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:

{

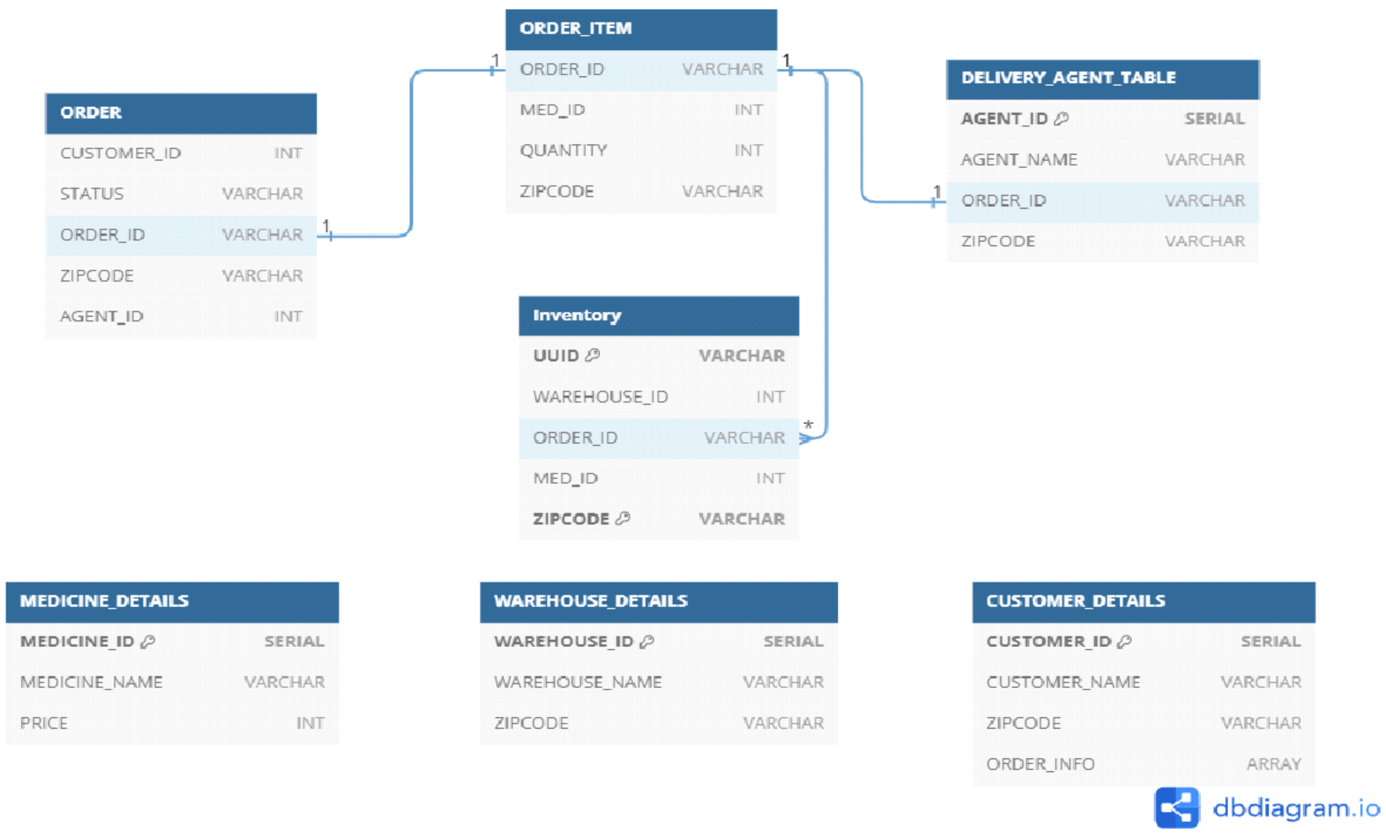
"WAREHOUSE_ID": 1,

"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.

Data Retrieval Proof:

delivery_agent table contents:

Query   Query History

```
select * from delivery_agent;
```

Data Output   Messages   Notifications

	agent_id integer	agent_name character varying (255)	order_id integer	zip_code integer
1	1	Grace	[null]	85200
2	2	Noah	[null]	85200
3	3	Leo	[null]	85200
4	4	Kate	[null]	85200
5	5	Ivy	[null]	85200
6	6	Bob	[null]	85201
7	7	Alice	[null]	85201
8	8	Jack	[null]	85201
9	9	Mia	[null]	85201
10	10	Grace	[null]	85201
11	11	Peter	[null]	85202
12	12	Ruby	[null]	85202
13	13	Noah	[null]	85202
14	14	Kate	[null]	85202
15	15	Leo	[null]	85202
16	16	Frank	[null]	85203
17	17	Alice	[null]	85203
18	18	Ruby	[null]	85203
19	19	Bob	[null]	85203

Total rows: 50 of 50   Query complete 00:00:00.060

inventory table contents:

Query   Query History

```
select * from inventory
```

Data Output   Messages   Notifications

	uuid [PK] integer	warehouse_id integer	order_id integer	med_id [PK] integer	zip_code [PK] integer
1	1	4	[null]	1	85200
2	2	7	[null]	1	85200
3	3	10	[null]	1	85200
4	4	2	[null]	1	85200
5	5	10	[null]	1	85200
6	6	1	[null]	1	85200
7	7	6	[null]	1	85200
8	8	4	[null]	1	85200
9	9	6	[null]	1	85200
10	10	5	[null]	1	85200
11	11	5	[null]	1	85200
12	12	7	[null]	1	85200
13	13	8	[null]	1	85200
14	14	2	[null]	1	85200
15	15	10	[null]	1	85200
16	16	1	[null]	1	85200
17	17	10	[null]	1	85200
18	18	5	[null]	1	85200
19	19	5	[null]	1	85200

Total rows: 1000 of 18000   Query complete 00:00:00.138

order_item table contents:

Query Query History

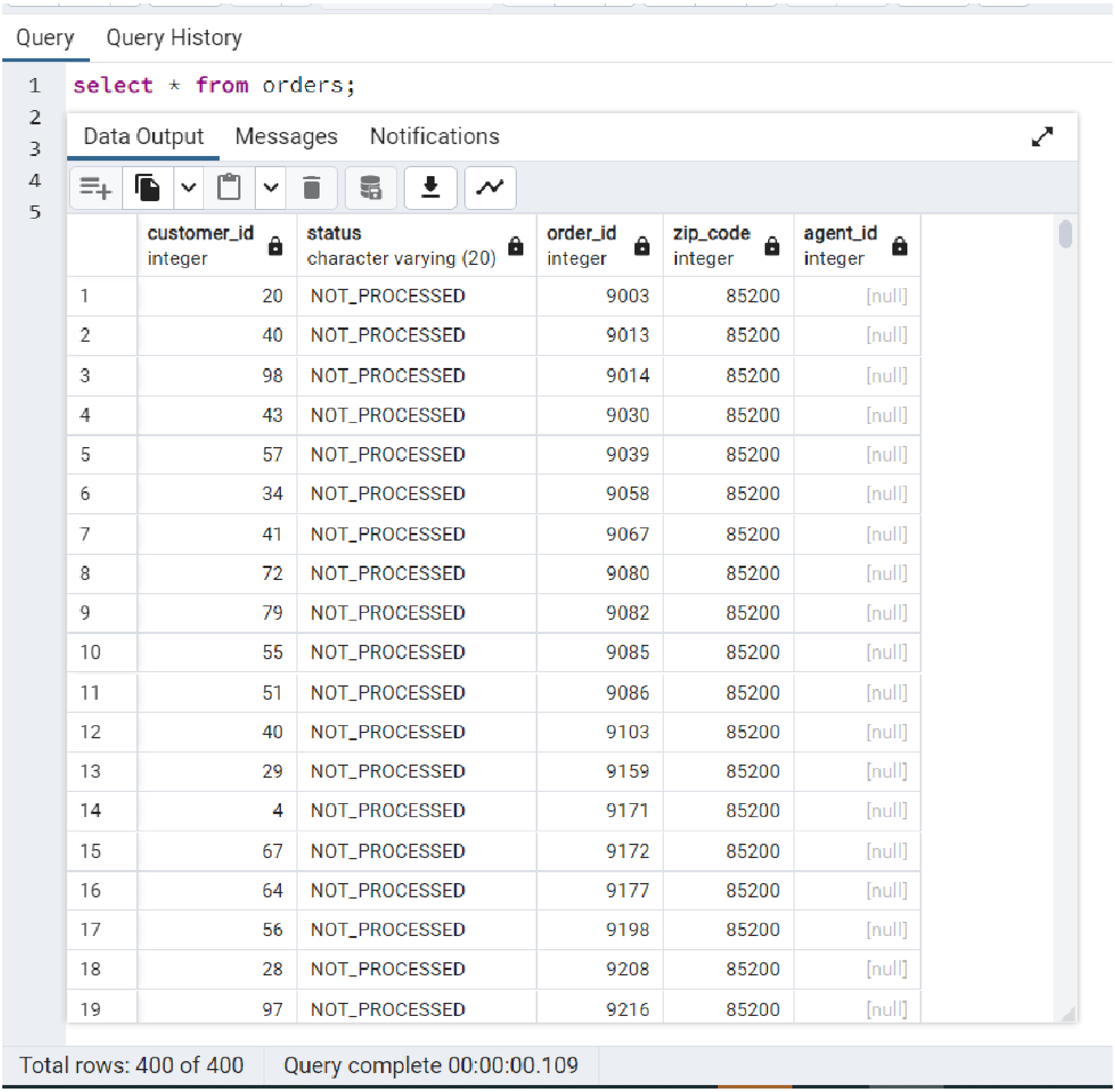
```
select * from order_item;
```

Data Output Messages Notifications

	order_id integer	med_id integer	quantity integer	zip_code integer
1	9003	4	21	85200
2	9003	7	29	85200
3	9003	11	17	85200
4	9013	45	17	85200
5	9013	3	17	85200
6	9013	14	13	85200
7	9013	8	7	85200
8	9014	88	25	85200
9	9030	46	37	85200
10	9030	85	33	85200
11	9030	2	23	85200
12	9030	98	9	85200
13	9030	49	28	85200
14	9039	91	31	85200
15	9058	83	14	85200
16	9058	12	13	85200
17	9058	90	24	85200
18	9058	80	31	85200
19	9067	65	33	85200

Total rows: 1000 of 1204 Query complete 00:00:00.206

order table contents:



Query Query History

```
select * from orders;
```

Data Output Messages Notifications

	customer_id integer	status character varying (20)	order_id integer	zip_code integer	agent_id integer
1	20	NOT_PROCESSED	9003	85200	[null]
2	40	NOT_PROCESSED	9013	85200	[null]
3	98	NOT_PROCESSED	9014	85200	[null]
4	43	NOT_PROCESSED	9030	85200	[null]
5	57	NOT_PROCESSED	9039	85200	[null]
6	34	NOT_PROCESSED	9058	85200	[null]
7	41	NOT_PROCESSED	9067	85200	[null]
8	72	NOT_PROCESSED	9080	85200	[null]
9	79	NOT_PROCESSED	9082	85200	[null]
10	55	NOT_PROCESSED	9085	85200	[null]
11	51	NOT_PROCESSED	9086	85200	[null]
12	40	NOT_PROCESSED	9103	85200	[null]
13	29	NOT_PROCESSED	9159	85200	[null]
14	4	NOT_PROCESSED	9171	85200	[null]
15	67	NOT_PROCESSED	9172	85200	[null]
16	64	NOT_PROCESSED	9177	85200	[null]
17	56	NOT_PROCESSED	9198	85200	[null]
18	28	NOT_PROCESSED	9208	85200	[null]
19	97	NOT_PROCESSED	9216	85200	[null]

Total rows: 400 of 400 Query complete 00:00:00.109