# **Assignment - NoSQL [ Dunder Mifflin Paper Company ]**

As per the given use case , I would go forward with a Document Store NoSQL database , specifically MongoDB. As in this case an easily scalable, resilient database is needed.

Doing a Fit analysis we can reach the following conclusions:

1. As per the Cap theorem a database can allow for any two of the three criteria of Consistency, Availability and Partition Tolerance. A Consistent and Partition Tolerant database is what’s needed for the given use case. Where the data should be consistent across nodes and the system continues to operate during failures of nodes. Which leaves us with databases like Redis, MongoDB , HBase etc.
2. We can see there is a requirement for all three process which are writes, reads and updates as such we will need a NoSQL Database that can handle complex, evolving schema and provide a balanced performance of read: write ratio . Also a need is there to be able to lookup objects using columns or values other than key values (need for complex indexes to be created).

**Why Mongodb ?**

1. Document databases are considered to be non-relational (or NoSQL) databases. Instead of storing data in fixed rows and columns, document databases use flexible documents. Document databases are the most popular alternative to tabular, relational databases.
2. A document typically stores information about one object and any of its related metadata. Documents store data in field-value pairs. The values can be a variety of types and structures, including strings, numbers, dates, arrays, or objects.
3. Document databases have a flexible schema, meaning that not all documents in a collection need to have the same fields. They map to objects in most popular programming languages, which allows developers to rapidly develop their applications and are distributed, which allows for horizontal scaling.
4. Thus a Document Store Database would be appropriate for this use case. Keeping this in mind MongoDB would be used for the implementation as it has robust support and works well with multiple languages for development.

**Normalised SQL Model for Dunder Mifflin Paper Company**

Graphical user interface, application

Description automatically generated

**De-Normalised NoSQL Data Model would be as follows:**

**Collection Name:** Stores

**Index created on :**

Multi key index on address , Compound index on staff.name and staff.id as a particular employee may be searched among the stores.

Would not advise to do a lookup on Items from stores collections, rather using the items collection would be faster and less expensive.

**Sharded On : State**

{

"id" : "Object/String",

"schema\_version" : "int",

"name" : "String",

"address" : {

"street" : "String",

"city" : "String",

"district" : "int",

"state" : "String",

"country" : "String",

"region" : "String"

},

"staff" : [

{

"employee\_id": "int",

"name" : "string",

"phone\_number" : "int",

"date\_of\_joining" : "datetime",

"designation" : "string",

"reporting\_manager\_id" : "int",

"reporting\_manager\_name" : "string"

}

],

"items" : [

{

"item\_id": "int",

"item\_name" : "string",

"brand" : "string",

"description" : "string",

"category" : "string",

"mrp" : "float",

"discount" : "float",

"price" : "float",

"quantity" : "float",

"sold" : "int",

"available" : "int",

"defective" : "int"

}

]

}

**Collection Name :** Reviews

**Index created on :**

Multi Key index on Reviews to fetch reviews by ratings

Compound index on order\_details.item\_id and order\_details.item\_name so as to lookup reviews specific to a product/item

**Sharded on: Ratings**

{

"id" : "Object/String",

"schema\_version" : "int",

"customer\_id" : "int",

"customer\_name" : "string",

"order\_details" : {

"order\_id" : "int",

"item\_id" : "String",

"item\_name" : "String",

"status" : "String",

"quantity" : "String"

},

"reviews" : [

{

"review\_id": "int",

"review\_body" : "string",

"ratings" : "int"

}

],

}

**Collection Name :** Items

**Index created on :**

Multi key index on product details and store\_details so that lookup on items can be done based on product details or store availability. This will allow for a fast lookup on the items available

**Sharded On : product\_details.category**

{

"id" : "Object/String",

"schema\_version" : "int",

"item\_id" : "int",

"product\_details" : {

"name" : "string",

"description" : "int",

"category" : "string",

},

"store\_details" :[

{

“id” : “int”

“name” : “string”,

“address” : “string”,

“type” : “string”,

“quantity” : “int”

}

],

"mrp" : "float",

"discount" : "float",

"price" : "float",

"quantity" : "float",

"sold" : "int",

"available" : "int",

"defective" : "int",

}

**Collection Name :** Customer

**Index created on :**

Compound index on firstname, lastname . Index on Email Id/username and a multi key index on the address would suffice as most users are searched referenced by name, username, email id or location

**Sharded On :** address.state

{

"id" : "Object/String",

"schema\_version" : "int",

"firstname" : "string",

"lastname" : "string",

"address" : {

"street" : "String",

"city" : "String",

"district" : "int",

"state" : "String",

"country" : "String",

"region" : "String"

},

"username" : "string",

"mobile" : "int",

"email" : "string",

"registered\_at" : "datetime",

"reviews" : [

{

"product\_id" : "int",

"product\_name" : "string",

"review\_id" : "int",

"review\_body" : "string",

"rating" : "int"

}

]

}

**Collection Name :** Order\_Items

**Index created on :**

Multi key index individually on item\_details, store details and customer would suffice.

Sharded on : store\_details.address.state

{

"id" : "Object/String",

"schema\_version" : "int",

"item\_details" : [

{

"name" : "string",

"description" : "int",

"category" : "string",

"mrp" : "float",

"discount" : "float",

"price" : "float",

"quantity" : "float"

}

],

"store\_details" : {

“id” : “int”

“name” : “string”,

"address" : {

"street" : "String",

"city" : "String",

"district" : "int",

"state" : "String",

"country" : "String",

"region" : "String"

},

“type” : “string”

},

“customer” : {

“id” : “int”,

“name” : “string”

},

"status" : "string",

"subtotal" : "float",

"shipping" : "float",

"tax" : "float",

"total" : "float",

"total\_discount" : "float",

"effective\_price" : "float"

}