Data Warehousing - Partitioning

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.

## **Horizontal Partitioning**

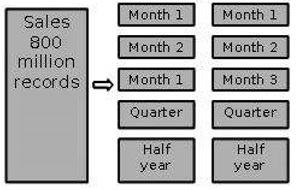
There are various ways in which a fact table can be partitioned. In horizontal partitioning, we have to keep in mind the requirements for manageability of the data warehouse.

### **Partitioning 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.



### **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.

* 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.

### **Partitioning Dimensions**

If a dimension contains large number of entries, then it is required to partition the dimensions. 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.

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 reduce space. Take a look at the following tables that show how normalization is performed.

Table before Normalization

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Product\_id** | **Qty** | **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**

**Indexing**

Indexing is used for the following reasons:

It saves cost and greatly improves performance and scalability.

It can replace a full-table scan by a quick read of the index followed by a read of only those disk blocks that contain the rows needed.

B-Tree Index

Most common type of indexing

Used for high-cardinality columns

Designed for few rows returned

Bitmap Indexes

Provide performance benefits and storage savings

Store values as 1s and 0s

Can be used instead of B-tree indexes when:

Tables are large

Columns have low cardinality

Row splitting tends to leave a one-to-one map between partitions. The motive of row splitting is to speed up the access to 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.