Database Indexes: Comprehensive Review

DSC 208R - Data Management for Analytics

Motivation for Indexing

Consider typical SQL queries that retrieve records based on specific conditions, such as 'SELECT * FROM Movies WHERE Year=2017' or 'SELECT * FROM Movies WHERE Year¿=2000 AND Year¡2010'.

- The naive option to obtain matching records is a full filescan of the table, applying the predicate to each tuple.
- This approach has an $O(N)$ cost for both I/O (input/output) and CPU, where N is the number of tuples in the table.

Overview of Indexes

An index is an auxiliary data structure in a database that helps locate and retrieve records much faster than an $O(N)$ filescan.

- **IndexKey (SearchKey)**: Refers to the attribute(s) on which a file is indexed. It can be any permutation of any subset of a relation's attributes and does not necessarily have to be a primary or candidate key.
- RDBMSs support indexes designed and optimized for search, insert, and delete operations in SQL.
- In practice, queries can become millions of times faster when suitable indexes are available on a table.

Comparison: Indexed Access vs. Filescan

- **Filescan**: Reads every page of the table. This is necessary for queries like 'SELECT COUNT(*) FROM Movies' or 'SELECT * FROM Movies WHERE Name LIKE '
- **Indexed Access**: Reads only the relevant pages of the table via the index. This is beneficial for queries that involve 'WHERE' conditions on the indexed attribute(s).

Types of Indexes

1. B+Tree Indexes

- B+Trees are multi-level indexes, forming a balanced tree structure.
- They are the most common type of index in RDBMSs due to their efficiency in handling range queries.
- Each node in a B+Tree is typically a disk block.
- All data records are stored at the leaf level, which are linked lists.
- B+Trees support efficient retrieval for equality searches, range searches, and ordered retrieval.

Example SQL for B+Tree Index Creation:

```
CREATE INDEX MIn1 ON Movies USING btree (Year);
```

This index is useful for 'WHERE year = 2017' and 'WHERE year ¿= 2000 AND year ¡ 2010'.

Composite B+Tree Indexes: If a B+Tree index has a composite 'IndexKey' (multiple attributes), the order of attributes matters; it is a sequence.

```
CREATE INDEX MIn1 ON Movies USING btree (Name, Year);
CREATE INDEX MIn2 ON Movies USING btree (Year, Name);
```

- An index on ‘(Year, Name)‘ (‘MIn2‘) is useful for queries like ‘SELECT * FROM Movies WHERE Year ¿= 2015‘ because of the prefix match on ‘Year‘.
- An index on ‘(Name, Year)‘ (‘MIn1‘) is generally useless for a query like ‘SELECT * FROM Movies WHERE Year ¿= 2015‘ because the ‘Year‘ attribute is not a prefix of the ‘IndexKey‘.

2. Hash Indexes

- Hash indexes are single-level indexes that use a hash function to map ‘IndexKey‘ values to data record locations.
- They are generally efficient for equality searches (‘=‘).
- They are NOT efficient for range queries (‘¿‘, ‘¡‘, ‘BETWEEN‘) or ‘LIKE‘ pattern matching with wildcards at the beginning.

Example SQL for Hash Index Creation:

```
CREATE INDEX MIn2 ON Movies USING hash (Name);
```

This index is useful for ‘WHERE name = ’The Lion King’‘ but not ‘WHERE name LIKE ’

Composite Hash Indexes: If a hash index has a composite ‘IndexKey‘, the order among attributes does not matter; it is a set.

When to Use Indexes

Indexes are beneficial for:

- Attributes frequently used in ‘WHERE‘ clause predicates.
- Attributes involved in join conditions.
- Attributes used for ‘GROUP BY‘ or ‘ORDER BY‘ clauses.
- Foreign key columns, as they are often used in join conditions.
- When the table is large and the queries retrieve a small percentage of its records.

Indexing for DML Operations

While indexes speed up reads, they slow down writes (INSERT, UPDATE, DELETE).

- Each DML operation on a table also requires corresponding updates to its indexes.
- Therefore, ‘CREATE INDEX‘ statements are usually used only when the expected performance gain for reads outweighs the performance loss for writes.

Index Selection

Deciding what indexes to build on a given database is a crucial task, often part of a Database Administrator’s (DBA) job.

- This is a formalized problem called ”index selection,” with extensive research and semi-automated tuning products.
- Index selection is typically decided based on query workload patterns.