X6412.4LST, the operation system we have chosen to use for both databases.

All SQL queries were done on the relational database management system MySQL 5.5. For a comprehensive overview we present the CRUD operations and the indexes' creation summarized in table 1.

TABLE 1 CRUD Operations

Operation	**MySQL**	**MongoDB**
Schema creation	CREATE SCHEMA `BlogDB`	mongo BlogDB
Table creation	CREATE TABLE IF NOT EXISTS `BlogDB`.`users` (`id` INT NOT NULL, `first_name` VARCHAR(64) NULL , `last_name` VARCHAR(45) NULL , PRIMARY KEY (`id`));	Creation at first insert:db.articles.insert ({ user_id: "1", first_name: "Ciprian", last_name: "Truica" }) Explicit: db.createCollection("articles")
Add a new column	ALTER TABLE USERS ADD JOIN_DATE DATETIME	Does not exist
Delete a column	ALTER TABLE USERS DROP COLUMN JOIN_DATE	Does not exist
Create an index	CREATE INDEX IDX1 ON USERS(first_name)	db.articles.ensureIndex({dirst_name : 1})
Create a compound index	CREATE INDEX IDX2 ON USERS(first_name, last_name)	db.articles.ensureIndex({ first_name : 1, last_name : -1 })
Drop a table	DROP TABLE USERS	db.articles.drop()
Insert	INSERT INTO USERS(id, first_name, last_name) VALUES (1, "Ciprian", "Truica")	db.articles.insert({ _id: "1", age: 45, status: "A" })
Select	SELECT * FROM USERS	db.articles.find()
Select fields	SELECT frist_name, last_name STATUS FROM USERS	db.articles.find({ }, { first_name: 1, last_name: 1 })
Select with where	SELECT u.first_nameFROM `BlogDB`.`users` AS uWHERE u.id = 1;	db.articles.find({user_id:"1"}, { "first_name" : 1});
Ordered Select ASC	SELECT * FROM USERS ORDER BY USER_ID ASC	db.articles.find({}).sort({user_id : 1})
Ordered Select DESC	SELECT * FROM USERS ORDER BY USER_IDDESC	db.articles.find({}).sort({user_id: -1 })
Select with count	SELECT COUNT(*) FROM USERS	db.articles.count()
Explain	EXPLAIN SELECT * FROM USERS	db.articles.find().explain()
Update	UPDATE `BlogDB`.`articles` SET title="MongoDB" WHERE id = 1;	db.articles.update({_id: "1"}, $set : { "article.title": "MongoDB" }}, {upsert: true});
Delete	DELETE FROM USERS	db.articles.remove( )
Delete using where	DELETE FROM USERS WHERE id=”1”	db.articles.remove({ _id: "1" })
Delete a table from dictionary	DROP TABLE `BlogDB`.`articles`	db.articles.drop()

V. Conclusions

MongoDB is a very flexible, schema-less database that that can be implemented in a distributed architecture. “MongoDB was build for the cloud” developers boast. MongoDB can scale horizontally using Sharding. Data in a collection can split across multiple shards. Also MongoDB provides build in load balancing; data is duplicated to keep the system up and running in case of a failure. From the point of CRUD operations this fact is not seen, data will be manipulated the same and to interrogate a distributed MongoDB system will not need any other query methods. Indexes full potential is seen in a distributed system.

Their main role is to help read queries perform fast. Although adding secondary indexes build more overhead in storing documents their B-tree structure is very helpful of keeping track of data that is split and stored on different servers.

MongoDB supports master-slave replication. From the point of view of the CRUD operations they are not influenced in any way by the number of slaves servers a master server has. In MongoDB there is no use of a JOIN operation. Documents can be nested inside other documents. Using the no normalization encourages data redundancy, an idea not shared by most developers due to the fact that this can create confusion in a database regarding the way records are store. But using a schema-less design comes in handy when using CRUD operations; they are more natural to write and they are easier to understand at a first glance.

MongoDB is a more rapid database management system. If you want a simple database that will respond very fast, this is the choice you should make[3]. To achieve scalability and mush higher performance, MongoDB gave up ACID(Atomicity, Consistency, Isolation, Durability) transaction, having a weaker concurrency model implemented known as BASE (Basically Available, Soft state, Eventual consistency). This means that updates are eventually propagated to all nodes in due time.

In conclusion, if a developer wants to build a web application that is fast and flexible, than MongoDB is the right choice. If the application designer’s main concern is the relation between data and to have a normalized database that uses ACID transactions, then the right choice is a classic relational database.

VI . Acknowledgments.

The research presented in this paper was partially performed under the European FP7 project ERRIC (http://www.erric.eu).

References

[1] E. Redmond and J. R. Wilson, “Seven Databases in Seven Weeks: A Guide to Modern Databases and the NoSQL Movement”, 2012

[2] K. Banker, “MongoDB in action”, 2011

[3] A.Boicea, F. Rădulescu, and L.I. Agapin, “MongoDB vs. Oracle - database comparison”, *The 3-rd Conference EIDWT*, Bucharest, 2012

[4] http://www.mongodb.org/