The PostgreSQL Btree Index

INFO-H417: Lab Session 5

2023-2024

PostgreSQL allows the creation of indexes, which are special database objects used to speed up data access. This exercise session will focus on the *Btree* index. **Read the blog *Indexes in PostgreSQL — 4 (Btree)*[1] to answer the following questions about Btree indexes in PostgreSQL.** The blog uses a demo database that can be found here[2].

1. What data is stored in the elements of the leaf nodes? value of index and pointer to the row
2. What data is stored in the elements of the internal and root nodes? minimum value of page
3. What is required for a type to be indexed using a Btree index? That is: if we define a new type and want to be able to create a Btree index on it, what operators also need to be defined for this type? Why is this necessary?
4. Does the Btree store the TIDs of all the rows of the indexed column (like a dense index) or only a subset of the rows (like a sparse index)?
5. What properties of the Btree allows us to answer inequality ($<, <=, >, >=$) or range searches (*between*) efficiently?
6. Let's assume that an index contains 3 levels, each containing respectively 1 (root), 10 and 1000 nodes/pages. On average, how many index pages will be fetched when answering a query containing a search by equality? How many will that be when the query contains a search by inequality?
7. How can a Btree index be used to answer a query of the form: `SELECT ... FROM ... ORDER BY col`? Could we use the same index for a query of the form: `SELECT ... FROM ... ORDER BY col DESC`?
8. What is an *index only scan* and when is it used? Give an example of a query where an index only scan would be used.

[1] https://postgrespro.com/blog/pgsql/4161516
[2] https://postgrespro.com/docs/postgrespro/9.6/demodb-bookings

9. Let's assume we have a table `T(A, B)` and we create two multi-column indexes on it.

   ```
   CREATE INDEX idx_ab ON T(A, B);
   CREATE INDEX idx_ba ON T(B, A);
   ```

   Which of the two indexes will be used to answer the following queries, and why?

   ```
   SELECT A, B FROM T WHERE A = 'a';
   ```
   first

   ```
   SELECT A, B FROM T WHERE A = 'a' AND B = 'b';
   ```
   both

   ```
   SELECT A, B FROM T WHERE B = 'b';
   ```
   second

10. How are `NULL` values handled by Btree indexes?

    we can specify either null first or null last
    by default null last