mongoDB

{ name: mongo, type: DB }

Agenda:

- Difference between NoSQL and SQL DB.
- When and when not to use NoSQL.
- Pros and cons of NoSQL DB.
- Introduction to mongoDB.
- Installation.
- Collections and documents.
- Data modelling
- Embedded document and Document Referencing
- Basic query -> Creating DB, Collection, Documents -> get/update/delete document

Introduction to mongoDB

- MongoDB is a document database that provides high performance, high availability, and easy scalability.
- Document Database
- Documents (objects) map nicely to programming language data types.
- Embedded documents and arrays reduce need for joins.

- **High Performance**
  Embedding makes reads and writes fast.
  Indexes can include keys from embedded documents and arrays.

- **High Availability**
  Replicated servers with automatic master failover.

- **Easy Scalability**
  Automatic sharding distributes collection data across machines.
  Eventually-consistent reads can be distributed over replicated
  servers.

Latest Stable version of mongodb

l4.0

To get started, there are six simple concepts we need to understand

- MongoDB has the same concept of a database with which you are likely already familiar (or a schema for you Oracle folks). Within a MongoDB instance you can have zero or more databases, each acting as high-level containers for everything else.
- A database can have zero or more collections . A collection shares enough in common with a traditional table that you can safely think of the two as the same thing.
- Collections are made up of zero or more documents .Again, a document can safely be thought of as a row .

- A document is made up of one or more fields , which you can probably guess are a lot like columns .

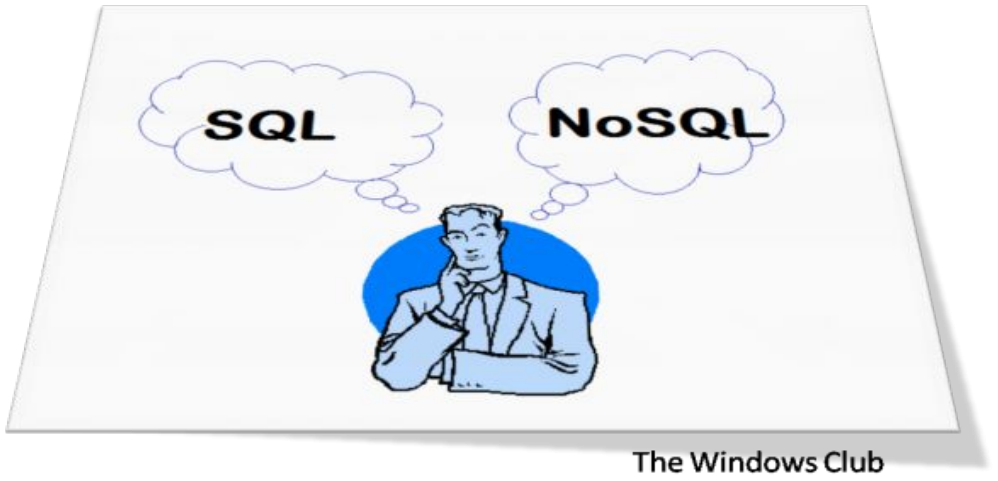
- Indexes in MongoDB function much like their RDBMS counterparts.

- When we ask MongoDB for data, it returns a cursor, which we can do things to, such as counting or skipping ahead, without actually pulling down data.

Counterparts from SQL world

- Table become Collection
- Row become Document
- Column become Field

The core difference comes from the fact that relational databases define columns at the
table level whereas a document-oriented database defines its fields at the document level.



Collection

- A collection is the equivalent of an RDBMS table.
- Collections do not enforce a schema and is a grouping of MongoDB documents.
- For example “student” collection containing several documents for each student.

Documents

MongoDB documents are composed of field-and-value pairs and have the following structure:

```
{
   field1: value1,
   field2: value2,
   field3: value3,
   ...
   fieldN: valueN
}
```

NOTE: Field names are strings. Field names cannot contain null characters, dots (.) or dollar signs ($).Each document within a collection can have its own unique set of fields .

CRUD

C
CREATE
R
READ
U
UPDATE
D
DELETE

Create Collection

- db.createCollection(name, {<options>})

- For example:
  db.createCollection(“demo”).

Insert

- db.collection.insert(document)
- For example:
  db.unicorns.insert({name: 'Aurora', gender: 'f', weight: 450})

- NOTE:The insert() method has the following behaviors:

- If the collection does not exist, then the insert() method will
- create the collection.
- If the document does not specify an _id field, then MongoDB will add
- the _id field and assign a unique ObjectId for the document before inserting.
- Most drivers create an ObjectId and insert the _id field, but the mongod will
- create and populate the _id if the driver or application does not.
- If the document specifies a new field, then the insert() method inserts the
- document with the new field. This requires no changes to the data model for the
- collection or the existing documents.

Read

- Find() command is a very useful command for retrieving data on the basis of some condition or selection criteria.

  Syntax:

  ```
  db.collection.find(<criteria>, <projection>)
  where criteria and projection both are
  optional.
  It returns a cursor to the documents that
  match the query
  criteria.
  ```

- The projection parameter takes a document of the following form:

  { field1: <boolean>, field2: <boolean> ... }
- The <boolean> value can be any of the following:-

- 1 or true to include the field. The find() method always includes the _id field even if the field is not explicitly stated to return in the projection parameter.
- 0 or false to exclude the field.

- NOTE: A projection cannot contain both include and exclude specifications, except for the exclusion of the _id field.
- In projections that explicitly include fields, the _id field is the only field that you can explicitly exclude.

Query for Equality

```
db.unicorns.find({gender:
'm'})
```

Query for Ranges

- $gt (greater than)
- $gte (greater than equal to)
- $lt (less than)
- $lte(less than equal to)
- $ne (not equal to)

For example:

```
db.unicorns.find({gender: {$ne: ’f’}, weight: {$gte: 701}})
```

Query Using Operators

- $in
- $or
- $and
- $exists

Update

- Modifies an existing document or documents in a collection.
  Syntax: db.collection.update(query, update, options)

  Options:
- 1. **upsert**: If set to true, creates a new document when no document matches the query criteria.
  The default value is false, which does not insert a new document when no match is found.

- 2. **multi**: If set to true updates multiple documents otherwise only first one is updated by default.

$set operator

- $set modifier is used to replace the value of a field to the specified value.
- If the field does not exist, the $set operator will add the field with the specified value.

  For example:
  db.unicorns.update({name: ’Rooooodles’}, {$set: {weight: 590}})

Some other useful operators

- $inc
- $push
- $addToSet

Ordering

```
db.unicorns.find().sort({weight: -1})

NOTE: We specify the fields we want to sort on,
using 1 for ascending and -1 for descending.
```

Exercise

1. Create a Database named 'MYWall'
2. Create two collections named 'user and "post'
3. Insert some post related information like post_id, title, description , posted_by, comments, created_at etc…
4. Comments should be an array of objects which has comment_id, message, timestamp
5. Write query to add a new comment for a post
6. Write query to update description of a post
7. Write query to update comment message
8. Write query to remove a comment

Pagination

Paging results can be accomplished via the limit and skip cursor methods.

For example:

```
db.unicorns.find().sort({weight:
-1}).limit(2).skip(1)
```

Count

```
db.unicorns.count({vampires: {$gt: 50}})
Or
db.unicorns.find({vampires: {$gt: 50}}).count()
```

Remove

- db.collection.remove(query, justOne)
- Removes documents from a collection.

- db.collection.drop()
- Called on a collection to drop it from the database.

Arrays and Embedded Documents

MongoDb turns out to be incredibly handy when dealing with many-to-one or many-to-many relationships.

Example:

```
db.employees.insert({_id: ObjectId("4d85c7039ab0fd70a117d733"),
 name: 'Siona',
 manager: [ObjectId ("4d85c7039ab0fd70a117d730"),
              ObjectId("4d85c7039ab0fd70a117d732")]
 })

db.employees.insert({_id: ObjectId("4d85c7039ab0fd70a117d734"),
 name: 'Ghanima',
 family: {mother:'Chani',
 father: 'Paul', brother: ObjectId("4d85c7039ab0fd70a117d730")}
})
```

Thank You!!