POLYGLOT PERSISTENCE

NOSQL ASSIGNMENT

K.M.D.R BANDARA

INDEX NO: 18000223

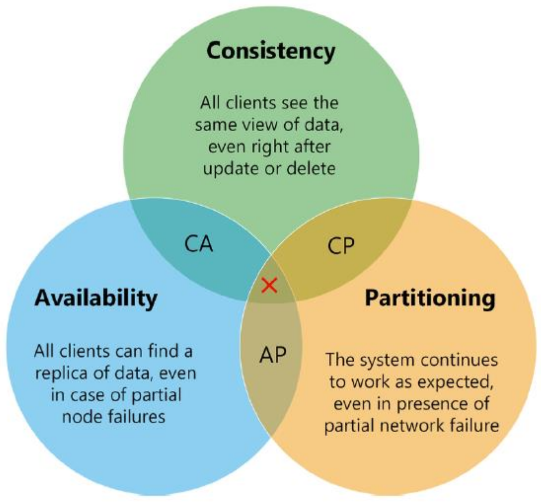
TABLE OF CONTENT

• Introduction

This document only describes about the database design of “Order Delivery System (ODS)” Which is a system that connect customers and suppliers. There are many numbers of customers suppliers, customers and orders, Therefore the system has to have a heavy database to store all those data.

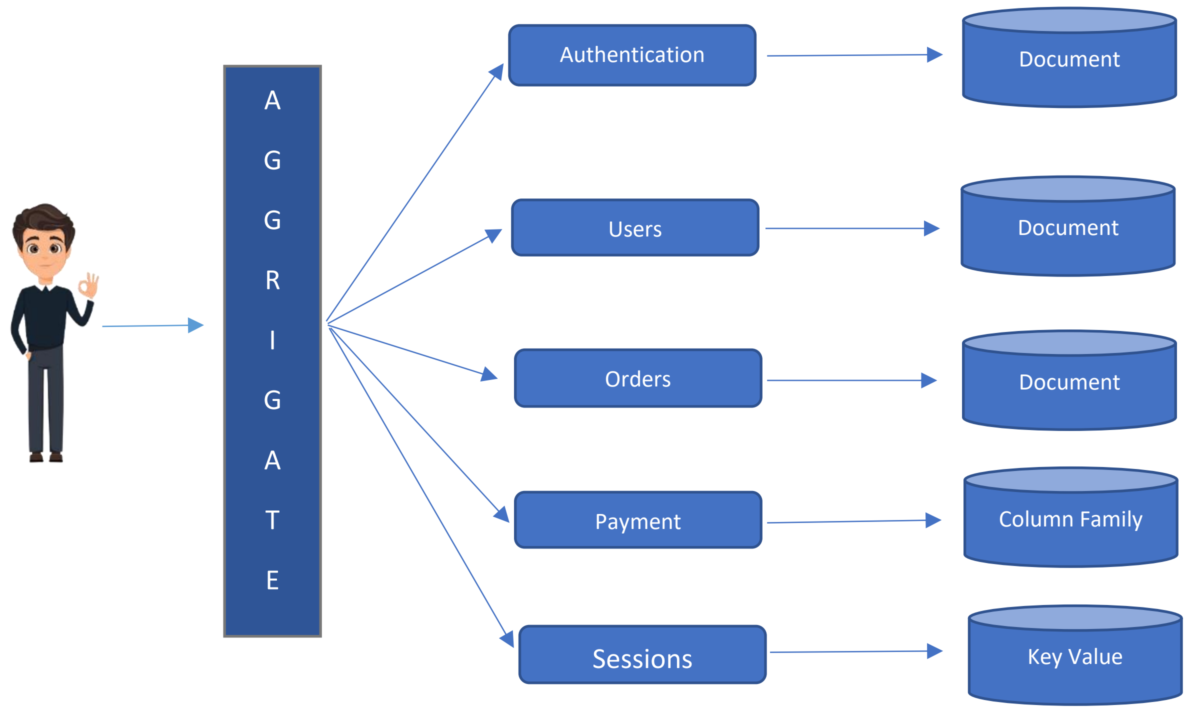
For this system uses different models of databases to support the logical functionalities of the given workflow of ODS to make it fast and efficient. When choose the appropriate databases for each component CAP theorem is used. So, this design is intended to provide the concept of polyglot persistence.

CAT Theorem


Consistency
All clients see the same view of data, even right after update or delete
CA
CP
Availability
All clients can find a replica of data, even in case of partial node failures
AP
Partitioning
The system continues to work as expected, even in presence of partial network failure

The document also provides detailed reasoning for each database type selected for the scenarios, aggregates and the queries required with assumptions and justifications.

- **ODS Overview**



- **Assumptions.**
  1. Customers can order many items at once from one shop only.
  2. Data validation is primarily done at the front-end.
  3. Sensitive data about customers and suppliers are not stored in the system.

- **Work Flow**

Authentication: When a user tries to login to the system by using username(email) and password authenticate process should take place by using user email and password. After the user log in, the authorization process verifies what the user can do.

User : All suppliers, customers, delivery managers and system administrators comes under this category. All the parties can register with the system. Customers can place orders through the system and they also can select preferred suppliers.

Orders : When the order is places by the customer, they receive an “order receive” response immediately after placing the order. Then the order is assigned to suppliers through the system. When the supplier accepts the order, the customer is notified with the delivery date, time and cost.

Payment : In this system, customers could make the payment online or cache on delivery.

Sessions : Keep the session where the user’s login.

- **Reasons for select those type of databases for each different logical functional stage**.

There are four types of databases in NOSQL,

1. Document databases
2. Key Value databases
3. Column Family databases
4. Graph data structure.

Authentication

Database type: - Document type databases (MongoDB)

Users should authenticate before communicating with the servers. According to **CAP theorem** analysis authentication should have high consistency, average availability and high partition tolerance. Databases should provide fast responses to enhance the authorization process. As well as user data should have stored a long time of time. So document type databases are more suitable for store authentication data.

User

Database type: - Document type databases (MongoDB)

In document databases it is easier to store and query data in a database by using the same document-model format they use in their application code. The flexible, semi structured, and hierarchical nature of documents and document databases allows to evolve with applications’ needs. The document model works well with use cases such as catalogs, user profiles, and content management systems where each document is unique and evolves over time. Document databases enable flexible indexing, powerful ad hoc queries, and analytics over collections of documents. In this system there are different types of users so the data about all of them are not similar. Therefore, it is easy to use a document type database.

Orders

Database type: - Document type databases (MongoDB)

In this system customers could place orders via SMS, call, mobile app or online then we can achieve consistency and partition tolerance together.

According to **CAP** analysis, the using database type should have high consistency, average availability, and high partition tolerance. As well need high deep query ability. After considering above requirements most suitable database type is document type databases for store orders data.

Payment

Database type: - Column Family databases (Cassandra)

Payment calculation and handling is a very critical part of the system. A separate database should be used to keep the records. In this system customers could make the payment online or cash on delivery. As in the assumption below all payment security handle by the payment gateway.

According to **CAP** analysis, the database which use for store payment data should have high availability, high partition tolerance and average consistency. As well as calculations are very critical, responsible response time, transaction like qualities also important for this database. So, column family database type is the most suitable type to store payment data.

Sessions

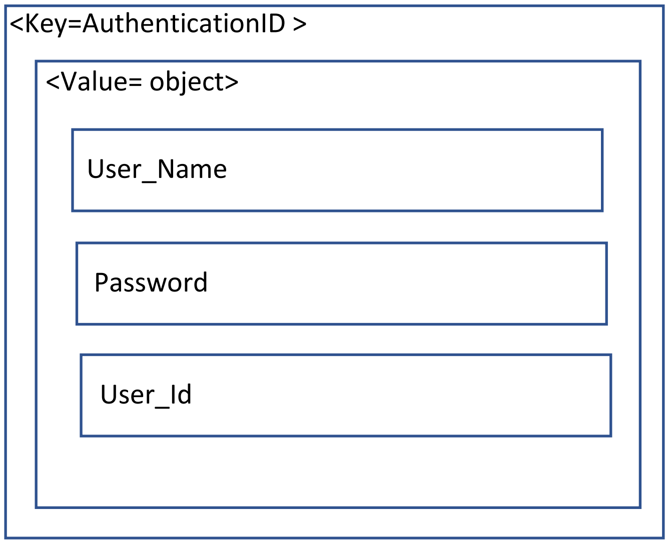
Database type: - Key Value databases (riak)

A key-value database stores data as a collection of key-value pairs in which a key serves as a unique identifier. Therefore, in here the purpose is to check whether the user is online or not. So, the status can be got as value and user ids can be get as keys. Hence the status can be retrieved using user id easily. As well high consistency and availability are very important.

- **Aggregations and Queries**

Authentication

Aggregation: -



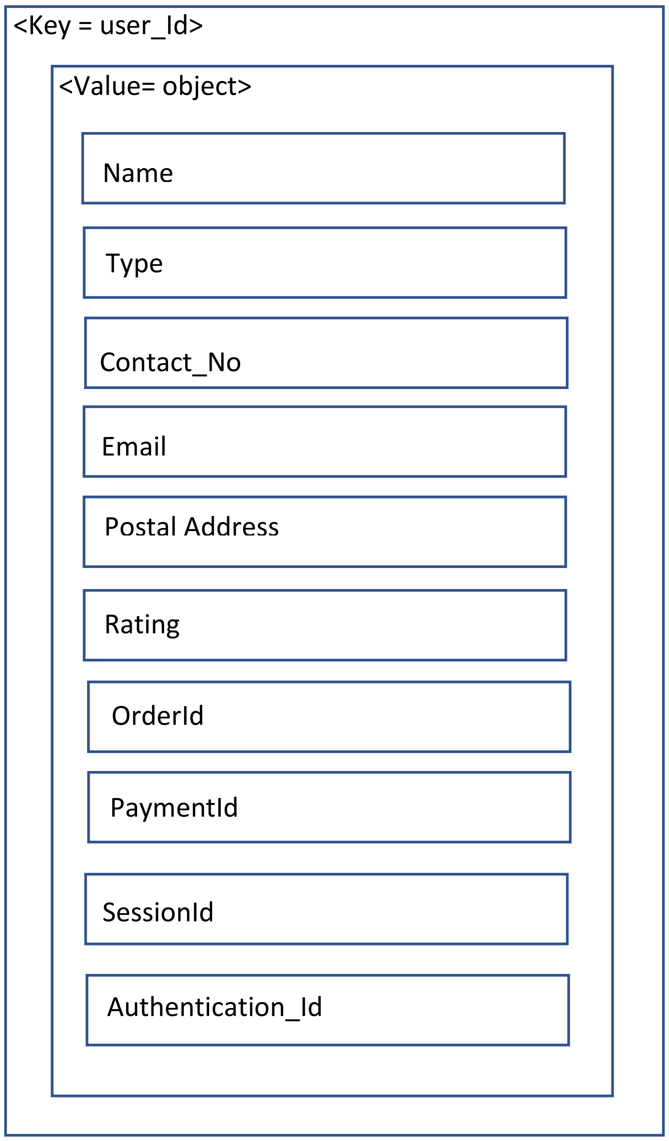
Queries: -

Insert data

```
db.authentication.insertOn({
    User_Name:"Darshana",
    Password:"********"
})
```

Users

Aggregation: -



Queries: -

Insert Data

```
Db.Users.insertOn(
    {
        UserName: “Darshana Bandara”,
        Type: “Customer”,
        PostalAddress: “Ibbagamuwa, Kurunegala”,
        PhoneNumber: “0717365756”,
        Email: “darshanaravindu9196@gmail.com”
    }
)
```

Find details

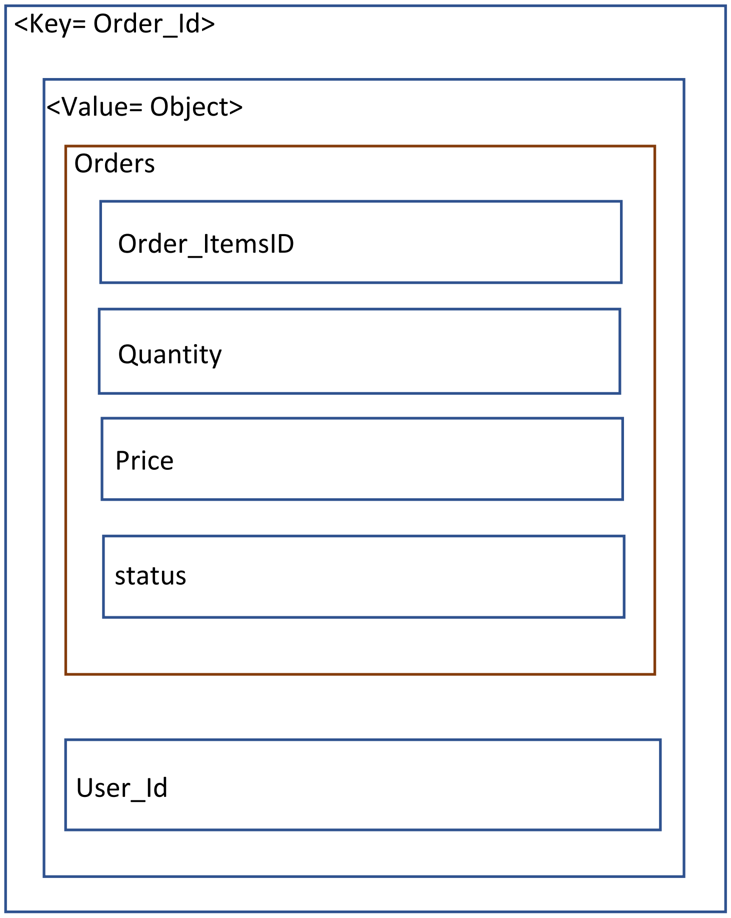
```
db.users.find({name:"Darshana"})
```

Fletch most rated 10 suppliers

```
db.suppliers.find().sort({rating:-1}).limit(10)
```

Orders

Aggregation: -



Queries: -

Insert Data

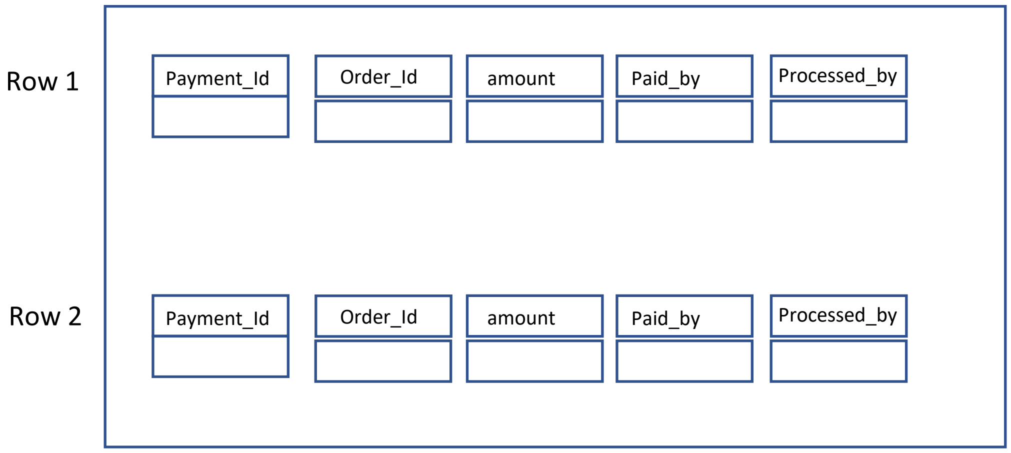
```
db.orders.insertOne( {
"orderID": 0005,
"order" : {
    "order_ItemsID" : "01",
    "quantity" : "10",
    "status" : "Pending",
    "userId" : 0001,
}
})
```

Get customer details according to order_ID

```
var orders = db.orders.find({order_Id:0005})
var customerId = orders.user_Id
db.users.find({user_Id:customerId})
```

Payment

Aggregation: -



Queries: -

Create Table

```
CREATE TABLE payment(
    Payment_id int PRIMARY KEY,
    orderId int,
    amount int,
    paidBy int,
    processedBy int
)
```

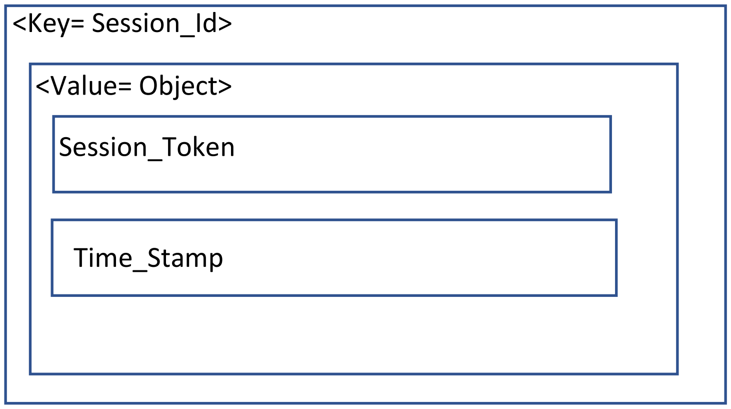
Insert data into a table

```
INSERT INTO  payment (Payment_id, orderId, amount, paidBy, processedBy)

VALUES (001, 0005, 300.00, 0001, 1005)
```

Session

Aggregation: -



Queries: -

Create table

```
CREATE TABLE Session (
    sessionID INT64 NOT NULL,
    SessionToken INT64 NOT NULL,
    TimeStamp TIMRSTAMP NOT NULL,
    )
```

Insert Table

```
INSERT INTO Session (SessionToken, TimeStamp) VALUES
    (“active”, “22.30”);
```