# 11. PARTITIONING STRATEGY

Partitioning is done to enhance performance and facilitate easy management of data. Partitioning also helps in balancing the various requirements of the system. It optimizes the hardware performance and simplifies the management of data warehouse by partitioning each fact table into multiple separate partitions. In this chapter, we will discuss different partitioning strategies.

## Why is it Necessary to Partition?

Partitioning is important for the following reasons:

- For easy management,
- To assist backup/recovery,
- To enhance performance.

### For Easy Management

The fact table in a data warehouse can grow up to hundreds of gigabytes in size. This huge size of fact table is very hard to manage as a single entity. Therefore it needs partitioning.

### To Assist Backup / Recovery

If we do not partition the fact table, then we have to load the complete fact table with all the data. Partitioning allows us to load only as much data as is required on a regular basis. It reduces the time to load and also enhances the performance of the system.

**Note:** To cut down on the backup size, all partitions other than the current partition can be marked as read-only. We can then put these partitions into a state where they cannot be modified. Then they can be backed up. It means only the current partition is to be backed up.

### To Enhance Performance

By partitioning the fact table into sets of data, the query procedures can be enhanced. Query performance is enhanced because now the query scans only those partitions that are relevant. It does not have to scan the whole data.

# Horizontal Partitioning

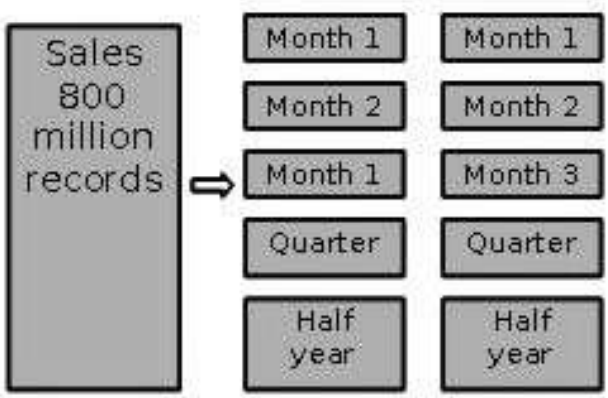
There are various ways in which a fact table can be partitioned. In horizontal partitioning, we have to keep in mind the requirements for manageability of the data warehouse.

## Partition by Time into Equal Segments

In this partitioning strategy, the fact table is partitioned on the basis of time period. Here each time period represents a significant retention period within the business. For example, if the user queries for **month to date data** then it is appropriate to partition the data into monthly segments. We can reuse the partitioned tables by removing the data in them.

## Partition by Time into Different-sized Segments

This kind of partition is done where the aged data is accessed infrequently. It is implemented as a set of small partitions for relatively current data, larger partition for inactive data.




### Points to Note

- The detailed information remains available online.
- The number of physical tables is kept relatively small, which reduces the operating cost.
- This technique is suitable where a mix of data dipping recent history and data mining through entire history is required.
- This technique is not useful where the partitioning profile changes on a regular basis, because repartitioning will increase the operation cost of data warehouse.

## Partition on a Different Dimension

The fact table can also be partitioned on the basis of dimensions other than time such as product group, region, supplier, or any other dimension. Let's have an example.

Suppose a market function has been structured into distinct regional departments like on a **state by state** basis. If each region wants to query on information captured within its region, it would prove to be more effective to partition the fact table into regional partitions. This will cause the queries to speed up because it does not require to scan information that is not relevant.

### Points to Note

- The query does not have to scan irrelevant data which speeds up the query process.
- This technique is not appropriate where the dimensions are unlikely to change in future. So, it is worth determining that the dimension does not change in future.
- If the dimension changes, then the entire fact table would have to be repartitioned.

**Note:** We recommend to perform the partition only on the basis of time dimension, unless you are certain that the suggested dimension grouping will not change within the life of the data warehouse.

## Partition by Size of Table

When there are no clear basis for partitioning the fact table on any dimension, then we should **partition the fact table on the basis of their size.** We can set the predetermined size as a critical point. When the table exceeds the predetermined size, a new table partition is created.

### Points to Note

- This partitioning is complex to manage.
- It requires metadata to identify what data is stored in each partition.

## Partitioning Dimensions

If a dimension contains large number of entries, then it is required to partition the dimension. Here we have to check the size of a dimension.

Consider a large design that changes over time. If we need to store all the variations in order to apply comparisons, that dimension may be very large. This would definitely affect the response time.

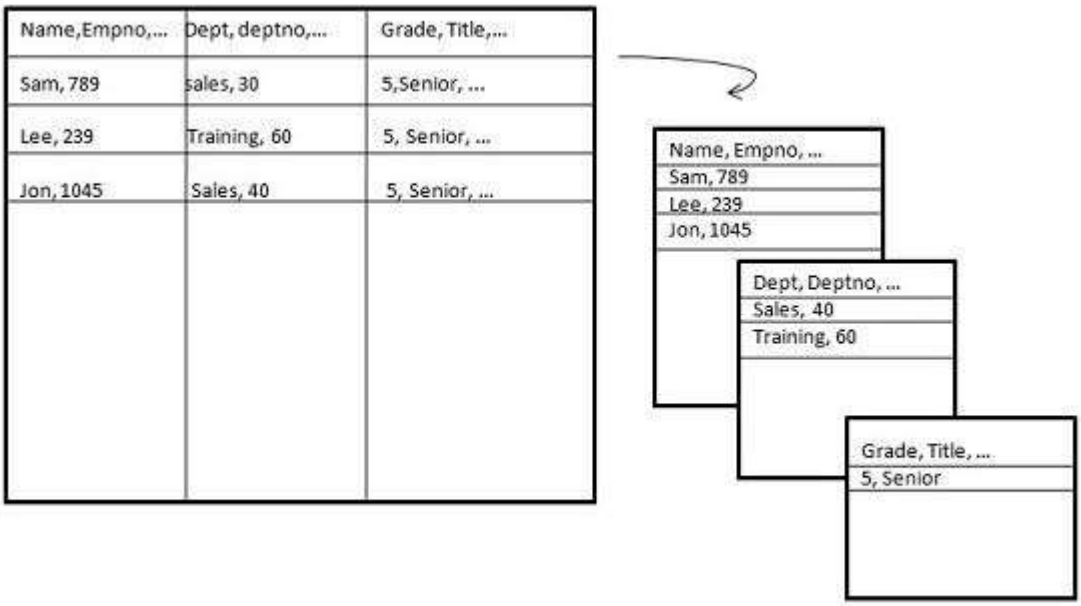
## Round Robin Partitions

In the round robin technique, when a new partition is needed, the old one is archived. It uses metadata to allow user access tool to refer to the correct table partition.

This technique makes it easy to automate table management facilities within the data warehouse.

## Vertical Partition

Vertical partitioning splits the data vertically. The following image depicts how vertical partitioning is done.




Vertical partitioning can be performed in the following two ways:

- Normalization
- Row Splitting

## Normalization

Normalization is the standard relational method of database organization. In this method, the rows are collapsed into a single row, hence it reduces space. Take a look at the following tables that show how normalization is performed.

### Table before Normalization

| Product_id | Qnty | Value | sales_date | Store_id | Store_name | Location | Region |
|---|---|---|---|---|---|---|---|
| 30 | 5 | 3.67 | 3-Aug-13 | 16 | sunny | Bangalore | S |
| 35 | 4 | 5.33 | 3-Sep-13 | 16 | sunny | Bangalore | S |
| 40 | 5 | 2.50 | 3-Sep-13 | 64 | san | Mumbai | W |
| 45 | 7 | 5.66 | 3-Sep-13 | 16 | sunny | Bangalore | S |

### Table after Normalization

| Store_id | Store_name | Location | Region |
|---|---|---|---|
| 16 | sunny | Bangalore | W |
| 64 | san | Mumbai | S |

| Product_id | Quantity | Value | sales_date | Store_id |
|---|---|---|---|---|
| 30 | 5 | 3.67 | 3-Aug-13 | 16 |
| 35 | 4 | 5.33 | 3-Sep-13 | 16 |
| 40 | 5 | 2.50 | 3-Sep-13 | 64 |
| 45 | 7 | 5.66 | 3-Sep-13 | 16 |

## Row Splitting

Row splitting tends to leave a one-to-one map between partitions. The motive of row splitting is to speed up the access to a large table by reducing its size.

**Note:** while using vertical partitioning, make sure that there is no requirement to perform a major join operation between two partitions.

## Identify Key to Partition

It is crucial to choose the right partition key. Choosing a wrong partition key will lead to reorganizing the fact table. Let's have an example. Suppose we want to partition the following table.

```
Account_Txn_Table
transaction_id
account_id
transaction_type
value
transaction_date
region
branch_name
```

We can choose to partition on any key. The two possible keys could be

- region
- transaction_date

Suppose the business is organized in 30 geographical regions and each region has different number of branches. That will give us 30 partitions, which is reasonable. This partitioning is good enough because our requirements capture has shown that a vast majority of queries are restricted to the user's own business region.

If we partition by transaction_date instead of region, then the latest transaction from every region will be in one partition. Now the user who wants to look at data within his own region has to query across multiple partitions.

Hence it is worth determining the right partitioning key.