Address, Marks

Real-world example:

Your personal details are in one office, academic records in another office.

4. HORIZONTAL PARTITIONING (ROW-WISE SPLIT)

Key Points:

1. Relation is sliced horizontally (by rows).
2. Each server stores complete rows but for different ranges.
3. Each partition is independent.

Example:

Server 1 → Students with Roll No 1–1000
Server 2 → Students with Roll No 1001–2000

Real-world example:

Each bank branch maintains full records, but only for customers of that area.

5. WHEN IS PARTITIONING APPLIED?

1. When the dataset becomes extremely large and hard to manage.
2. When request traffic is very high and a single DB server becomes slow.
3. When response time increases due to heavy load.

Real-world example:

One cashier is too slow for a crowded mall → multiple counters are opened.

6. ADVANTAGES OF PARTITIONING

1. PARALLELISM

Multiple servers process queries at the same time.

2. AVAILABILITY

If one partition/server fails, others still work.

3. PERFORMANCE

Queries scan smaller datasets → faster execution.

4. MANAGEABILITY

Backup, restore, delete operations become easier.

5. REDUCED COST

Scaling up (bigger machine) is expensive.
Scaling out (more machines) using partitioning is cheaper.

7. DISTRIBUTED DATABASE

Definition:

A distributed database is a single logical database spread across multiple physical servers connected by a network.

Important Points:

1. Appears as one database to the user.
2. Data is stored at different locations.
3. Uses optimisation techniques like:

- Clustering
- Partitioning
- Sharding

Why needed?
Because of huge data size and massive request load (see point 5).

Real-world example:
Google Drive looks like one storage, but data is stored in many data centers.

8. SHARDING

What is Sharding?

Sharding is a technique to implement Horizontal Partitioning.

Core Idea:

Instead of storing all data in one DB instance, data is split across multiple DB instances (shards).
A routing layer decides which shard should handle the request.

Example:
User ID 1–1M → Shard 1
User ID 1M–2M → Shard 2

Real-world example:
Courier hub routes parcels to the correct city warehouse.

9. PROS OF SHARDING

1. SCALABILITY

Easy to add more shards as data grows.

2. AVAILABILITY

Failure of one shard does not stop the entire system.

10. CONS OF SHARDING

1. COMPLEXITY

Shard mapping logic is complex.
Routing layer must be maintained.

2. RE-SHARDING PROBLEM

Uneven data distribution may require reshuffling data.

3. ANALYTICAL QUERIES ISSUE

Data is spread across shards.
Query must collect data from all shards (Scatter-Gather problem).
This makes analytics slow.

Real-world example:
Counting total sales requires collecting reports from every branch.