Database Search

Databases (tabular)

- Tables with many columns
- Want to search quickly on some columns
- Maintain “INDEX” of columns to search on
  - Store a sorted version of column
  - Needs column to be “comparable”: integer, short string, date/time etc.
    - Long text fields are not good for index
    - Binary data not good

Example: MySQL

https://dev.mysql.com/doc/refman/8.0/en/index-btree-hash.html

8.3.9 Comparison of B-Tree and Hash Indexes

Understanding the B-tree and hash data structures can help predict how different queries perform on different storage engines that use these data structures in their indexes, particularly for the `MEMORY` storage engine that lets you choose B-tree or hash indexes.

- B-Tree Index Characteristics
- Hash Index Characteristics

Index-friendly query

```
SELECT * FROM tbl_name WHERE key_col LIKE 'Patrick%';
```

```
SELECT * FROM tbl_name WHERE key_col LIKE 'Pat%_ck%';
```

Index-**un**friendly query

```
SELECT * FROM tbl_name WHERE key_col LIKE '%Patrick%';
```

```
SELECT * FROM tbl_name WHERE key_col LIKE other_col;
```

Multi-column index

- (index_1, index_2, index_3): compound index on 3 columns:
  - first sorted on index_1, then on index_2, then on index_3
  - all values with same index_1 will be sorted on index_2,
  - all values with same index_1 and index_2 will be sorted on index_3
  - etc.
- eg. (date-of-birth, city-of-birth, name)
  - can query for all people born on same date in same city with same name easily
  - but...

Multi-index friendly

```
... WHERE index_part1=1 AND index_part2=2 AND other_column=3

    /* index = 1 OR index = 2 */
... WHERE index=1 OR A=10 AND index=2

    /* optimized like "index part1='hello'" */
... WHERE index_part1='hello' AND index_part3=5

    /* use index on index1 but not on index2 or index3 */
... WHERE index1=1 AND index2=2 OR index1=3 AND index3=3;
```

Multi-index **un**friendly

```
   /* index part1 is not used */
... WHERE index_part2=1 AND index_part3=2
```

```
    /*  Index not used in both parts of the WHERE clause  */
... WHERE index=1 OR A=10
```

```
    /* No index spans all rows  */
... WHERE index_part1=1 OR index_part2=10
```

Hash-index

- Only used in in-memory tables
- Only for equality comparisons - cannot handle “range”
- Does not help with “ORDER BY”
- Partial key prefix cannot be used
- But VERY fast where applicable...

Query Optimization

Database specific

- https://dev.mysql.com/doc/refman/8.0/en/index-btree-hash.html
- https://www.sqlite.org/optoverview.html
- Postgres:

Chapter 59. Genetic Query Optimizer

Table of Contents

59.1. Query Handling as a Complex Optimization Problem
59.2. Genetic Algorithms
59.3. Genetic Query Optimization (GEQO) in PostgreSQL
    59.3.1. Generating Possible Plans with GEQO
    59.3.2. Future Implementation Tasks for PostgreSQL GEQO
59.4. Further Reading

Summary

- Setting up queries properly impacts application performance
- Building proper indexes crucial to good search
- Compound indexes, multiple indexes etc. possible
  - Too many can be waste of space
- Make use of structure in data to organize it properly