**Schema Design:**

Mongo DB is a schemaless i.e dynamic schema and each document can have its know schema. And this make us tough to decide which schema does a document should have. In most applications, documents as same schema but there are still choices they made. We can embedded a documents or we could put it as a separate collection.

And these decisions will have the impact on performance and ease of programming. So through the series of use cases, we are going to take you through Mongo DB schema design.

**public** **static** **void** main(String[] args) {

**final** MongoClient mongoClient = **new** MongoClient(**new** MongoClientURI(

"mongodb://localhost"));

**final** MongoDatabase blogDatabase = mongoClient.getDatabase("school");

MongoCollection<Document> studentsCollection = blogDatabase

.getCollection("temp2");

Bson sort = Sorts.*ascending*("\_id");

MongoCursor<Document> cursor = studentsCollection.find().sort(sort).skip(1)

.iterator();

**while** (cursor.hasNext()) {

Document document = cursor.next();

List<Document> docs = (ArrayList<Document>) document.get("scores");

Float lowest = -1f;

Document lowestDoc = **null**;

List<Document> finalDocs = **new** ArrayList<Document>();

**for** (Document document2 : docs) {

Float float1 = Float.*valueOf*(document2.get("score").toString());

**if**(!document2.get("type").toString().equalsIgnoreCase("homework")){

finalDocs.add(document2);

**continue**;

}

**if** (lowest >= float1 || lowest.floatValue() == -1f) {

lowest = float1;

**if**(lowestDoc != **null**){

finalDocs.add(lowestDoc);

}

lowestDoc = document2;

}**else**{

finalDocs.add(document2);

}

}

Document updateDocs = **new** Document().append("$set", **new** Document().append("scores", finalDocs));

studentsCollection.updateMany(**new** Document().append("\_id", document.get("\_id")),updateDocs );

//System.out.println("\_id" + document.get("\_id") + " " + finalDocs ) ;

}

}

1. **MongoDB schema Desing**

**In** relational database system, we have the best database schema design principle i.e to keep the schema in the third normal form.

In MongoDB, we can even keep the database desing in the third normal form. But in Mongo, we are keep the data, such that it is conducive to the application using the data(Application driven schema).

So we have to think about the application data patterns. We have to think about which piece of data are used together, what pieces of data is mostly read-only, what piece of data is written all the time. And then, we gonna organise data within mongoDB to specifically suit the appicaiton data accesss patterns. This is bit different from the relational world, where instead we try to keep the data in a way which is agnostic to the application.

Basic Facts about MongoDB(revision):

1. it supports rich documents.
2. Pre joins/ embedded data
3. No joins . if we have to do the join then we have to do in the applicaion. (Joins are very hard to scale and we are more concerned on performance)
4. No constraints. (no FK or any other)
5. Atomic operations(mongo does not support transactions but support atomic operation within a document.).
6. No declared schema(But usually, all documents in a collection has a pretty similar structure but only small changes ).
7. **Relational Normalization**

Lets start by first reviewing the 3rd normal form which is used in the relation database schema design. This will helps us to understand the trade-offs that we talk about the mongoDB world.

**Goals of Normalization:**

1. Free the database from modification anomalies i.e data should be in such a way that we should not update the same data, in different rows. Like we have a database table of posts with columns postid, postContent, author\_name, author\_emailID. Now assume there are multiple rows with same email id i.e author. In such case, we have make sure all the emailIDs are modified in update else the data may go wrong. Instead we can keep the author details in one table and refer it from post table, which will avoid problems.
2. Minimize the resdesign when extending the DB. It is little controversal,if it does or not. But that’s the idea of normalisation.
3. Avoid bias to any particular access pattern.

While design the schema with MongoDb, then we gonna complete forget the 3rd goal from Realation DB. And the reason is, if we are not biased towards any particular access pattern, we are equally bad at all of them. And one of the ideas of mongoDB is to tune up your database to the application that we are trying to write and the problem we are trying to solve. And so, we are not going to worry about avoiding the bias towards any particular

For 2nd goal, mongoDB is very flexible in design. We can modify the structure at point of time and we can use the existing documents in the same old schema fasion. We don’t need to change anything.

For 1st goal, we may think that by embedding data would cause these, it doesn’t have to. And if fact, we are mostly going to avoid embedding data in documents in mongoDB, in ways that create these anomalies. So we are going to be careful not to create them for the most part. Occasionally, for performance reasons, we are goint to decide that we do want to duplicate data within the documents but that this don’t be the default. The default is that we are going to avoid it, so that we don’t have this anomalies where data can be inconsistently changed. And in some applicaion we might allow and it doesn’t matter. Or you might want to keep the data up-to date in the applcaion, but mostly we are going to avoid it.