Benefits of partitioning:

-Partitioning allows you to physically separate data into smaller pieces that can be optimized for specific queries.

-Each partitioned data block can be indexed separately, reducing the time required to build or update indexes.

-Partitioning allows you to manage data storage more efficiently, as individual sections can be moved to different storage media depending on the frequency of access.

-Optimizing insert, update, and delete operations.

1. An explanation of the partitioning strategy:

Splitting the table by month (sale\_date) is the best strategy in cases where the data has strong temporal coherence and when queries are often targeted at specific time periods. This allows:

-Improve query performance.

-Make data management easier.

-Optimize storage.

1. Step-by-step documentation:

1. Defining the table structure and partitioning key:

-Create a main table specifying the partition type (PARTITION BY RANGE (sale\_date)).

2. Creating partitions:

-Define time boundaries for each partition (for example, months) and create corresponding partition tables with date ranges.

3. Data insertion:

-Data is automatically routed to the appropriate section based on the ‘sale\_date’ value when it is inserted.

4. Query optimization:

-Use predicates based on ‘sale\_date’ to ensure maximum performance by accessing only the relevant sections.

1. Maintenance strategy:

Approach and rationale:

Maintaining performance and manageability of large amounts of data requires regular partition maintenance, including the creation of new partitions and the removal of obsolete ones.

1. Service automation:

-Using the ‘maintenance\_task’ function, which automatically manages the life cycle of partitions: deletes the partition of the previous month and creates a new partition for the month ahead.

2. Maintenance schedule:

-A function can be scheduled to run monthly, for example on the first day of each month, which corresponds to the start and end of date ranges for partitions.

Planning example:

You can use a PostgreSQL task scheduler such as pg\_cron to automatically run the ‘maintenance\_task’ at a specific time.

‘SELECT cron.schedule('0 0 1 \* \*', $$ CALL maintenance\_task() $$);’

1. Personal reflection:

Errors in SQL syntax: Especially when generating dynamic SQL queries to create and delete partitions through the ‘maintenance\_task’ function. It was necessary to carefully check the correct formatting of strings and dates.