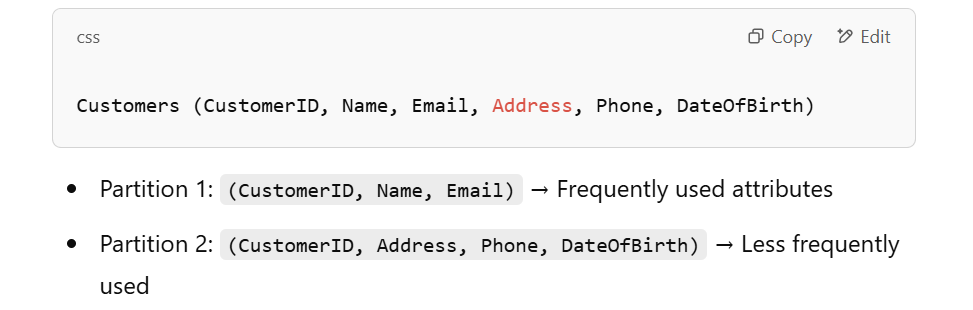
**Vertical and Horizontal Partitioning of Database Tables**

**1. Vertical Partitioning**

Vertical partitioning involves splitting a table into smaller tables by grouping **columns (attributes)**. Each partition contains a subset of columns but retains the primary key for reconstruction.

* **Use Case**: It is beneficial when queries frequently access only a subset of columns, reducing I/O operations.
* **Example**: Consider a Customers table:



**Advantages**:

* Improves performance for column-specific queries
* Reduces memory usage for frequently accessed attributes
* Useful in **column-oriented databases** (e.g., OLAP systems)

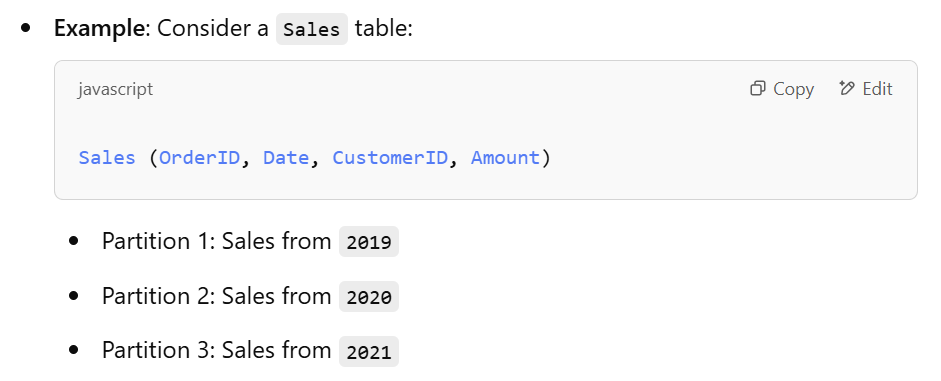
**Disadvantages**:

* Requires expensive **join operations** to reconstruct the original table
* Not ideal if queries need all columns frequently

**2. Horizontal Partitioning**

Horizontal partitioning divides a table into multiple smaller tables (partitions) based on **rows (tuples)**. Each partition contains a subset of the data but all columns.

* **Types**:
  + **Range Partitioning**: Data is divided based on a column value range (e.g., partition by year).
  + **Hash Partitioning**: A **hash function** determines the partition based on a column (e.g., CustomerID % 4).



**Advantages**:

* Queries scanning a specific range of data are **faster**
* Improves **load balancing** in distributed systems
* Supports **parallel query execution**

**Disadvantages**:

* Requires **partitioning logic** in queries
* Can lead to **data imbalance** if not well distributed

**3. Differences in Allocation**

* **Vertical Partitioning**: Data is allocated **by attributes (columns)**. It improves performance for attribute-specific queries but requires joins.
* **Horizontal Partitioning**: Data is allocated **by records (rows)**. It improves performance for queries on subsets of data but requires indexing strategies.