MONGODB

Content

- Part 1: Introduction & Basics
- 2: CRUD
- 3: Schema Design
- 4: Indexes
- 5: Aggregation
- 6: Replication & Sharding

History

mongoDB = “Hu**mongo**us DB”

- Open-source
- The port 27017 is used for mongoDB server.
- Document-based
- “High performance, high availability”
- Automatic scaling
- C-P on CAP

-blog.mongodb.org/post/475279604/on-distributed-consistency-part-1

-mongodb.org/manual

Other NoSQL Types

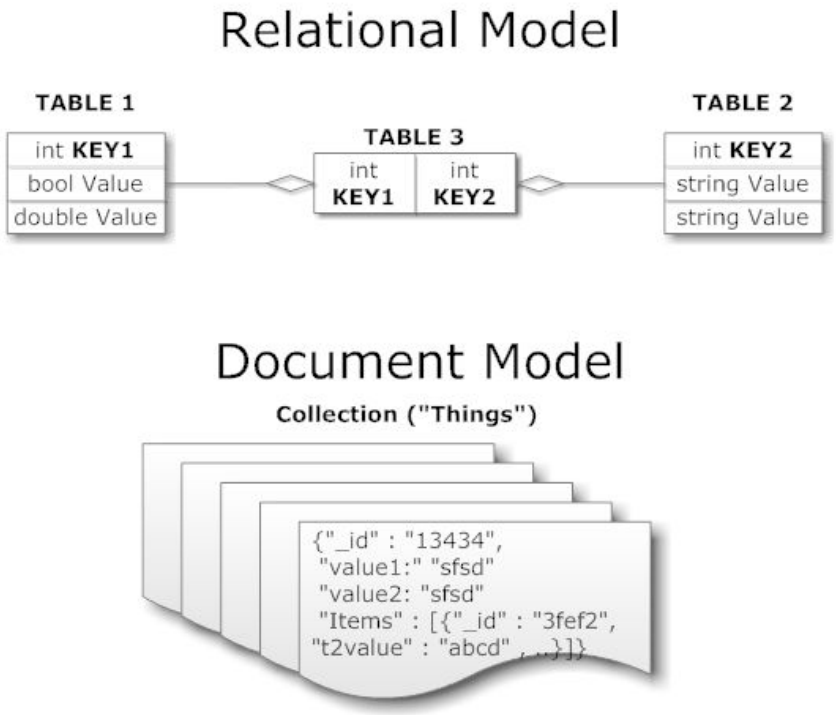
Key/value (Dynamo)

Columnar/tabular (HBase)

Document (mongoDB)

Graph (Neo4j)

http://www.aaronstannard.com/post/2011/06/30/MongoDB-vs-SQL-Server.aspx



Motivations

Problems with SQL

- Rigid schema
- Not easily scalable (designed for 90's technology or worse)
- Requires unintuitive joins

Perks of mongoDB

- Easy interface with common languages (Java, Javascript, PHP, etc.)
- DB tech should run anywhere (VM's, cloud, etc.)
- Keeps essential features of RDBMS's while learning from key-value noSQL systems

Company Using mongoDB

"MongoDB powers Under Armour's online store, and was chosen for its dynamic schema, ability to scale horizontally and perform multi-data center replication."

http://www.mongodb.org/about/production-deployments/

In Good Company

mongoDB

-Steve Francia, http://www.slideshare.net/spf13/mongodb-9794741?v=qf1&b=&from_search=13

The _id Field

- By default, each document contains an _id field. This field has a number of special characteristics:
  - Value serves as primary key for collection.
  - Value is unique, immutable, and may be any non-array type.
  - Default data type is ObjectId, which is “small, likely unique, fast to generate, and ordered.” Sorting on an ObjectId value is roughly equivalent to sorting on creation time.

http://docs.mongodb.org/manual/reference/bson-types/

mongoDB vs. SQL

mongoDB	SQL
Document	Tuple
Collection	Table/View
PK: _id Field	PK: Any Attribute(s)
Uniformity not Required	Uniform Relation Schema
Index	Index
Embedded Structure	Joins
Shard	Partition

CRUD

Create, Read, Update, Delete

Getting Started with mongoDB

To install mongoDB, go to this link and click on the appropriate OS and architecture: http://www.mongodb.org/downloads

First, extract the files (preferrably to the C drive).

Finally, create a data directory on C:\ for mongoDB to use
i.e. “md data” followed by “md data\db”

http://docs.mongodb.org/manual/tutorial/install-mongodb-on-windows/

Getting Started with mongoDB

Open your mongodb/bin directory and run mongod.exe to start the database server.

To establish a connection to the server, open another command prompt window and go to the same directory, entering in mongo.exe. This engages the mongodb shell—it's that easy!

http://docs.mongodb.org/manual/tutorial/getting-started/

CRUD: Using the Shell

To check which db you're using	db
Show all databases	show dbs
Switch db's/make a new one	use <name>
See what collections exist	show collections

Note: db's are not actually created until you insert data!

CRUD: Using the Shell (cont.)

To insert documents into a collection/make a new collection:

db.<collection>.insert(<document>)
<=>
INSERT INTO <table>
VALUES(<attributevalues>);

CRUD: Inserting Data

Insert one document

```
db.<collection>.insert({<field>:<value>})
```

Inserting a document with a field name new to the collection is inherently supported by the BSON model.

To insert multiple documents, use an array.

MongoDB – Bulk.insert() Method

MongoDB, the Bulk.insert() method is used to perform insert operations in bulk.

The Bulk.insert() method is used to insert multiple documents in one go.

Unordered Insertion of documents:

```
var bulk = db.students.initializeUnorderedBulkOp();
bulk.insert( { first_name: "Sachin", last_name: "Tendulkar" } );
bulk.insert( { first_name: "Virender", last_name: "Sehwag" } );
bulk.insert( { first_name: "Shikhar", last_name: "Dhawan" } );
bulk.insert( { first_name: "Mohammed", last_name: "Shami" } );
bulk.insert( { first_name: "Shreyas", last_name: "Iyer" } );
bulk.execute();
```

db.students.find().sort({'_id':-1}).limit(5).pretty()

Schema Validation in MongoDB

- **Schema validation** in **MongoDB** provides a structured approach to define and enforce rules for document structures within collections.
- By specifying validation criteria such as data types, required fields, and custom expressions using **JSON** Schema syntax, MongoDB ensures data integrity and consistency.
- **Mongodb schema validation**
- **Schema validation** in **MongoDB** is a feature that allows us to set the structure for the data in the documents of a collection.
- We follow some set of rules, and **validation rules**, which ensure that the data we insert or update follows a **specific predefined schema** and ensures the data must have only specific datatypes, required fields, and validation expressions mentioned in the predefined schema.
- When we create a collection for the first time and we want it to meet specific criteria then we can define the collection with the schema validation rules.
- These validation rules can include specifying the required fields we want, and the datatype for those fields, and also allow the user's custom expressions. We use the command **$jsonSchema** for specifying the rules.

When to use Schema Validation

- **Schema Validation** is like setting rules for **how your document must look in our database.**
- When we are experimenting on a new application in which we think that the incoming data might change the application's fields and we are unsure about the structure we might not want to use schema validation.

- **Step 1**: We create a collection named of 'students' using the createCollection() command.

- Step 2: With the **'$jsonShema'** command inside the validator we specify the schema validation rules.
- Here with the required property we give a list of fields that every document must have when inserted into the collection.

- **Step 3**: Give all the fields and their datatypes inside the properties.

```
db.createCollection("Students", {
    validator: {
        $jsonSchema: {
            bsonType: "object",
            required: ["name", "id"],
            properties: {
                name: {
                    bsonType: "string",
                    description: "Name must be a string."
                },
                id: {
                       bsonType: "int",
                    description: "id must be an integer."
                },

            }
        }
    }
});
show collections;
db.Students.insert({name:"s",id:1})
db.Students.find()
```

CRUD: Querying

- Done on collections.
- Get all docs: db.<collection>.find()
  - Returns a cursor, which is iterated over shell to display first 20 results.
  - Add .limit(<number>) to limit results
  - SELECT * FROM <table>;

- Get one doc: db.<collection>.findOne()

CRUD: Querying

To match a specific value:

db.<collection>.find({<field>:<value>})

“AND”

db.<collection>.find({<field1>:<value1>, <field2>:<value2>
})

SELECT *

FROM <table>

WHERE <field1> = <value1> AND <field2> = <value2>;

CRUD: Querying

OR
db.<collection>.find({ $or: [<field>:<value1><field>:<value2>]})

SELECT *
FROM <table>
WHERE <field> = <value1> OR <field> = <value2>;

Checking for multiple values of same field

db.<collection>.find({<field>: {$in [<value>, <value>]}})

CRUD: Querying

Including/excluding document fields

db.<collection>.find({<field1>:<value>}, {<field2>: 0})

SELECT field1
FROM <table>;

db.<collection>.find({<field>:<value>}, {<field2>: 1})

Find documents with or w/o field

db.<collection>.find({<field>: { $exists: true}})

Find Query Options

•The sort() method orders the documents in the result set.

• The following operation returns documents in the collection sorted in ascending order by the name field:

•sort() corresponds to the ORDER BY statement in SQL.

```
db.bios.find().sort( { name: 1 } )
```

- Limit the Number of Documents to Return
- The limit() method limits the number of documents in the result set. The following operation returns at most 5 documents in the collection:
- db.bios.find().limit(5)
- limit() corresponds to the LIMIT statement in SQL.

- The skip() method controls the starting point of the results set. The following operation skips the first 5 documents in the bios collection and returns all remaining documents:
- db.bios.find().skip(5)

- **Combine Methods**
- The following statements chain cursor methods limit() and sort():
- db.bios.find().sort({ name: 1 }).limit(5)
- db.bios.find().limit(5).sort({ name: 1 })

Thanks