**What does being schema-less mean for a NoSQL Database?**

*Schema-less is a bit of a misnomer, it's better to think of it as:*

*SQL = Schema enforced by a RDBMS on Write*

*NoSQL = Partial Schema enforced by the DBMS on Write, PLUS schema fully enforced by the Application on Read (Externalised schema)*

*So while a supposed Schema-less NoSQL data-store will in theory allow you to store any data you like (typically key value pairs, in a document) without prior knowledge of the keys, or data types, it will be pointless unless you have some mechanism to retrieve and use the data. So essentially the schema is partially moved from the RDBMS into the application code. I say partially as you'll have added Indexes to document collections and or partitioned the data for performance, so the NoSQL DBMS will have a partial schema defined locally, and possibly enforced via Unique constraints.*

*As to adding additional attributes to a document/object in the store. Depending on how much padding is around the document (un-used space), in it's physical data block, adding a few more key value pairs to the documents may result in the document having to be physically moved to a larger contiguous block of storage, and the associated indexes re-built. If you plan to use the new keys in a frequently utilised query then you'll be wanting to also add a suitable new index, which will obviously require some physical storage, take a while to initially build and possibly lead you to ask the sysadmin to allocate more memory to the DBMS, to allow the new index(s) to be cached.*

**When to Redis? When to MongoDB? [closed]**

votes

*I would say, it depends on kind of dev team you are and your application needs.*

*For example, if you require a lot of querying, that mostly means it would be more work for your developers to use Redis, where your data might be stored in variety of specialized data structures, customized for each type of object for efficiency. In MongoDB the same queries might be easier because the structure is more consistent across your data. On the other hand, in Redis, sheer speed of the response to those queries is the payoff for the extra work of dealing with the variety of structures your data might be stored with.*

*MongoDB offers simplicity, much shorter learning curve for developers with traditional DB and SQL experience. However, Redis's non-traditional approach requires more effort to learn, but greater flexibility.*

*Eg. A cache layer can probably be better implemented in Redis. For more schema-able data, MongoDB is better. [Note: both MongoDB and Redis are technically schemaless]*

*If you ask me, my personal choice is Redis for most requirements.*

*Lastly, I hope by now you have seen http://antirez.com/post/MongoDB-and-Redis.html*

**MongoDB relationships: embed or reference?**

*This is more an art than a science. The Mongo Documentation on Schemas is a good reference, but here are some things to consider:*

*Put as much in as possible*

*The joy of a Document database is that it eliminates lots of Joins. Your first instinct should be to place as much in a single document as you can. Because MongoDB documents have structure, and because you can efficiently query within that structure (this means that you can take the part of the document that you need, so document size shouldn't worry you much) there is no immediate need to normalize data like you would in SQL. In particular any data that is not useful apart from its parent document should be part of the same document.*

*Separate data that can be referred to from multiple places into its own collection.*

*This is not so much a "storage space" issue as it is a "data consistency" issue. If many records will refer to the same data it is more efficient and less error prone to update a single record and keep references to it in other places.*

*Document size considerations*

*MongoDB imposes a 4MB (16MB with 1.8) size limit on a single document. In a world of GB of data this sounds small, but it is also 30 thousand tweets or 250 typical Stack Overflow answers or 20 flicker photos. On the other hand, this is far more information than one might want to present at one time on a typical web page. First consider what will make your queries easier. In many cases concern about document sizes will be premature optimization.*

*Complex data structures:*

*MongoDB can store arbitrary deep nested data structures, but cannot search them efficiently. If your data forms a tree, forest or graph, you effectively need to store each node and its edges in a separate document. (Note that there are data stores specifically designed for this type of data that one should consider as well)*

*It has also been pointed out than it is impossible to return a subset of elements in a document. If you need to pick-and-choose a few bits of each document, it will be easier to separate them out.*

*Data Consistency*

*MongoDB makes a trade off between efficiency and consistency. The rule is changes to a single document are always atomic, while updates to multiple documents should never be assumed to be atomic. There is also no way to "lock" a record on the server (you can build this into the client's logic using for example a "lock" field). When you design your schema consider how you will keep your data consistent. Generally, the more that you keep in a document the better.*

*For what you are describing, I would embed the comments, and give each comment an id field with an ObjectID. The ObjectID has a time stamp embedded in it so you can use that instead of created at if you like.*

***Where does mongodb stand in the CAP theorem?***

*MongoDB is strongly consistent by default - if you do a write and then do a read, assuming the write was successful you will always be able to read the result of the write you just read. This is because MongoDB is a single-master system and all reads go to the primary by default. If you optionally enable reading from the secondaries then MongoDB becomes eventually consistent where it's possible to read out-of-date results.*

*MongoDB also gets high-availability through automatic failover in replica sets:* [*http://www.mongodb.org/display/DOCS/Replica+Sets*](http://www.mongodb.org/display/DOCS/Replica+Sets)

***How to check if a field contains a substring?***

*db.users.findOne({"username" : {$regex : ".\*son.\*"}});*

***What are alternatives to MongoDB?***

*There are a number of NoSQL alternatives such as Redis, CouchDB, and Cassandra. Alternatively, you could turn to a relational database management system such as MySQL, Oracle, or SQL Server.*

***Update MongoDB field using value of another field***

*The best way to do this is to use the aggregation framework to compute our new field.*

*MongoDB 3.4*

*The most efficient solution is in MongoDB 3.4 using the $addFields and the $out aggregation pipeline operators.*

*db.collection.aggregate(*

*[*

*{ "$addFields": {*

*"name": { "$concat": [ "$firstName", " ", "$lastName" ] }*

*}},*

*{ "$out": "collection" }*

*]*

*)*

*Note that this does not update your collection but instead replace the existing collection or create a new one. Also for update operations that require "type casting" you will need client side processing, and depending on the operation, you may need to use the find() method instead of the .aggreate() method.*

***Explain what is horizontal scalability?***

*Horizontal scaling means that you scale by adding more machines into your pool of resources whereas Vertical scaling means that you scale by adding more power (CPU, RAM) to an existing machine.*

***What are the differences between MongoDB and MySQL?***

*My SQL:*

*Written in: c or c++*

*Type: RDBMS*

*Main points: Table, Row or column*

*License: GPL V2*

*Schemas: Strict*

*Scaling: vertically*

*Key features: Full-text searching and indexing, Triggers, SSL support, Unicode support, Query cashing, Integrated replication support*

*Risk: Risk of SQL injection attacks*

*Analysis: a great choice if you have a structure data and need a traditional relational database*

*Data structure: Structure data with clear schema*

*Developer: Microsoft*

*Initial release: 1989*

*License: commercial*

*Server operating systems: Linux, windows*

*Server-side scripts: Transact SQL and .NET languages*

*Triggers: Yes*

*Map-reduce: No*

*MongoDB:*

*Written in: c , c++ and javascript*

*Type: Document oriented*

*Main points: Collection,Document, and field*

*License: GNU AGPL V3.0*

*Schemas: dynamic*

*Scaling: Horizontally*

*Key features: Native replication, In memory speed, embedded data models support, Rich query language support*

*Risk: Less risk of attack due to design*

*Analysis: a great choice if you have a structured or unstructured data with the potential for rapid growth.*

*Data Structure: No schema definition is required*

*Developer: MongoDB, INC*

*Initial release: 2009*

*License: open source*

*Server operating systems: Linux, Windows,Solaris*

*Server-side scripts: javascript*

*Triggers: No*

*Map-reduce: yes*

*Read more: Difference between MySQL and MongoDB | Comparing MongoDB with MySQL*