An Introduction to Basic Data Structures in Postgres 16

Leo Custodio
BDS Research Inc.
contact@lcustodio.com

Abstract

This article provides a technical overview of two fundamental data structures in PostgreSQL: the heap (base table storage) and the B-Tree index. It explains how the heap stores row data on disk and how it interacts with PostgreSQL's Multi-Version Concurrency Control (MVCC) mechanism. Additionally, the article delves into the structure, usage, and performance benefits of B-Tree indexes, illustrating how they enable efficient data retrieval. By understanding these data structures, readers can optimize queries, choose appropriate indexing strategies, and maintain better database performance.

Keywords

PostgreSQL Heap Storage, B-Tree Indexing, Database Performance

Introduction

PostgreSQL is a powerful, open-source relational database system known for its robust feature set, extensibility, and standards compliance. Under the hood, PostgreSQL relies on well-established data structures to store data and make query execution performant. Two particularly important structures are:

- The Heap (Table Storage): This is the base structure where all table data is stored.
- B-Tree Indexes: These are the most commonly used indexes in PostgreSQL, enabling efficient data retrieval operations.

Understanding these data structures is crucial for DBAs, developers, and anyone who seeks to optimize performance, understand query execution plans, or troubleshoot database operations.

1. Heap Storage in PostgreSQL

What is the Heap?

PostgreSQL stores table data in what is commonly referred to as a "heap." Contrary to the name, it's not a heap in the strict algorithmic sense (like a binary heap used in priority queues), but rather an unordered collection of rows. The heap consists of multiple files on disk, each divided into fixed-size pages (commonly 8KB by default). Each page can store multiple rows or "tuples," along with some metadata such as transaction identifiers for concurrency control and tuple visibility information.

How are Rows Organized?

In a heap, rows are inserted wherever there is available space. PostgreSQL does not reorder rows on disk after insertion. As a result, data retrieval without an index scan will read pages sequentially in the order they appear on disk, irrespective of logical ordering such as primary keys. Over time, deletions and updates (which in PostgreSQL are effectively insertions of new versions of a row and marking old versions as expired) can create fragmentation, leading to less efficient sequential reads.

Tuple Visibility and MVCC:

PostgreSQL employs Multi-Version Concurrency Control (MVCC) to manage concurrent transactions. Each tuple in the heap contains information that helps the database engine decide if that tuple is visible to a particular transaction. With MVCC, readers do not block writers and vice versa. This design ensures smooth concurrent operations but also means that over time, some tuples become "dead" (no longer visible to any transaction) and must be reclaimed via Vacuum processes.

Example – Selecting Data from a Heap:

Suppose we have a simple table:

```
CREATE TABLE employees (
    id SERIAL PRIMARY KEY,
    name TEXT,
    role TEXT
);
```

When we insert data:

```
INSERT INTO employees (name, role) VALUES
('Alice', 'Engineer'), ('Bob', 'Manager');
```

These rows end up appended to the heap. A SELECT * FROM employees; without any index usage will trigger a sequential scan of the heap, reading every page of the underlying file until all tuples are retrieved.

2. B-Tree Indexes in PostgreSQL

What is a B-Tree Index?

A B-Tree index is a balanced tree data structure used to quickly locate rows in large data sets. By maintaining a tree of keys, B-Tree indexes reduce the need to scan entire heaps, thus improving query performance dramatically. When you create a primary key or a unique constraint in PostgreSQL, a B-Tree index is typically built under the hood.

Structure of a B-Tree:

The B-Tree is a multi-level structure composed of a root node, internal nodes, and leaf nodes. The leaf nodes contain pointers to the actual heap tuples (or their identifiers, known as TIDs—Tuple Identifiers). The internal nodes serve as a navigation layer, allowing PostgreSQL to quickly descend through the tree to find the correct leaf. B-Trees are kept balanced, meaning the tree's height remains relatively small even as data grows large, ensuring O(log n) search time complexity.

How PostgreSQL Uses B-Trees:

When you execute a query with a condition that can use an index (for example, `SELECT * FROM employees WHERE id = 10;`), PostgreSQL's optimizer checks if a B-Tree index on id exists. If it does, the engine performs an index scan rather than a sequential scan. This process involves:

Starting from the root of the B-Tree.

Traversing down internal nodes following the comparisons until the correct leaf is found.

Fetching the TID from the leaf node and then retrieving the row directly from the heap.

Example – Creating and Using a B-Tree Index:

```
-- Creating an index on the 'role' column
CREATE INDEX idx_employees_role ON employees
(role);
-- Now a query filtering by role can use the index
EXPLAIN ANALYZE SELECT * FROM employees WHERE
role = 'Engineer';
```

In the EXPLAIN ANALYZE output, you'll see something like an "Index Scan" instead of "Seq Scan," indicating that PostgreSQL used the B-Tree index to efficiently locate rows with `role = 'Engineer'`.

Conclusion

The heap and B-Tree indexes are two essential building blocks of PostgreSQL's storage and retrieval strategy. The heap provides a flexible, append-friendly structure for storing row data, while B-Tree indexes deliver fast access to subsets of that data. By understanding how these data structures work, you can make informed decisions about when to use indexes, how to structure queries, and how to maintain efficient database performance over time.

References

Geschwinde, E., & Schönig, H. J. (2002). *PostgreSQL Developer's Handbook.* Sams Publishing.

Momjian, B. (2000). PostgreSQL: *Introduction and Concepts.* Addison-Wesley.

PostgreSQL Global Development Group. (2023). *PostgreSQL 16 Documentation.* https://www.postgresql.org/docs/16/

Riggs, S., Ciolli, G., Krosing, H., & Bartolini, G. (2015). *PostgreSQL 9 Administration Cookbook.* Packt Publishing Ltd.