DWH Indexes & Partitions

Partitioning

Partitioning helps to scale a data warehouse by dividing database objects into smaller pieces, enabling access to smaller, more manageable objects. Having direct access to smaller objects addresses the scalability requirements of data warehouses.

> *Following advantages of partitioned objects:*

- Faster queries, as users are more likely to query on isolated and smaller data sets. Data contention is less likely.
- Backup and recovery can be performed on a low level of granularity to manage the size of the database.
- Part of a database object can be placed in compressed storage while other parts can remain uncompressed.
- You can transfer or access subsets of data quickly and efficiently, while maintaining the integrity of a data collection. For example, an operation such as loading data from an OLTP to an OLAP system.

Partitions' Types And Strategies:

Horizontal Partitioning

In this technique, the dataset is divided based on rows or records. Each partition contains a subset of rows, and the partitions are typically distributed across multiple servers or storage devices.

Vertical Partition

Unlike horizontal partitioning, vertical partitioning divides the dataset based on columns or attributes. In this technique, each partition contains a subset of columns for each row. Vertical partitioning is useful when different columns have varying access patterns or when some columns are more frequently accessed than others.

Key-based Partitioning

Using this method, the data is divided based on a particular key or attribute value. The dataset has been partitioned, with each containing all the data related to a specific key value. Key-based partitioning is commonly used in distributed databases or systems to distribute the data evenly and allow efficient data retrieval based on key lookups.

Range Partitioning

Range partitioning divides the dataset according to a predetermined range of values. You can divide data based on a particular time range, for instance, if your dataset contains timestamps. When you want to distribute data evenly based on the range of values and have data with natural ordering, range partitioning can be helpful.

Hash-based Partitioning

Hash partitioning is the process of analyzing the data using a hash function to decide which division it belongs to. The data is fed into the hash function, which produces a hash value used to categorize the data into a certain division. By randomly distributing data among partitions, hash-based partitioning can help with load balancing and quick data retrieval.

Round-robin Partitioning

In round-robin partitioning, data is evenly distributed across partitions in a cyclic manner. Each partition is assigned the next available data item sequentially, regardless of the data's characteristics. Round-robin partitioning is straightforward to implement and can provide a basic level of load balancing.

Partitioning Technique	Description	Suitable Data	Query Performance	Data Distribution	Complexity
Horizontal Partitioning	Divides dataset based on rows/records	Large datasets	Complex joins	Uneven distribution	Distributed transaction management
Vertical Partitioning	Divides dataset based on columns/attributes	Wide tables	Improved retrieval	Efficient storage	Increased query complexity
Key-based Partitioning	Divides dataset based on specific key	Key-value datasets	Efficient key lookups	Even distribution by key	Limited query flexibility
Range Partitioning	Divides dataset based on specific range	Ordered datasets	Efficient range queries	Even distribution by range	Joins and range queries
Hash-based Partitioning	Divides dataset based on hash function	Unordered datasets	Even distribution	Random distribution	Inefficient key-based queries
Round-robin Partitioning	Divides dataset in a cyclic manner	Equal-sized datasets	Basic load balancing	Even distribution	Limited query optimization

Indexing in Data Warehouses

1. Bitmap Index: These use bit arrays (or vectors) to represent the membership of elements in sets. They're extremely space-efficient when the domain (the number of distinct values) of the indexed attribute is low. For example, a column like "Gender" with values "Male" and "Female" can benefit from a bitmap index.

2. B-tree Index: This is one of the most common forms of indexing. B-tree indices are hierarchical, with a tree-like structure, and are most effective for high cardinality attributes or when the distribution of values is unpredictable.

3. Join Index: By pre-computing the result of joins between tables and storing it, join indices can drastically speed up join operations. They can be particularly useful in star or snowflake schema scenarios in data warehouses where joins between fact tables and dimension tables are frequent.

4. Materialized Views: These are precomputed result sets that are stored in the database, which can be based on multiple base tables and may contain aggregate data. They are updated periodically. While they do occupy storage space, they can significantly speed up complex queries.

5. Partitioning: This is more of a physical data organization strategy than an indexing method per se. By storing data in separate, smaller chunks, databases can improve query performance. For instance, data can be partitioned by year, allowing a query that only concerns data from one particular year to scan only one partition rather than the entire table.

References:

https://docs.oracle.com/en/database/oracle/oracle-database/19/vldbg/partition-warehouse.html#GUID-CD9A8484-D111-461B-8883-04F520A30CE4

https://www.tutorialspoint.com/dwh/dwh_partitioning_strategy.htm

https://www.geeksforgeeks.org/data-partitioning-techniques/

https://medium.com/@Ric_Royal/understanding-data-warehouses-exploring-data-structures-and-indexing-methods-f713f783f78a

https://www.youtube.com/watch?v=1fETPYKyb70