* **MongoDB vs Cassandra vs Neo4J**
* **MongoDB**
* **Replica sets**

MongoDB supports replication of data across multiple servers with the creation of replica sets. A replica set contains the following members:

* A single primary instance, or *primary*, which contains the master copy of the data set. When a client writes new data to MongoDB, it writes that data to the primary.
* One or more secondary instances, or *secondaries*, which maintain additional copies of the primary's data set. Secondary are [eventually consistent](https://docs.mongodb.com/manual/reference/glossary/#term-eventual-consistency) with the primary.
* *Primary is majorly used for write purpose (serves as master) and secondary servers are majorly served for read purpose.*
* [Sharding](http://docs.mongodb.com/manual/sharding) is a database partitioning approach used by MongoDB to store data across multiple machines. By horizontally scaling your data across many servers, you are no longer limited by the read/write throughput and storage capacity of a single server
* **MongoDB: save() vs insert()  
  save** is a wrapper for update and **insert**. Functionally, **save** and **insert** are **very similar**, especially if no \_id value is passed. However, if an \_id key is passed, **save**() will update the document, while **insert**() will throw a duplicate key error
* A **collection** may store documents those who are not same in structure. This is possible because **MongoDB** is a Schema-free database. In MongoDB, you don't need to create collection. MongoDB creates collection automatically, when you insert some document.  
  show collections: shows collections in current db
* **Commands**use cs441project: switched to db cs441project  
  db: currently selected db
* created database (mydb) is not present in list. To display database, you need to insert at least one document into it  
  db.movie.insert({"name":"tutorials point"})
* show dbs: show databases
* db.dropDatabase(): drop selected db
* db.createCollection("cs441test", {autoIndexId:true})
* db.COLLECTION\_NAME.find(): select all documents from a collection
* db.mycol.find().pretty(): display results in formatted way
* db.javaCollection.find({ "forks\_count": { $gt:0 } }).count(): search using a specific property:
* db.cCollection.count() + db.cppCollection.count() + db.csharpCollection.count() + db.goCollection.count() + db.htmlCollection.count() + db.javaCollection.count() + db.javascriptCollection.count() + db.phpCollection.count() + db.pythonCollection.count() + db.scalaCollection.count()
* db.javaCollection.find({ "forks\_count": { $gt:501} }, {html\_url:1}): only retrieve html\_url in returned object, but it still contains \_id. To remove \_id: go to next query
* db.javaCollection.find({ "forks\_count": { $gt:501} }, {html\_url:1, \_id:0})

\_id:0 supresses \_id in the results

* db.usersCollection.find({ "followers": { $gt:51} }, {followers:1, \_id:0})
* Object.keys(db.usersCollection.findOne()): get the collection schema/keys
* db.cCollection.find({ "forks\_count": { $gt:4 } }).count()+ db.cppCollection.find({ "forks\_count": { $gt:4 } }).count()+ db.csharpCollection.find({ "forks\_count": { $gt:4 } }).count()+ db.goCollection.find({ "forks\_count": { $gt:4 } }).count()+ db.htmlCollection.find({ "forks\_count": { $gt:4 } }).count()+ db.javaCollection.find({ "forks\_count": { $gt:4 } }).count()+ db.javascriptCollection.find({ "forks\_count": { $gt:4 } }).count()+ db.phpCollection.find({ "forks\_count": { $gt:4 } }).count()+ db.pythonCollection.find({ "forks\_count": { $gt:4 } }).count()+ db.scalaCollection.find({ "forks\_count": { $gt:4 } }).count()

**get from all the collections**

* **MySQL**
* Not very useful: frequency of commits vs popularity:

SELECT ar.repoID, tr.numberOfCommits, datediff(ar.updatedAt, ar.createdAt) as duration, ((datediff(ar.updatedAt, ar.createdAt))/tr.numberOfCommits) as frequency, ar.updatedAt, ar.createdAt, (ar.forksCount + ar.watchersCount) as popularity FROM toprepocommitstable as tr, alllanguagerepotable as ar WHERE tr.repoID = ar.repoID

ORDER BY `popularity` DESC

* User’s followers and his repo watchers relation  
    
  SELECT user.userID, user.followersCount, lang.watchersCount

FROM alllanguagerepotable as lang, usertable as user

WHERE lang.ownerID = user.userID

ORDER by lang.watchersCount

* Number of lines in toprepolanguage vs openissues in alllanguagereport  
    
  SELECT tr.repoID, SUM(tr.numberOfLines), ar.openIssue

FROM toprepolanguagetable as tr, alllanguagerepotable as ar

WHERE tr.repoID = ar.repoID

GROUP BY tr.repoID

ORDER BY SUM(tr.numberOfLines) DESC

LIMIT 50

* Get userId, repoId and similarity for Scala Spark recommendation:

SELECT ar.ownerID, ar.repoID, (ar.forksCount + ar.watchersCount) as popularity

FROM alllanguagerepotable as ar

WHERE ar.repoID IN ( SELECT toprepolanguagetable.repoID FROM toprepolanguagetable WHERE LANGUAGE='java' )

ORDER by (ar.forksCount + ar.watchersCount) DESC

* Sum of openissues per language

SELECT tr.language, SUM(ar.openIssue)

FROM alllanguagerepotable as ar, toprepolanguagetable as tr

where tr.repoID = ar.repoID

GROUP by tr.language

* **Troubleshooting**

1. create user 'testuser'@'localhost' identified by 'password';

2. **Problem:** MongoDB: not authorized on admin to execute command { listDatabases: 1.0 }

**->** Tried  
  
$ db.grantRolesToUser("admin",["readWrite"]) // no luck

2016-11-30T06:09:13.525+0000 E QUERY [thread1] Error: not authorized on admin to execute command { grantRolesToUser: "admin", roles: [ "readWrite" ], writeConcern: { w: "majority", wtimeout: 30000.0 } } :

\_getErrorWithCode@src/mongo/shell/utils.js:25:13

DB.prototype.grantRolesToUser@src/mongo/shell/db.js:1473:19

@(shell):1:1

$ db.createUser(... {... user: "admin",... pwd: "password",... roles: [ { role: "root", db: "admin" } ]... }... );

// no luck

2016-11-30T06:08:05.668+0000 E QUERY [thread1] Error: couldn't add user: not authorized on admin to execute command { createUser: "admin", pwd: "xxx", roles: [ { role: "root", db: "admin" } ], digestPassword: false, writeConcern: { w: "majority", wtimeout: 30000.0 } } :\_getErrorWithCode@src/mongo/shell/utils.js:25:13DB.prototype.createUser@src/mongo/shell/db.js:1267:15@(shell):1:1

Started following <http://www.codexpedia.com/devops/mongodb-authentication-setting/> after reading this thread (<http://stackoverflow.com/questions/23943651/mongodb-admin-user-not-authorized>)

**Solution**: from <https://docs.bitnami.com/google/components/mongodb/#how-to-connect-to-the-mongodb-database>

Edit the */opt/bitnami/mongodb/mongodb.conf* file and comment the following line  
setParameter = enableLocalhostAuthBypass=0

$ sudo mongo

* db = db.getSiblingDB('admin')

admin

db.createUser( { user: "admin", pwd: "new\_password", roles: [ "readWriteAnyDatabase", "userAdminAnyDatabase", "dbAdminAnyDatabase", "clusterAdmin" ]} )  
  
Successfully added user: {

"user" : "admin",

"roles" : [

"readWriteAnyDatabase",

"userAdminAnyDatabase",

"dbAdminAnyDatabase",

"clusterAdmin"

]

}

3. **Problem:** scala Exception in thread "main" com.mysql.jdbc.exceptions.jdbc4.CommunicationsException: Communications link failure The last packet sent successfully to the server was 0 milliseconds ago. The driver has not received any packets from the server

-> Tried all the solutions mentioned here but no luck, still wondering what can be the reason.  
<http://stackoverflow.com/questions/2983248/com-mysql-jdbc-exceptions-jdbc4-communicationsexception-communications-link-fai>

-> **Solution**: Finally, we decided to use Google Cloud SQL, and followed these directions to access the database remotely.

<https://cloud.google.com/sql/docs/mysql-client>