Paseo Posse – Hyperlocal Delivery System

Query optimization, distributed indexing, and the use of stored procedures are crucial aspects of improving the performance and efficiency of a database system, especially in a distributed environment like the hyperlocal delivery project you described.

**Query Optimization:**

**Partitioning for Pruning:**

Partitioning tables based on regions (ZIP codes) enables data pruning, where the database engine only accesses the partitions relevant to the specific query. This reduces the amount of data scanned and improves query performance.

**Use of Skip Lock Command:**

The SKIP LOCKED command in our SELECT queries helps avoid contention and efficiently handles concurrent transactions. It allows transactions to skip over rows that are already locked by other transactions, reducing waiting times and improving overall system responsiveness.

**Use of UUID in Inventory Table:**

The use of UUIDs as primary keys in the Inventory table, along with the distributed structure, ensures unique identifiers across partitions. This can enhance distributed indexing efficiency.

**Stored Procedures:**

Avoiding N+1 Queries:

N+1 query problems occur when a script or application issues N+1 separate queries to fetch related data. Using stored procedures consolidates these queries into a single stored procedure call, reducing the number of round trips between the application and the database.

Avoiding Multiple Calls from Python Script:

By encapsulating complex logic and queries in stored procedures, you reduce the need for multiple calls from the Python script. This can result in better performance and code maintainability.

**Distributed Indexing on Partitioned Tables:**

Distributed indexing involves creating and managing indexes across multiple nodes or partitions in a distributed database. This approach is especially beneficial in scenarios where data is horizontally partitioned, as is the case in our hyperlocal delivery project where tables are partitioned based on regions (ZIP codes). Below we have explained how distributed indexing works and its impact on the Inventory and Orders tables:

**Inventory Table:**

How Indexing Improves Select Queries:

Scenario:

The Inventory table contains information about the availability of various items in different regions. Given the distributed nature of the system, it's essential to have efficient indexes on columns commonly used in SELECT queries.

Impact:

When a SELECT query is executed for a specific med\_id and zip\_code, the database engine can utilize the index on the relevant partition, significantly reducing the amount of data that needs to be scanned. This results in faster query performance.

**Orders Table:**

Considerations for Orders Table:

Insertion Frequency:

In the Orders table, where insertion is more frequent than select, indexing strategies need careful consideration. Indexing comes with an overhead during insertion, as indexes need to be updated each time a new record is added.

Trade-Offs:

While indexing can significantly improve SELECT query performance, it might impact the speed of insert operations, especially in scenarios with high insertion rates. The trade-off between read and write performance should be considered.