Sql-Bitmap Index

Here's a summary of BitMap indexes, along with an example in SQL:

What is a BitMap index?

A BitMap index, also known as a Bit Vector or Binary Tree index, is a type of index that uses a compact binary format to store a bit map of the data. It's designed for columns with low-cardinality (i.e., most values are rare), such as boolean flags, category codes, or state IDs.

How does it work?

In a BitMap index:

1. A fixed-length bit map is created, which has one bit per row in the table.
2. When data is inserted, the corresponding bits in the bit map are set to 1 if the value is present; otherwise, they're set to 0.
3. The query processor uses the bit map to quickly identify rows that match the search criteria.

Advantages:

1. **Space-efficient**: BitMap indexes use significantly less storage than traditional indexes (e.g., B-tree or clustered).
2. **Fast queries**: Queries that involve boolean predicates (e.g., `WHERE is_active = 1`) can be executed quickly using the bit map.
3. **Improved data locality**: By storing related bits together, the bit map promotes better cache usage and reduced page I/O.

Example in SQL:

Let's say we have a table called `orders` with columns `id`, `customer_id`, and `is_active`. We want to create an index on the `is_active` column using a BitMap index.

```
 CREATE TABLE orders (
  id INT PRIMARY KEY,
  customer_id INT,
  is_active BIT(1)
);

CREATE INDEX idx_is_active ON orders (is_active) USING BITMAP;
```

Assuming we have some sample data:

```
 +----+--------------+-----------+
| id | customer_id | is_active |
+----+--------------+-----------+
| 1  |       101   |         0 |
| 2  |       102   |         1 |
| 3  |       103   |         0 |
| 4  |       104   |         1 |
+----+--------------+-----------+
```

If we execute a query like `SELECT * FROM orders WHERE is_active = 1`, the BitMap index would quickly locate rows with `is_active` set to 1 (rows 2 and 4), allowing for efficient retrieval of those records.

Keep in mind that BitMap indexes are most useful when:

- The column has low-cardinality values (e.g., boolean flags, category codes).
- Queries frequently filter on the indexed column.
- Space is a concern, and storage efficiency is necessary.

Curated by Brajesh Kumar