Paseo Posse – Hyperlocal Delivery System

The selected domain for the project is a hyperlocal delivery system for essential groceries and medications. This system is crucial in today's world where quick, efficient, and localized delivery services are in high demand.

A distributed database is justified in this scenario due to the geographical dispersion of data and operations. It allows for data localization, which can improve performance, reliability, and availability. It also enables scalability as the service expands to new regions.

The distributed structure of this solution involves partitioning or sharding the database based on regions. Each region has its own inventory stock and assigned delivery agents. This structure allows for efficient query optimization and handling of concurrency and atomicity in the distributed database. It ensures reducing the load on the system and improving overall performance.

**Entity-relationship diagrams and table definitions**

**Postgres Database tables**

**1) ORDER\_ITEM Table**

This table represents the items included in each order. It includes information such as the order ID, medication ID, quantity, and delivery ZIP code.

Similar to the ORDER table, this table is also partitioned by the ZIPCODE field.

**Table schema:**

*ORDER\_ITEM(*

*ORDER\_ID VARCHAR(10),*

*MED\_ID INT NOT NULL,*

*QUANTITY INT NOT NULL,*

*ZIPCODE VARCHAR(5) NOT NULL,*

*PRIMARY KEY(ORDER\_ID)*

*) PARTITION BY LIST (ZIPCODE);*

**2) ORDER Table**

This table represents orders placed by customers. It includes information such as the customer ID, order status, order ID, delivery ZIP code, and the assigned agent ID.

The table is partitioned by the ZIPCODE field, which means that data is distributed based on the ZIP code, allowing for efficient localization and retrieval of data for specific regions.

**Table schema:**

*ORDER(*

*CUSTOMER\_ID INT NOT NULL,*

*STATUS VARCHAR(10) NOT NULL,*

*ORDER\_ID VARCHAR(10),*

*ZIPCODE VARCHAR(5) NOT NULL,*

*AGENT\_ID INT,*

*CONSTRAINT fk\_order*

*FOREIGN KEY(ORDER\_ID)*

*REFERENCES ORDER\_ITEMS(ORDER\_ID)*

*) PARTITION BY LIST (ZIPCODE);*

**3) DELIVERY\_AGENT\_TABLE**

This table stores information about delivery agents, including their ID, name, the order they are assigned to, and the ZIP code for delivery.

It is also partitioned by the ZIPCODE field.

**Table schema:**

*DELIVERY\_AGENT\_TABLE(*

*AGENT\_ID SERIAL,*

*AGENT\_NAME VARCHAR(255) NOT NULL,*

*ORDER\_ID VARCHAR(10),*

*ZIPCODE VARCHAR(5) NOT NULL,*

*PRIMARY KEY(AGENT\_ID),*

*CONSTRAINT fk\_DELIVERY\_AGENT\_TABLE*

*FOREIGN KEY(ORDER\_ID)*

*REFERENCES ORDER\_ITEMS(ORDER\_ID)*

*) PARTITION BY LIST (ZIPCODE);*

**4) Inventory Table**

This table represents the inventory, including a universally unique identifier (UUID), warehouse ID, order ID, medication ID, and ZIP code.

The table is partitioned by the ZIPCODE field.

**Table schema:**

*Inventory (*

*UUID UUID DEFAULT uuid\_generate\_v4(),*

*WAREHOUSE\_ID INT NOT NULL,*

*ORDER\_ID VARCHAR(10),*

*MED\_ID INT NOT NULL,*

*ZIPCODE VARCHAR(10) NOT NULL,*

*PRIMARY KEY (UUID, ZIPCODE)*

*CONSTRAINT fk\_* *Inventory*

*FOREIGN KEY(ORDER\_ID)*

*REFERENCES ORDER\_ITEMS(ORDER\_ID)*

*) PARTITION BY LIST (ZIPCODE);*

**MongoDB database collection design:**

In MongoDB, we have incorporated sharding, which is a horizontal scaling approach that distributes data across multiple servers. It's typically based on a sharding key, and in our case, we use a key related to the geographic distribution, such as ZIPCODE. Sharding helps in achieving better read and write performance by distributing the data load across multiple shards.

Additionally, MongoDB's replication ensures high availability and fault tolerance. Each shard in the sharded cluster would have its set of replica nodes to provide redundancy and ensure data availability even in the case of node failures.

MongoDB includes the following collections.

**1) CUSTOMER\_DETAILS Collection:**

The CUSTOMER\_DETAILS collection stores information about customers, including their ID, name, ZIP code, and an array (ORDER\_INFO) containing details of the last 4 orders.

Storing the last 4 orders directly in the customer document can be beneficial for scenarios where users frequently check their recent order history. It can significantly improve read performance when a user wants to view their recent orders, as the data is readily available in the customer document without the need to execute a query to the PostgreSQL database.

This denormalization technique is a trade-off between read and write efficiency. While it may increase redundancy, it optimizes read operations for common use cases.

**Example entry:**

*{*

*"CUSTOMER\_ID": 1,*

*"CUSTOMER\_NAME": "John Doe",*

*"ZIPCODE": "12345",*

*"ORDER\_INFO": [*

*{"ORDER\_ID": "A123", "STATUS": "Delivered"},*

*{"ORDER\_ID": "B456", "STATUS": "Shipped"},*

*{"ORDER\_ID": "C789", "STATUS": "Processing"},*

*{"ORDER\_ID": "D012", "STATUS": "Delivered"}*

*]*

*}*

**2) MEDICINE\_DETAILS Collection:**

Attributes: MEDICINE\_ID (serial primary key), MEDICINE\_NAME, PRICE

This collection stores static information about medicines, such as their ID, name, and price.

**Example entry:**

*{*

*"MEDICINE\_ID": 1,*

*"MEDICINE\_NAME": "Aspirin",*

*"PRICE": 5*

*}*

**3) WAREHOUSE\_DETAILS Collection:**

Attributes: WAREHOUSE\_ID (serial primary key), WAREHOUSE\_NAME, ZIPCODE

This collection stores details about warehouses, including their ID, name, and ZIP code.

**Example entry:**

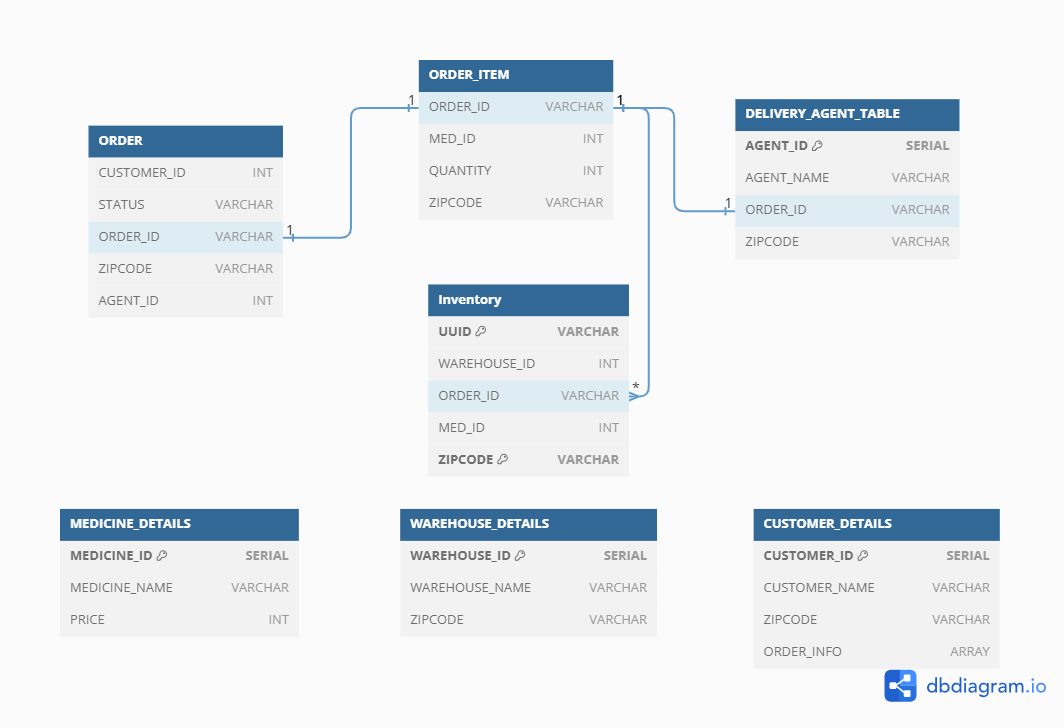
*{*

*"WAREHOUSE\_ID": 1,*

*"WAREHOUSE\_NAME": "Central Warehouse",*

*"ZIPCODE": "88345"*

*}*

ERD using Crow’s Foot notation

**Data Distribution Strategy Overview:**

Geographical Consideration:

Both PostgreSQL and MongoDB databases leverage data distribution based on geographical factors, specifically ZIP codes. This aligns with the hyperlocal nature of the delivery system, allowing for efficient data access and management specific to different regions.

Query Optimization:

The chosen distribution strategy aims to optimize queries related to specific geographic regions. By organizing data based on ZIP codes, the system can quickly retrieve and process information relevant to a particular area, improving overall query performance.

Scalability:

The data distribution plan facilitates scalability as the delivery system expands to new regions. By partitioning and sharding data based on ZIP codes, the databases can effectively scale horizontally, accommodating the growth of the hyperlocal delivery service.

Combination of Partitioning and Sharding:

PostgreSQL uses partitioning for tables, while MongoDB utilizes sharding for collections. This combination of techniques addresses the specific needs of each database system and the type of data they manage.

High Availability and Fault Tolerance:

MongoDB's replication strategy ensures high availability and fault tolerance by maintaining multiple copies of data across replica sets. This enhances the system's resilience to node failures and contributes to continuous service availability.

The chosen data distribution strategy considers geographical factors, optimizes queries, supports scalability, and ensures high availability and fault tolerance. The combination of partitioning and sharding is tailored to the specific requirements of PostgreSQL and MongoDB in the context of a hyperlocal delivery system for essential groceries and medications.