**Data Partitioning in SQL**

**What is Data Partitioning and why needed:**

* Partitioning divides a large table into smaller, more manageable chunks.
* Helps **improve performance**, **simplify maintenance**, and **scale efficiently**.

**Example in banking:**

A bank processes millions of transactions daily. Running reports on such a massive table without partitioning can be painfully slow and inefficient.

**Types of Data Partitioning in SQL**

| **Type** | **Description** | **Banking Example** |
| --- | --- | --- |
| **Range Partitioning** | Based on value range (e.g., dates). | Partition by TransactionDate |
| **List Partitioning** | Based on a list of values. | Partition by BranchCode |
| **Hash Partitioning** | Uses a hash function for even data distribution. | Partition by AccountID |
| **Composite Partitioning** | Combines methods. | Range on TransactionDate + Hash on AccountID |

**Benefits of Data Partitioning**

* 🚀 Faster queries by scanning smaller partitions.
* 🛠️ Easier maintenance (e.g., purge old partitions).
* 📦 Efficient backups & restores.
* ⚙️ Parallelism support for performance gains.
* 🔒 Logical separation for better security control.

**Practical Scenarios for Banking Domain**

| **Scenario** | **Recommended Partitioning** |
| --- | --- |
| Monthly transaction summary | Range Partition on TransactionDate |
| Branch performance report | List Partition on BranchCode |
| Large data volume load balancing | Hash Partition on AccountID |
| Composite need (region + date) | Composite: List + Range |