DexterityDB Indexing System

Traditional indexing systems are designed to categorize data and make it easier to find. DexterityDB has reinvented the index with its revolutionary patent-pending design. This new indexing system incorporates new techniques to provide faster computation as well as reduced storage space.

Unique Data Structures

DexterityDB indexes utilize a new, revolutionary hybrid data structure that is based on B+ Trees, a common data structure in traditional databases. In other B+ Tree indexing systems, the number of nodes is equal to the number of rows in the database. Each node contains a row's value for the indexed column and a reference to the row, itself. Many rows can share the same value for the given field, resulting in the same key being duplicated in many rows.

DexterityDB indexes remove these duplications by grouping rows that have the same value for the given index. This design has the following benefits:

- Reduces tree size and saves space
- Eliminates unnecessary key checks, allowing for quicker index lookup.

Fetchless Operations

Statistics and Analytics

For many databases, indexes are used to locate raw data, but other processes follow to further filter and/or analyze the resulting data to satisfy queries. DexterityDB is smart. It pushes off all fetching operations (ones that require the retrieval of raw data) for as long as it can, sometimes eliminating the fetching stage altogether and dealing only with indexes to satisfy a given query. For many statistical and analytical queries, such as counts, all of the required information is already present in these indexes and there is no reason to fetch the raw data at all to compute these results. To include this additional information without taking up additional storage, DexterityDB indexes inherently contain characteristics about the raw data without directly storing it.

For example, given the following query:

SELECT AVG(age), MIN(name) FROM employees;

To complete this query, traditional indexing systems would fetch each row and iterate through them, looking for the first name alphabetically (comparing name values on each iteration). As it does this, it would also be keeping a running sum of each employee's age. Then it would calculate the average based on how many rows it went through (in this case, all of them). This would require a lot of raw data analysis. In contrast, DexterityDB would only need to look at the indexes for the "name" and "age" fields. For MIN(name), only the keys for the "name" index would need to be checked to find the first one alphabetically. For AVG(age), these indexes already have inherent information for each key, which it can use to calculate the average age of the employees. These approaches have the following benefits:

- Fewer checks and calculations are necessary because duplicate keys do not exist.
- If the result of a statistical query can be calculated with only indexes (which are considerably smaller than the raw data it references), there is no need to fetch any of the raw data from the disk, eliminating an entire set of disk I/O operations that would otherwise be necessary. I/O operations are often a bottleneck in traditional database systems. If datasets are large, raw data may need to be loaded into RAM in chunks which can lead to sluggish performance.
- RAM usage is minimized because less data is used to satisfy a query.

Set Operations

This technology can also be used to accelerate set operations, such as intersections and unions. Many traditional indexing systems only take advantage of a single index before fetching raw data and doing additional filtering to find a result of an AND or OR operation. Some systems utilize index merging techniques, but DexterityDB is unique in that it can use many indexes to satisfy a single query. This advanced form of index merging can be used to satisfy many different kinds of queries. Value lists in these indexes' key-value pairs are structured in such a way that they can be quickly and directly compared to find similarities and/or differences between them without needing information from the actual rows. This allows the operation engine to wait and only fetch the results of these comparisons at the very end of the query. This approach has the following benefits:

- DexterityDB indexes can be directly compared without needing the actual row data. Moreover, no parsing is necessary because the values in the index are just simple, uniform numbers, not a complex structure like what would be used to represent a row. This system eliminates raw data comparisons, which are slow, because raw data needs to be fetched from disk and then parsed to find the relevant data within the row.
- RAM usage is minimized because less data is used to satisfy a query.

Find Out More

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