DexterityDB Automatic Partitioning

Background

Database partitioning is a useful way to improve performance and availability, while also making a database easier to maintain. The problem is that partitioning can be complex and/or tedious. To make this process simpler, it is common for databases to partition by row or table by default.

Moreover, the locking characteristics of a database can rely on the partitioning schema used. If partitioning is done at the table level, then there will be fewer locks executed, which will result in speed benefits. However, this setup does not allow for parallel processes within each table. For instance, even in a 10,000 row table, if one row is being updated or added, there are plenty of other rows that can be updated, or read at the same time without impacting the results of the first query. So, to achieve parallel processing capabilities, many storage engines have row level locks, reserving only the rows that are required to handle a particular query. This is useful to distribute queries across CPU cores, but it also requires a lot of locking. These locks take considerable time and can cause significant delays in query execution.

Multithreading with Chunks

Dexterity's partitioning system intelligently analyzes many different parameters, including data characteristics and hardware specifications, to automatically partition database tables in order to minimize locks and maximize performance by scaling and multithreading queries. These partitions are known as chunks.

This kind of partitioning makes it possible for DexterityDB to achieve virtually linear vertical scalability. For servers with a higher core/thread count, automatic partitioning will ensure that data is chunked in a way that allows threads to be allocated as efficiently as possible for increased speed benefits.

When DexterityDB's automatic partitioning system is paired with its unique operation engine, the system is completely lockless. Queries are broken down into smaller operations and operations are scheduled on the chunks with relevant data. Because only one operation is happening on a chunk at a time, there are no locks and no risk of having two operations

execute on the same chunk at the same time. DexterityDB's resource manager ensures the integrity of the task queue, making sure that proper order of queries is achieved even when operations are performed across chunks.

This combination allows DexterityDB to use multiple threads to perform a single query. Examples of this are:

- Multi-threaded read - queries are analyzed and broken down into smaller operations so that they can be threaded and assigned to the predefined chunks that contain relevant data. This allows tasks to be done in parallel, resulting in massively scalable read queries and analytics.
- Multi-threaded bulk insert - data is inserted across automatically predefined chunks, allowing for lockless, parallel writes. Data access does not overlap between threads, so there is no need to waste time with locking. Reorganization of the data in the partitions will then happen in the background to optimize for read queries. This allows for massively scalable writes/inserts.
- Multi-threaded update - when data from different chunks needs to be updated, the query is broken down and operations are assigned to the relevant chunks. The threads operate on the chunks at the same time allowing data to be updated in parallel.
- Multi-threaded delete - Similar to inserts and updates, deletes can happen in parallel if the data resides in different chunks.

Find Out More

To learn more, contact us at contact@dexteritydb.com