

CSE 512: Distributed Database Systems

Project: Distributed Database System for a Smart Building

Part – 3

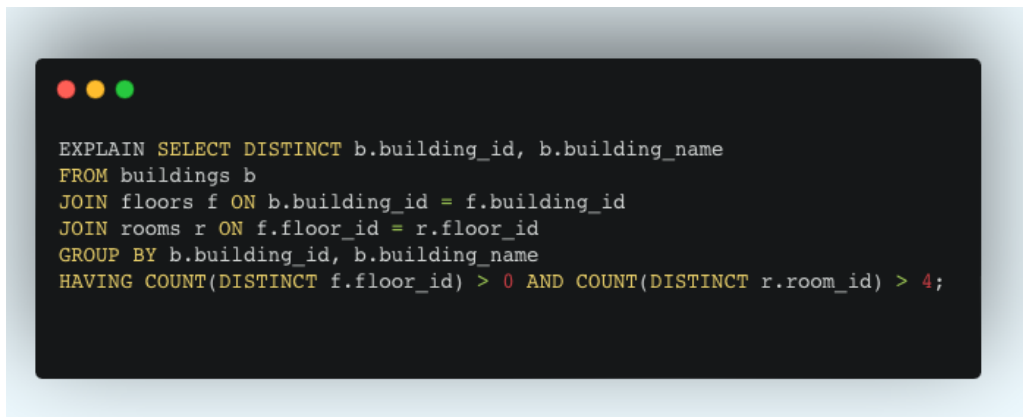
Query Optimization using Indexing in PostgreSQL:

Indexing is a database optimization technique that involves creating data structures to improve the speed of data retrieval operations on a database table. It is primarily aimed at enhancing the speed of data retrieval operations, such as SELECT queries. Without indexes, a database management system (DBMS) may need to perform a full table scan, checking every row to find the desired data. This can be inefficient for large datasets.

By default, PostgreSQL performs B-Tree indexing. They are the most used indexing technique. They are well-suited for various data types and provide efficient search, insertion, and deletion operations.

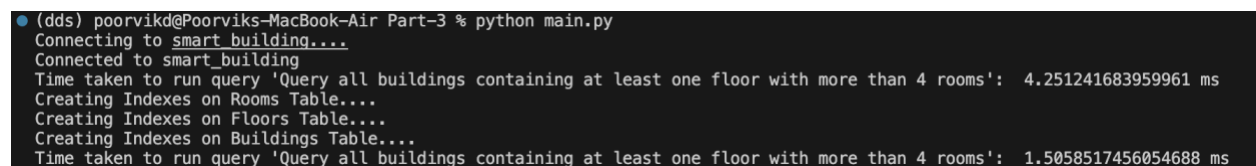
Well-designed indexes can significantly improve the overall performance of database applications. To showcase the impact of an index, we run the query to find all buildings containing at least one floor with more than four rooms with and without indexes on the buildings, floors, and rooms tables.

We indexed buildings table on building_id, floors table on (floor_id, building_id), and rooms table on (room_id, floor_id)

A screenshot of a PostgreSQL terminal window with a dark background and light-colored text. The window has three colored window control buttons (red, yellow, green) in the top-left corner. The SQL query displayed is:

```
EXPLAIN SELECT DISTINCT b.building_id, b.building_name
FROM buildings b
JOIN floors f ON b.building_id = f.building_id
JOIN rooms r ON f.floor_id = r.floor_id
GROUP BY b.building_id, b.building_name
HAVING COUNT(DISTINCT f.floor_id) > 0 AND COUNT(DISTINCT r.room_id) > 4;
```

The screenshot below is a clear indication of the improved query processing by PostgreSQL.

A screenshot of a terminal window with a dark background and light-colored text. The text shows the execution of a Python script that connects to a PostgreSQL database, runs a query, and reports the time taken. The output is:

```
(dds) poorvikd@Poorviks-MacBook-Air Part-3 % python main.py
Connecting to smart_building....
Connected to smart_building
Time taken to run query 'Query all buildings containing at least one floor with more than 4 rooms': 4.251241683959961 ms
Creating Indexes on Rooms Table....
Creating Indexes on Floors Table....
Creating Indexes on Buildings Table....
Time taken to run query 'Query all buildings containing at least one floor with more than 4 rooms': 1.5058517456054688 ms
```

Query Optimization using Partitioning & Sharding:

PostgreSQL, a powerful open-source relational database, offers advanced features like sharding and partitioning to optimize query performance in large-scale deployments. Sharding and partitioning are techniques that enhance the scalability and efficiency of a database system.

Sharding:

Definition: Sharding involves horizontally partitioning a database by distributing its data across multiple nodes or servers. Each shard contains a subset of the overall dataset, and collectively, the shards form a distributed database system.

Benefits for Query Optimization:

1. **Parallel Query Execution:** Sharding allows queries to be executed in parallel across multiple shards. This parallelism significantly improves the performance of read-heavy operations, as each shard can process its subset of data independently.
2. **Load Distribution:** By distributing data across shards, the overall database load is distributed, preventing hotspots, and ensuring that no single node becomes a bottleneck. This is crucial for balancing the system's workload and maintaining responsiveness.
3. **Scalability:** Sharding provides horizontal scalability, enabling the addition of new nodes to accommodate growing data volumes and increasing query loads. This makes it possible to handle larger datasets and higher query throughputs.

Partitioning:

Definition: Partitioning involves dividing large tables into smaller, more manageable pieces called partitions based on a specified column (e.g., date, range, or hash).

Benefits for Query Optimization:

1. **Elimination of Unnecessary Data:** Partitioning allows the database engine to skip irrelevant partitions when executing queries. This "pruning" of unnecessary data can significantly reduce the amount of data scanned, leading to faster query performance.
2. **Improved Query Planning:** Partitioning provides the query planner with more information about the data distribution. This knowledge allows for better optimization of execution plans, resulting in more efficient query processing.
3. **Faster Data Loading and Deletion:** When loading or deleting data, partitioning can expedite these operations as they only affect a specific partition rather than the entire table. This is particularly beneficial for time-series data or other scenarios where data is naturally segmented.

We implement partitioning of tables in Part-2.

Compiled By:

Distributed Nerds

Ahraz Rizvi

Keshava Rajavaram

Poorvik Dharmendra

Sahara Abdi