Lesson 9 -> Partitioning vs Sharding

1️⃣ Partitioning vs Sharding (Clear Distinction)

Partitioning

- **Logical division of data**
- Usually **within the same database system**
- May or may not be on different machines

> Think: “How is data split inside a DB?”

Sharding

- **Physical distribution of data across multiple databases / nodes**
- Always implies **horizontal partitioning**
- Used when a single DB cannot handle scale

> Think: “Which database node holds this data?”

Simple Analogy

- Partitioning → rooms inside one building
- Sharding → multiple buildings

2️⃣ Database Partitioning (Techniques)

2.1 Horizontal Partitioning (Row-based)

What it is

- Rows are split across partitions
- Same schema in each partition

Example:

`orders_2024 orders_2025`

Each partition contains different rows.

Pros

✅ Scales reads & writes

✅ Queries scan less data

Cons

❌ Cross-partition joins expensive

❌ Partition management complexity

2.2 Vertical Partitioning (Column-based)

What it is

- Columns split into separate tables
- Based on access patterns

Example:

`users_basic (id, name) users_profile (id, address, preferences)`

Pros

✅ Faster reads for common queries

✅ Smaller row size

Cons

❌ Joins needed for full data

❌ Schema complexity

2.3 Range-Based Partitioning

What it is

- Data partitioned based on value ranges

Example:

`orders: Jan–Mar → P1 Apr–Jun → P2`

Pros

✅ Very efficient range queries

✅ Good for time-series data

Cons

❌ Hot partitions
❌ Manual rebalancing

Used heavily in:

- Retail orders
- Logs
- ML features by time

2.4 Hash-Based Partitioning

What it is

- Hash function decides partition

Example:

```
hash(user_id) % 4 → partition
```

Pros

✅ Even data distribution
✅ Avoids hot partitions

Cons

❌ Poor range queries
❌ Rehashing pain when adding partitions

2.5 Key-Based Partitioning

What it is

- Partitioning based on a natural key

Example:

```
user_id → partition
```

Often implemented via hashing.

Pros

✅ Predictable routing
✅ Simple logic

Cons

❌ Key skew causes imbalance

2.6 Composite Partitioning

What it is

- Combine multiple strategies

Example:

- Range by date
- Hash by user_id inside range

Used in **high-scale retail & ML systems**.

3️⃣ Database Sharding (Techniques)

Sharding is **horizontal partitioning + distribution**.

3.1 Range-Based Sharding

How it works

```
Shard 1 → user_id 1–1M Shard 2 → user_id 1M–2M
```

Pros

✅ Simple routing
✅ Range queries efficient

Cons

❌ Hot shards
❌ Rebalancing difficult

3.2 Hash-Based Sharding

How it works

`hash(user_id) % N → shard`

Pros

✅ Uniform load
✅ Easy scale (initially)

Cons

❌ Resharding is painful
❌ Range queries inefficient

3.3 Consistent Hashing (Important)

How it works

- Hash ring
- Minimal key movement when adding/removing shards

Pros

✅ Scales well
✅ Minimal data reshuffle

Cons

❌ Complex to implement
❌ Debugging harder

Used in:

- Cassandra
- DynamoDB
- Redis Cluster

3.4 Directory-Based Sharding

How it works

- Lookup service maps key → shard

Example:

`user_id → shard_3`

Pros

✅ Flexible
✅ Easy rebalancing

Cons

❌ Directory is SPOF
❌ Extra network hop

3.5 Geo-Based Sharding

How it works

- Shards based on geography

Example:

- India → shard IN
- US → shard US

Pros

✅ Low latency
✅ Compliance friendly

Cons

❌ Cross-region queries hard

Used by:

- Global retail platforms
- Recommendation systems

4️⃣ Partitioning vs Sharding (Side-by-Side)

Aspect	Partitioning	Sharding
Scope	Logical	Physical

Aspect	Partitioning	Sharding
Machines	Same or multiple	Multiple
Scaling	Limited	High
Complexity	Medium	High
Failure impact	Local	Larger

5️⃣ ML & Retail-Specific Examples

Feature Store

- Partition by date
- Shard by entity_id

User Embeddings

- Hash-based sharding
- Consistent hashing

Online Inference Cache

- Redis cluster with hash slots

Training Data

- Range partitioned by time
- Sharded by region

6️⃣ Failure & Edge Cases (Architect Thinking)

Hot Partitions

- Skewed keys
- Celebrity users

Solution:

- Salting keys
- Adaptive sharding

Cross-Shard Queries

- Aggregations across shards

Solution:

- Fan-out queries
- Pre-aggregations

Rebalancing

- Adding/removing nodes

Solution:

- Consistent hashing
- Background migration

7️⃣ Interview-Ready One-Liners

- Partitioning organizes data inside a database; sharding distributes it across databases.
- Sharding always implies horizontal partitioning.
- Hash-based sharding favors uniform load; range-based favors query efficiency.
- Consistent hashing minimizes data movement during scaling.

How This Fits Your Growth Path

Understanding this means:

- You can design **feature stores**
- You can scale **online inference**
- You can discuss **data-heavy system tradeoffs**

This is **ML Architect / Head of DS-level knowledge**.