Olga Kushanova

BUILDING, TESTING AND EVALUATING DATABASE CLUSTERS

OSA project

Bachelor's Thesis
Information Technology

May 2014

MAMK
University of Applied Sciences

DESCRIPTION

MAMK University of Applied Sciences	**Date of the bachelor's thesis** 7.05.2014
Author(s) Olga Kushanova	**Degree programme and option** Information Technology

Name of the bachelor's thesis

BUILDING, TESTING AND EVALUATING DATABASE CLUSTERS

Abstract

The purpose of this study was to research idea and functionality of clustered database systems. Since relational databases started to lose their functionality in modern data size and manipulation a new solution had to be found to overcome the limitations. On one side the relational databases started to support clustered implementations, which made the database more reliable and helped to achieve better performance. On the other side, a totally new data store structure came with NoSQL movement. From the beginning NoSQL was developed as a system which can be spread across multiple servers therefore it supports clustering in its structure.

Another aim of the study was to give ideas and guidance for OSA project. It is an Open Source Archive project carried in MAMK, which goal is to create a functional long term data store – dark archive with open source tools. For the project this thesis presents guidance for building and designing clustered database system.

The thesis research was carried out with practical implementation of different clustered solutions. There have been viewed and analysed four different systems presenting SQL and NoSQL cluster types. From relational databases MariaDB Replication and Galera clusters were built and for NoSQL side document-oriented MongoDB Replica Set and Sharded cluster. All the systems for testing and evaluation have been built independently from their initial state. Tests were mainly focused on cluster implementation, failover solution, availability, backup system and performance.

The results of the project have shown that clustered database proves to be a way for a system to support massive data, high user access and data recovery. Although in an open source environment building and configuring a cluster can bring challenges and take time to implement it.

This work has cleared out some points for MariaDB and MongoDB clusters implementation solutions. However there lots more elements to consider for cluster building and in future for NoSQL into SQL implementation.

Subject headings, (keywords)

database, DBMS, RDB, NoSQL, cluster, CAP, sharding, replication, MariaDB, MongoDB

Pages 78	**Language** English	**URN** Leave blank

Remarks, notes on appendices

Tutor Matti Juutilainen	**Employer of the bachelor's thesis** Mikkeli University of Applied Sciences (change to a company name, if applicable)

CONTENTS

1 INTRODUCTION

Upon the years not only in computer existence a question of storing data was raised. Through the evolution of data manipulation and application development scale of data changed dramatically. The topic of this thesis is focused on a subject of database environment and its evolutionary way along the years. In this paper Relational Database with SQL are viewed, its successor NoSQL group, their developments and reasons for creation. The aim of the study is to get familiar with an open sourced solution for clustered databases and evaluate them.

Idea of this thesis is a part of the project held in MAMK – Open Source Archive. The main goal of the project is to build an environment which would be able to behave as a dark archive – a long term data preservation storage. The project plans to offer the dark archive for usage to MAMK and partners. This research covers the topic how the archive is going to handle data from user into the archive. From the archive point of view users could store any type of data inside therefore schema archive cannot satisfy all the input data needs. Additionally when in the future archive grows one single server database will not be enough for sufficient service delivery. Thus the project need to find ideas about clustered database solutions, their implementation and performance.

The research itself is fully based and built with open source software. Reason for that is to follow the project structure and idea about using freeware open source for more independent technology and the way how to project the costs of system in the future.

Research questions:

- Why the idea of a database is changing
- How different are types of modern databases
- How users and companies can adopt to the change
- Give recommendations and guidance for OSA project
- See performance of an open-source based database systems

What topics this thesis does not cover. Although here the nature of databases is covered this work does not show full mathematical scale origin of a database. Neither this work provides a reader with full structures of specific databases. In addition even though in this paper examples of queries and database manipulations are presented, they do not cover full database administration/user aspect.

Research structure:

The study progresses for the theoretical part is built in a chronological order since flat system into the newest graph database.

In chapter 2 part 2.1 a database as a term is explained. Inside there contains information important to understand in order to follow the main subject of the work. Additionally, the development of database system and reasons for creating it in a first place are explained. Parts 2.2 and 2.3 are fully dedicated to a topic of relational databases. Inside there the main points of SQL database operations are stated and usage of this type of a data store for different purposes are shown. Part 2.4 is introducing NoSQL database family and its 4 different types. 2.5 shows challenges of NoSQL. In 2.6 the best practices examples can be viewed - why enterprises move to NoSQL and how it affects performance and solves their challenges.

Chapter 3 introduces classification of different database. By introducing theorems CAP, ACID and BASE a structured way to understand place of each type of database is presented. Part 4.3 contains a table which has structured data about each database which was selected in testing.

Next chapter 4 presents the idea about clustering. As it is impossible to imagine nowadays a full scale database environment contained only on one machine this chapter plays a big role in this work. In the part 4.2 different types of cluster architecture are presented, how they are built and what it means for the user. Part 4.3 connects the database term with the cluster – meaning for a database to be in a cluster. In there more information about sharding and vertical scaling is explained.

In chapters 5, 6 database solutions, mentioned in the paper are tested and shown their operation. At first MariaDB and MongoDB are tested on a single data server, then each is built in a cluster. The practical part is mostly oriented on building the clustered systems, including step by step setup, testing, taking measurements, finding and pointing out a solution for OSA project purposes.

Chapter 7 – conclusions - summarizes the whole research process. It presents the measurements in a table and gives a brief information for the solution. In addition it states possibilities for future research and development.

2 DATABASE ENVIRONMENT

This chapter contains information about the basic idea of a database and its definition. Here additional information about a database structure and organization is presented. Additionally, comparison of MySQL and NoSQL models is shown which helps to understand the development of databases and follow the latest (year 2013) database knowledge with an overview of databases and database management systems operations. CAP theory is also explained here as an overview on the existing types and structures of a database systems. In the end of the chapter software tools particularly used in this work are introduced in more detailed.

2.1 Database Basics

A database (DB) means simply a repository of information. It is one of the most important applications for computer for storing and managing information. DBs are primary used in banking for the transactions, airlines - schedules, universities - transcripts, human resources – records and salaries and etc. Data in a DB is organized and can be accessed within a database management system (DBMS) - a collection of different programs which allow a user to organize, administer and monitor DB storage. [1]

Quite usually users mix ideas and terms of DB and DBMS. It can be easily confused as those two terms are closely connected. The whole data, information can be stored in tables, objects or document form which is a DB. In its turn, DBMS gives an interface to access, store and retrieve that data. It can manage data across more than one DB and in many cases it can make analyses on data queries. The main purpose of DBMS is to store and transfer data into information. In order to operate properly a regular DBMS consists of three elements: physical database, database engine and database schema. Among the other functions DBMS provides concurrency, security and data integrity for a database. In this work term database is used for identifying the whole system, not only for the data contained inside. [2] [3] [4]

One of the specified DBMS usage is to make translations (within Open Database Connectivity, ODBC) between different DB's. ODBC standard was created in 1992 and it is supported by a number of different databases e.g. MySQL, Filemaker and others. Every database where ODBC driver is included in, makes it possible for any application (for example web application) to use the same set of commands with different databases. Therefore regardless on to a type of DBMS they can communicate with each other. Examples of DBMS are MySQL, ORACLE, MariaDB and a number of more now existing. [5]

With the growing number of DBMSs there is a certain need to be able to identify a purpose of each one. DBs can be classified according to data contained inside, for example it can be customer information oriented (names, addresses, account numbers, etc.) or library data (accordingly titles, authors and index number). In computer world databases most often are categorized according to an organization how data is stored. [6]

One of the oldest type of DBMS is the simplest card catalog. It must be well-organized and is efficient to use only if structured correctly. Although it is still very difficult to handle a search process in when the DB grows rapidly. Early computer model had a "flat file model" where records were stored one after another as a list of rows. So when a search was performed it always needed to start from the beginning to search sequentially. Such a DBMS was unable to handle massive data efficiently. Therefore a new type of DBMS was needed which could be reliable, fast, random accessed and easily extendable. [6]

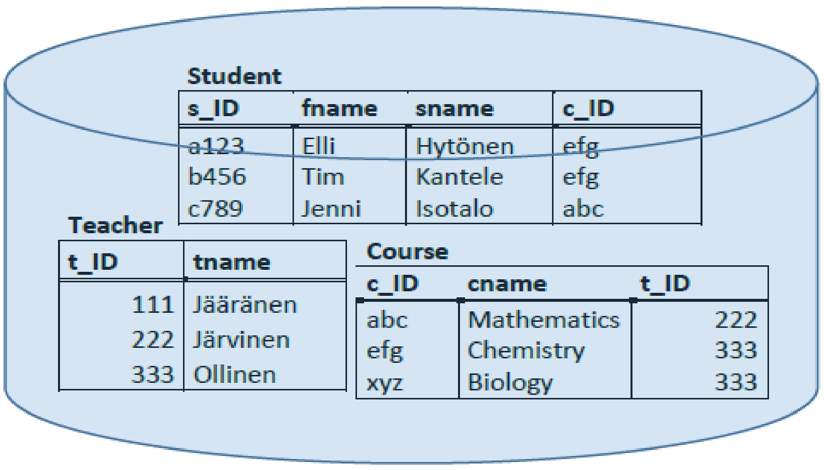
2.2 Relational Model and SQL Traditional View

A traditional and most well-known and by far most widely used type of computer database is a **relational model**. Originally a relational datatabase was developed for storing information for a long period of time. The very first relational database was proposed by Ted Codd (IBM researcher) in 1970. [6] It was a different way from a hierarchical model to store and organize data so that it can be accessed easily. In his seminar paper “a Relational Model of Data Banks” Codd presented a relational database system and defined it with Codd’s 12 rules. Those 13 rules (list is numbered from 0 to 12) give guidance and requirements for a database to perform as a relational one – including definition how data is stored and in which form, null values, views, access rules and etc. [7] [8]

The whole data load in a relational database is stored in tables– collections of rows and columns. Tables can be viewed as organized bodies of relational data. Rows contain unique sets of data and are organized by the column - each column identifies one type of data (real number, data string, integer, etc.). Every table, row or column requires a unique name, usually depends on data which is inside of the table so the search can be conducted easily. For a separate block of tables there exists a primary table where the main data is stored when the rest of data is divided into other sub tables. Different tables are connected by having matching data fields – relationships which a RDBM uses to link tables together. Those relationships are created with the help of **keys**. [9] [10]

A key for a database is a way to identify a record and with it allow a logical representation of information. In relational database some important values can be identified as primary key, foreign key, composite key or candidate key. A **primary key** is a minimal set of fields which can uniquely identify a value inside of one table. Examples of such keys are student number, personal ID, number of a library book, bank account number or any other identification number. It is not possible to have two people with the same ID since it is unique to each individual. Thus, when each table is created it can have only one primary key. At the same time there can be more options how a record can be uniquely identified, it is called a **candidate key**. There can be multiple candidate keys in a table and one of them is the primary one - the best applicable. A key does not always mean only one separate value. If a key is composed of more than attributes then it is called **composite key**. A **foreign key** is another type of key, it is usually a primary key for another table. For example in a school library system a table Book contains a column ‘BookID’ which is a primary key, but for the table Student the same ‘BookID’, book which is borrowed, will become a foreign key. Therefore while single table contains only one primary key at the same time there can be duplicates of a foreign key value. By including a primary key from one table into another one a foreign key is implemented. Thus, when two tables are joined, relationship is established. At first when a table is created keys can be assigned manually to the each column. [8] [9] [10] [11] [12]

An example of a single set of a relational DB is presented on a Figure 1. Such a DB is the simplest model how records at school/university are handled. There exists 3 tables “Student”, “Teacher” and “Course”. For a table Student it can be called “studentid, first name, second name” – relation. There each row contains information about one person with student ID (s_ID), first name (fname), and last name (lname). In this table all unique values are the ones which are in s_ID column. In this case DB sees student ID as a primary key value as it is non-repeatable and unique. For the table Course c_ID is a primary key when at the same time we can see the same column in Student table and that is how foreign key can be seen. Therefore, by viewing Student table user also can learn which course the student is taking. In a real database environment it looks and works more complicated with hundreds of tables and relations.


Student

s_ID	fname	sname	c_ID
a123	Elli	Hytönen	efg
b456	Tim	Kantele	efg
c789	Jenni	Isotalo	abc

Teacher

t_ID	tname
111	Jääränen
222	Järvinen
333	Ollinen

Course

c_ID	cname	t_ID
abc	Mathematics	222
efg	Chemistry	333
xyz	Biology	333

FIGURE 1 Database Example with 3 tables

In Figure 2 such term as a single **schema** is graphically shown. A schema for a DB can be compared to a much specified racks of hard drives. All of them are contained in the same room (DB) but at the same time each rack is unique in its structure and separated by walls, kept contained from outside and every table inside should be following schema rules only. Thus when a table inside of one schema is created it cannot be moved into another one, as each schema identifies specific structure of the tables created inside and cannot be changed. If it is necessary to move a table between schemas, a new table should be formed in desired location and new properties. On Figure 2 a big DB contains multiple schemas inside. A user can work with each of them separately but each schema identifies rules what data can be stored inside. When a login process is performed user should first log in into the DB and then choose a correct working schema. [2] [13] [14]

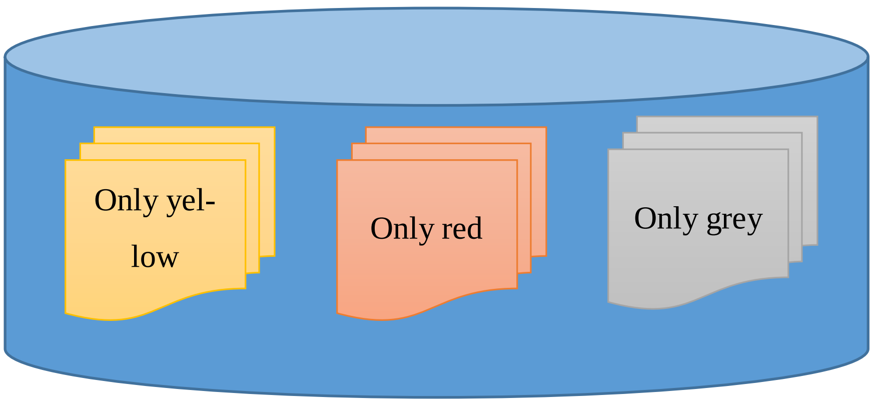


FIGURE 2 Database Schemas

From all DBMS functions manipulating data is primary. Under manipulation is understood adding new data sets, changing and reorganizing. For every request from a database a user writes a specific command, a **query** (meaning *to search, to question or to find*). Queries and requests are usually constructed using specialized database programming language. In 1970 a structured way to access the data and the data relations from the DB was proposed with a name of SEQUEL and later shortened to SQL. SQL is a Data Manipulation Language (DML) – language for accessing and manipulating data organized by the appropriate data model. It is a high-level non-procedural language which is used to communicate with the database. High-level language means that the commands are translated from human to machine language through an interpreter. And a low-level language is very close to computer hardware such as machine code or assembly language. In a non-procedural language a user specifies himself only what data to get but without the way how to retrieve it. On a contrary, a procedural language such as C or C+ defines not only what data to extract but also how to access it. [15]

On Figure 3 an example of SELECT query is presented to identify DML. Such a command is used to extract data from a specific table. In this example a user asks for values under column *sname* from table *student*. This is the simplest example that can

```
SELECT sname
FROM student
;
```

FIGURE 3 Select Query

be presented – more complicated structure can also include different constraints: conditional search, grouping, sorting and even calculation functions.

When a table is created another type of query is taken place. Data Definition Language (DDL) specifies attributes for DB schema. As an example of a CREATE query on Figure 4.This query creates new table with a name Grade, two columns with names s_ID (value of max 4 characters) and grade (integer). DDL statements are used to modify the structure of tables and other objects in the database. DDL is also responsible to handle and specify key constraints called alter table. [1]

```
CREATE TABLE Grade (
            s_ID char(4),
            grade int )
;
```

FIGURE 4 Create Table Query

When a DDL statement is brought into a database DDL compiler generates a set of tables stored in a data dictionary. Additionally, data dictionary contains **metadata** - information about data. Example of metadata can be any attributes of an element: height, weight or author. For the table metadata contains information about the length of the table, number of columns or where the table is located. [16]

2.3 Relational Databases in Practical Usage

Most known corporate developers are IBM with DB2, Oracle, Microsoft SQL Server, MySQL, dBASE. According to research company Gartner, the five leading commercial relational database vendors by revenue in 2011 were Oracle (48.8%), IBM (20.2%), Microsoft (17.0%), SAP including Sybase (4.6%), and Teradata (3.7%). Relational database is the most widely used DB solution and while being the most well-known relational DB is stated to be a mature state DB, reliable and along the years proved to be functional. Most of the developers have proven themselves through years and have kept good solutions, product support. From Open source side most well-known are MySQL Server, Cubrid, Firebird, MariaDB and SmallSQL. [17] [18]

Relational DBs are used on a daily basis mostly by companies and also by private users. Schools, hospitals, government, library, airports, business or banks, they all have database inside of their systems. DBs are used for big purposes - webpages, data storage and huge applications: internet stores, log data files, accounting software, airline registry, medical records for hospitals, statistics, marketing, etc. They also can be seen operational in individual use, personal budgeting, planning or grading. Most companies use relational DBs for simple data retrieve/storage. [9] [19] [20] [21]

Any sized company or organization can use a relational DB. There are couple of cases where DB can be effective and convenient to use:

- Data is more or less stable. Tables are not changing constantly and the data is mostly steady.

- Data set is from small to medium size. Although it is very hard to identify perfect data set size for best relational DB operation data cannot be too massive (no bigger than about few hundred million records).

- Data transactions are taken place.

- There is only a slightest possibility for sudden future grows. Relational database cannot be scaled “on-a-flow”, thus fast changes can bring system into non-operational state. Besides it will take time and effort flow to change whole database schema.

- System is expected to be least distributed. Here means cloud computing and relational database performance.

When a typical relational DB could hold from 10 to 1,000 tables it makes a significant difference in searching process and queries executing. Additionally, each table in its place contains a column or columns that other tables can key on to gather information from other tables. When a data load becomes substantial more relations are established, which means more key, rows, columns and tables needed. As the growing process continues interrelation and mapping query execution slows down. In the end it is inconvenient to use a system which requires even a slightest delay. For today’s time of online applications and internet dependency an application which uses 5 or 10 minutes for search becomes pointless. [12] Thus, RDB brings limitations for rapid growing services, online applications and big companies with growing data. [22]

At first, with relational model and information so huge to handle, companies started to buy and connect more and faster servers together to partition and distribute the information. Eventually too massive data takes over even the largest server available. [more in 3.2 Cluster in Database Solution]

For such a reason a relational solution exhausted its purposes for many usages. It was the time when new type of storage – reliable, fast and capacious, started to attract developers and followers. Such a solution was introduced to the world as NoSQL

2.4 NoSQL

For years relational databases were used to store data. By the time when web applications came into a high popular usage, the amount of data became so massive that relational model started to lose its value and usability. Most changes in a storage order were triggered by the rise of cloud computing, agile software development and demand to use unstructured data. In order for cloud based system to work properly it needs to act like a whole across multiple servers. For a complex and big SQL it takes time to merge tables and perform distributed joins tables within a cloud. Besides of being fast and reliable modern technology forces a database to be more flexible, set on future development and changes – agile. While an application needs to be changed on a fly it should stay operational and fully accessible at any time. Here is why unstructured data is needed – it allows to work with data even in its incomplete state also with a possibility to be changed in the future. As a response to the changes, NoSQL storage type was developed to support market demands and needs. [23] [24] [25]

NoSQL term stands (according to many sources) for Not Only SQL. For the first, the term was used by Carlo Strozzi in 1998. Although it is discussable whether his project was enough to be called NoSQL as Strozzi used the term for a lightweight but still a relational database. [26] [27] [28]

Later in 2009 the term was picked up again and used by a developer of Last.fm Jon Oskarsson on meeting in San Francisco. At the moment, term NoSQL database systems covers a type of database that is different from relational DB in its performance, data storage, query or any other standard behavior. They are specifically developed to support speed and scale of web type applications. As for the name some communities uses NoSQL as “No to SQL” – a system, which does not use SQL. [26] [27] [28]

An idea and need for developing NoSQL solutions first came as individual cases for companies such as Amazon, Facebook, LinkedIn, Google, Yahoo!, Twitter and others. For Google at first came Big Tables (Column oriented database), GFS (Distributed filesystem), Chubby (Distributed coordination system) and MAP Reduce (Parallel execution system). Those projects and documents inspired open-source developers to create a variety of under NoSQL name projects. [27] [28]

There are four different types of NoSQL categorized by the way data is stored in: Wide Column Store/Column Families, Document Store, Key Value/Tuple Store, Graph Database.

2.4.1 Wide Column Store

The way how column-oriented database stores data in a database is close to relational database – a table. Only that relational model stores values rows by rows when a wide column store keeps values by columns. In a code in Figure 5 two different structures are presented, comparing RDB and Wide Column store. Even from a small dataset the main difference can be seen that in RDB data goes together by the row and for Wide column a column gives full dataset. One more interesting fact that C3 value misses its fruit name and for RDB it can become a complication to query it and modify. For a wide column changing the value is not a problem. [23]

```
{       A1, Apple, 14, 230
        B2, Orange, 22, 410           RDB
        C3,, 32, 120
        D4, Melon, 45, 300 }

{       A1, B2, C3, D4
        Apple, Orange,, Melon                   Wide column
        14, 22, 32, 45
        230, 410, 120, 300 }
```

Figure 5 Wide Column

Advantage of such structure is that any table can be updated simply at any time. When a column is added it is not necessary to fill in every row with a value. In future it gives an opportunity for a table to be flexible and support unexpected grows and changes. At the same time for a number of columns the values stored are about the same length across rows which makes it easier to compress and store data efficiently (for example age or phone number in a user profile). [23]

Wide column data store was especially created for managing and manipulation of very large amounts of data distributed over many machines. Column family is a way to define the structure on the disk and arrange all columns, inside it contains name and keys pointed to the data. As it can be seen on Figure 6, which presents an idea how twitter posts (tweets) are structured, the key identifies a place. Although values are stored by columns keys are important to map a location. [23] [29]

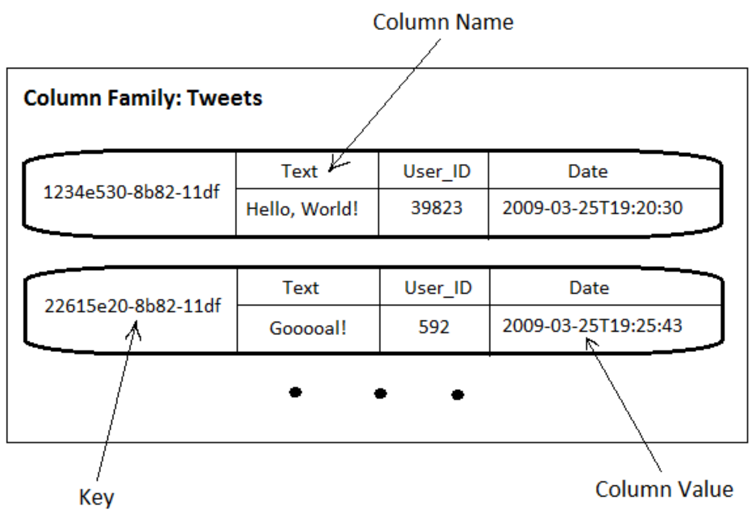


Figure 6 Column Family [49]

Examples of a Column oriented databases are Apache Cassandra, HBase, Google Big Table (Datastore).

2.4.2 Document-oriented

In a document-oriented database data is not restricted by table or a row frame. On the contrary, it is stored in a document type format with specific characteristics, which acts like a row a column itself. Documents are written in JSON – JavaScript Object Notation human-readable and machine easy to parse and generate format, very light-weight; XML (Extensible Markup Language) –markup language for structured information documents; BJSON – binary JSON, an extended form of JSON it allow to work with more data types and encode and decode between different languages.. [31] [32] [33]

With data access over HTTP using RESTful API (resourced based protocol) some document (CouchDB) database provides access to each document through its ID, which can be written inside of the document or embedded in. However, it is not necessary that an ID is included into the document body – it can also be provided within URL location. [28] [34] [36] [37] [39]

For the connection of related data for a relational database foreign keys would be used. In a document-oriented the whole data relation is encapsulated together or stored in a single document. [28] [36] [37] [39]

Document store allows inserting, retrieving and manipulating semi-structured data. In order to perform one document does not need to be completed to function or to follow a specific schema. For example, in a personal account file there can be possibly some data left out, such as a phone number or any other information. Comparing two different documents below, the first one contains the standard document data (document id, version and other properties) and some information about the user – first and last name and date of birth. Although the second user document gives more information (address and phone number) both objects are treated equally in a database. Additionally, even if any other object has no correlation to other documents it is still a part of a database. This feature makes the document-oriented database type very useful in web applications when there are different type of objects by bringing support of rapid changes in future. [22] [23] [28] [36] [37] [39]

Document 1

```
{
  “user”: {
   “id”                : “document-u1id”,
   “version”           : “1.0.0.0”,
   “create_time”       : “2010-10-10T10:10:10Z”,
   “last_update”       : “2012-12-12T12:12:12Z”
  },
  “type”: “user-profile”,
  “personal”: {
   “firstName”         : “Allen”,
   “lastName”          : “Brown”,
   “date_of_birth”     : “11-11-1991”,
  },
…
}
```

Document 2

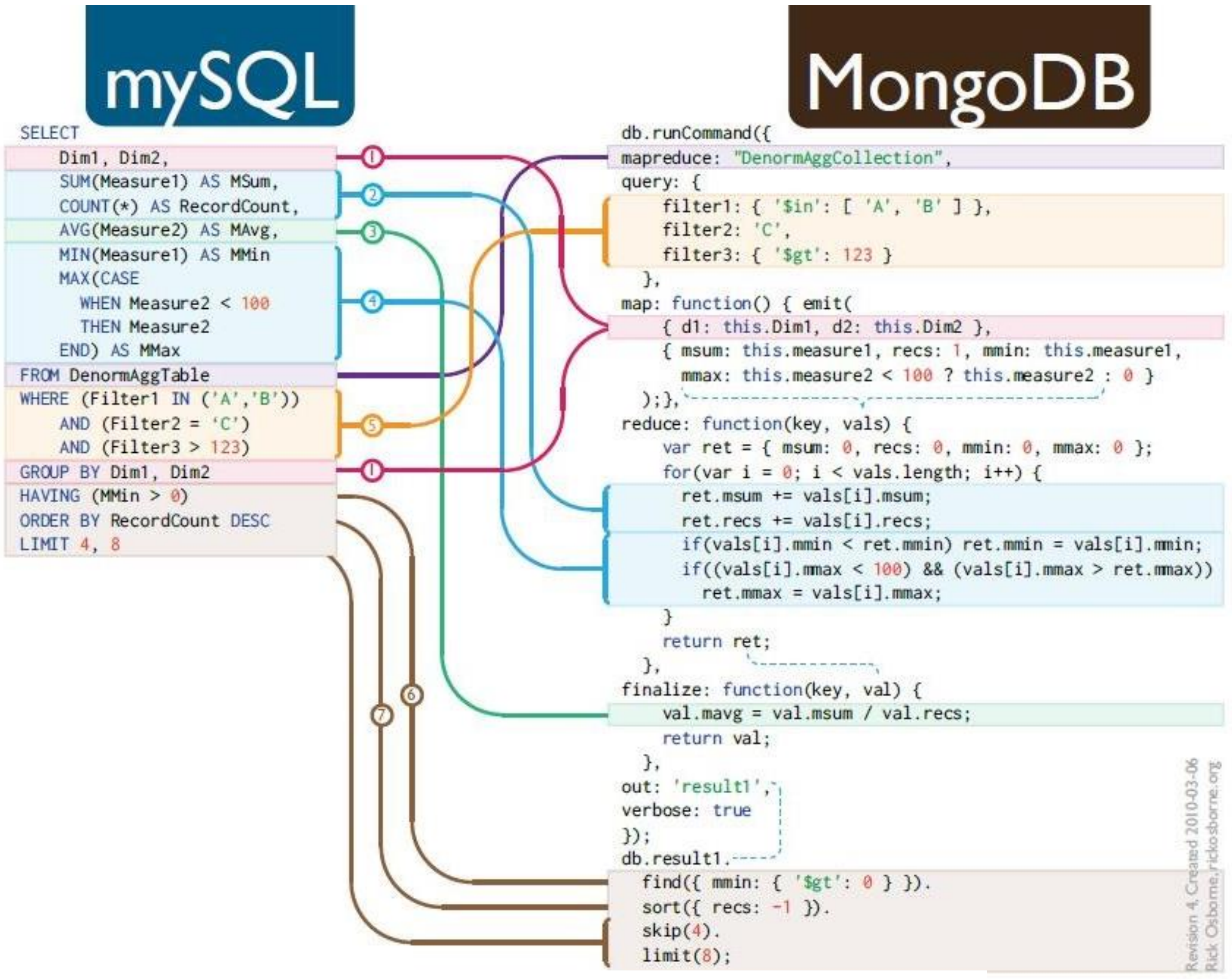
```
{
  “user”: {
   “id”                : “document-u2id”,
   “version”           : “1.0.0.0”,
   “create_time”       : “2010-10-10T10:10:10Z”,
   “last_update”       : “2012-12-12T12:12:12Z”
  },
  “type”: “user-profile”,
  “personal”: {
   “firstName”         : “Allen”,
   “lastName”          : “Brown”,
   “date_of_birth”     : “11-11-1991”,
   “address”           : “123 Database Street 5”,
   “phone_number”      : “123456789”,
  },
…
}
```

Under document-oriented category fall MongoDB, Couch DB, Lotus Notes, Cassandra.

MongoDB is a relatively young open-source database, which development started in 2007. It provides rich index and query support, including secondary, geospatial and text search indexes. Documents are encoded in BJSON for indexing and provides JSON-encoded query syntax for document retrieval. [35] [40]

To show how this specific type of a database operates on Figure 7 a translation between MySQL and MongoDB is introduced and explained. From MySQL side an ordinary select query with some sorting filters, and math functions is shown. In order to understand the functionality and difference between two systems statements are split into major steps. When working with MongoDB place where data stored is called collection: ‘mapreduce: “DenormArggCollection”’ is equal to ‘FROM DenormArggTable’. With the first line standard ‘db.’ user identifies that an action with database is performed. The statement can be followed with a name of collection, or a function, for example createcollection(), insert() or as in our case runCommand(), which is useful when query is expanded.

1. In MySQL data selection is categorized in the begging and in the end of query, after operations is grouped together in the end. MongoDB groups columns at the moment when they are pulled out as a map function. These allows to reduce size of the working set.
2. Standard functions of SUM and COUNT in NoSQL transfer into manual “+= logic”.
3. Aggregates connected with records count and manipulation are performed in finalization state.
4. CASE and WHEN commands transfers into procedural logic operations (here if statement).
5. Filtering has an ORM/ActiveRecord-looking style, every filter unit separated.
6. Aggregate filtering like HAVING is applied to the result set in the end, but not in the map/reduce.
7. ORDER BY = sort(); Ascending 1; Descending -1.



```
SELECT
    Dim1, Dim2,
    SUM(Measure1) AS MSum,
    COUNT(*) AS RecordCount,
    AVG(Measure2) AS MAvg,
    MIN(Measure1) AS MMin
    MAX(CASE
      WHEN Measure2 < 100
      THEN Measure2
    END) AS MMax
FROM DenormAggTable
WHERE (Filter1 IN ('A','B'))
    AND (Filter2 = 'C')
    AND (Filter3 > 123)
GROUP BY Dim1, Dim2
HAVING (MMin > 0)
ORDER BY RecordCount DESC
LIMIT 4, 8
```

```
db.runCommand({
mapreduce: "DenormAggCollection",
query: {
    filter1: { '$in': [ 'A', 'B' ] },
    filter2: 'C',
    filter3: { '$gt': 123 }
  },
map: function() { emit(
    { d1: this.Dim1, d2: this.Dim2 },
    { msum: this.measure1, recs: 1, mmin: this.measure1,
      mmax: this.measure2 < 100 ? this.measure2 : 0 }
  );},
reduce: function(key, vals) {
    var ret = { msum: 0, recs: 0, mmin: 0, mmax: 0 };
    for(var i = 0; i < vals.length; i++) {
      ret.msum += vals[i].msum;
      ret.recs += vals[i].recs;
      if(vals[i].mmin < ret.mmin) ret.mmin = vals[i].mmin;
      if((vals[i].mmax < 100) && (vals[i].mmax > ret.mmax))
        ret.mmax = vals[i].mmax;
    }
    return ret;
  },
finalize: function(key, val) {
    val.mavg = val.msum / val.recs;
    return val;
  },
out: 'result1',
verbose: true
});
db.result1.
  find({ mmin: { '$gt': 0 } }).
  sort({ recs: -1 }).
  skip(4).
  limit(8);
```

FIGURE 7 MySQL to MongoDB [41]

The document based database is completely different from the relational model. It does not support SQL and contains another logic for data manipulation. Document based database does not work with joints or tables. A table is represented as a collection and documents themselves. A document store is getting its popularity because of the structure and the fact that it is a schemaless solution.

2.4.3 Key-value

Partly resembling document-oriented database, key-value store is the simplest type of NoSQL. There is no specified schema forced on the value. As it can be viewed in Table 1 it is a hash table with unique keys and pointers to particular points of data. This structure allows easily scaling of large sets of data.

TABLE 1 Key-Value

key	value
firstName	Allen
lastName	Brown
location	Helsinki

As the database has the easiest structure compared to other NoSQL, it is very fast to perform. But at the same time it does not support vast, complicated requests or calculations. As the database is queried against keys it gives them the best performance in cache memory. [23] [28] [36] [37] [39] Examples of a Key-value database: Voldemort, Redis and Memcached.

2.4.4 Graph Database

Flexible graph model allows to present a database structure as relationships between documents or nodes which are represented as graphs. The whole project is only on a development stage so there are not so many proven solutions in the market. Some examples are Neo4J, InfoGrid, Infinite Graph, Circos. Figure 8 represents the idea how data is organized inside of a graph database.

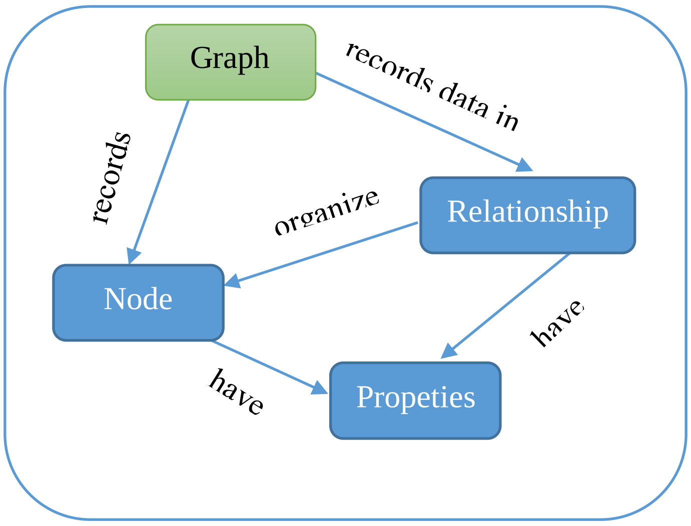


FIGURE 8 Graph database

The best use practices of a graph are transport links, people connections on social networks, and also network topologies. For that type of database to exist there always needs to be a connection between nodes and there can be only one or many relations. The reason for using Graph database for massive, interrelated data is that the database uses shortest path algorithms to make queries more efficient. Compared to relational database where with more relations more keys are added and then connections become heavy and long to process, a Graph database is made to identify relationships between nodes. [23] [28] [36] [37] [38] [39]

2.5 NoSQL Challenges

From this chapter it can be seen that NoSQL does not identify only one specific type of a database - the name represents a change into the idea of storing data, a fully new class. There is a variety of different compatible options to choose from. Which makes it a great alternative to RDB when traditional solution is lacking in scalability, performance and data maintenance (unstructured data). [22] [23] [42] [43]

NoSQL can bring a great deal change into the current data store model. Below are pointed some reasons why it is practical to choose NoSQL solution over Relational model:

- Can handle big massive of data
- Designed to scale
- Flexible – not limited by a specified schema
- Simple structure – easy to implement
- Relatively cheap to install, scale and maintain

Together with all the advantages NoSQL keeps some limitations and challenges which should be considered before installing the system. [22] [23] [42] [43]

Support Often companies which offer services are not at global reach as they are small and open-source. When the system fails there is a chance of not getting a timely support. [22] [23] [42] [43]

Administration One of the development ideas for NoSQL was to create a system to reduce administration work and make it user friendly. Unfortunately, at the moment it takes a certain skill to install it and then later to maintain and support. [22] [23] [42] [43]

Maturity For the companies it can be risky to apply a non-proven solution instead of implemented and tested one. It is a simple logic of an enterprise – they do not want to change an operational system into something they are not completely sure about. [22] [23] [42] [43]

Expertise Is connected to the previous point – there are not as many experts as for RDB. Almost every single developer is still in a learning state. The same is that not all the database are perfectly ready to be used. [22] [23] [42] [43]

Compatibility When all the applications with embedded relational database are fully operational, it might take time to change all the system structure and idea so it can be used together with NoSQL. In most cases it is better to rebuild the system completely. [22] [23] [42] [43]

Misunderstanding What happens now is also very interesting concept - NoSQL gets into a bigger movement when having it is just fashionable. Companies then install and use it without understanding the whole idea or purpose. For example, a company has a database which does not need to be distributed and is contained on a single server. After moving to NoSQL performance will not change (if not becomes slower). [44]

2.6 Best Practices Examples

Facebook created its Cassandra data store to power a new search feature on its Web site rather than use its existing database, MySQL. That particular internet service stores a huge deal of information and the information is usually not static. It changes and varies as users want to delete, update and add more data all the time. According to official facebook statistics information (on Oct. 2013) there are:

- 819 million monthly active users who used Facebook mobile products as of June 30, 2013

- 699 million daily active users on average in June 2013
- Approximately 80% of our daily active users are outside the U.S. and Canada
- 1.15 billion monthly active users as of June 2013 [45]

With such amount of active users texting, sharing, loading pictures, links and videos the data flows with a high speed and changes constantly. According to a presentation by Facebook engineer Avinash Lakshman [Cassandra Structured Storage System over a P2P Network], Cassandra can write to a data store taking up 50GB on disk in just 0.12 milliseconds, more than 2,500 times faster than MySQL. As it is seen the company had to change database strategy in order to assure the best performance. Cassandra project in Facebook combines elements of Google's Big Table and Amazon's Dynamo designs, Apache HBase distributed database. In the end they got strong consistency model, with automatic failover, load-balancing and compression. [44]

Craigslist is a private company which offers specified web services. They provide advertisements in 50 countries around the world, there are sections about jobs offers, sales and housing. They moved over 2 billion archived postings from MySQL into a set of replicated MongoDB servers in documented, schemaless format. With this move the service gained performance and reliability. As their archives are so large and replicated, so any change of a table would take at least couple of month. New and active advertisements are still handled separately in MySQL. The division brought the system into an easier state to handle and flexible for a change. [44]

One more Enterprise who applied MongoDB is the world-wide well known networking company Cisco. When in November 2011 they launched new platform WebEx Social for social and mobile collaboration the company had to think about the changes in the environment. For Cisco it was difficult to handle schema updates on their old relational database, additionally SQL queries took lots of overloading and were difficult to execute. Migration to MongoDB was made to manage user activities and social analytics. MongoDB allowed to accelerate reads from 30 seconds to minimum (some single cases gave such an improvement) tens of milliseconds. [46] [47]

For a conclusion, NoSQL has a variety of different solutions, many of the existing already proven itself useful in a current environment and received their acceptance from big enterprises. Apart from all the positive characteristics of new data system, a part of database community takes the importance of NoSQL very skeptically. Although the input and revolutionary ideas of a new DB concept cannot be denied it is not assured whether NoSQL will become dominion DB environment and RDB will not be used in the future. On the other hand potentially both systems can be used and operate together at the same time. One thing is for certain that traditional SQL is not going away just yet.

3 CHOOSING A DATABASE ENVIRONMENT

At the moment there are hundreds of different DBMS manipulating data. There are so many of them that a user often has no idea which one to choose. Some users rely on big labels such as IBM (handles DB2) or ORACLE. As the name comes with a price others make their choice on assets available. This brings a user to a shorter list of low-cost or open-source options. Although, the last ones do not always mean a dreadful quality and difficult maintenance. [48] [49]

3.1 CAP

In theory the most sufficient way to get a system one need is to recognize and analyze the requirements. Proposed in 2000 Symposium on Principles of Distributed Computing by Eric Brewer a conjecture and later proved by Seth Gilbert and Nancy Lynch theorem that for distributed computing it is impossible to provide simultaneously Consistency, Availability and Partitioning tolerance (CAP theorem). Because of its founder it is also known as “Brewer’s theorem” and it gives a vivid picture about the classification of different DBMS available. [48] [49]

According to the theorem a distributed computer system can have at most two from the CAP factors.

- **Consistency** - having a single up-to-date copy of data. So that every node operates with the same copy. As for distributed system is highly inconvenient to be without consistency, term weak consistency means eventual consistency. In such a system data is replicated across the nodes they are all working with the same object but different versions. The newest version is somewhere on the cluster and eventually every node will learn about it. After that the system reaches the consistent state.

- **Availability** - information can be accessed easily from any node at any time.
- **Partitioning tolerance** - the system will continue working even after a fail over of one element.

The theorem was introduced for developers of new systems so it will be easier to identify the goals and predict the final product. Here CAP theorem is shown as a tool to ease the process of choosing the best fit environment. [48] [49]

For example a database prioritizing **consistency and availability (CA)** is a simple one node system, which stores only one version of data. Relational traditional database is the simplest example, it is not distributed therefore there is no need to store more than one copy of data. [48] [49]

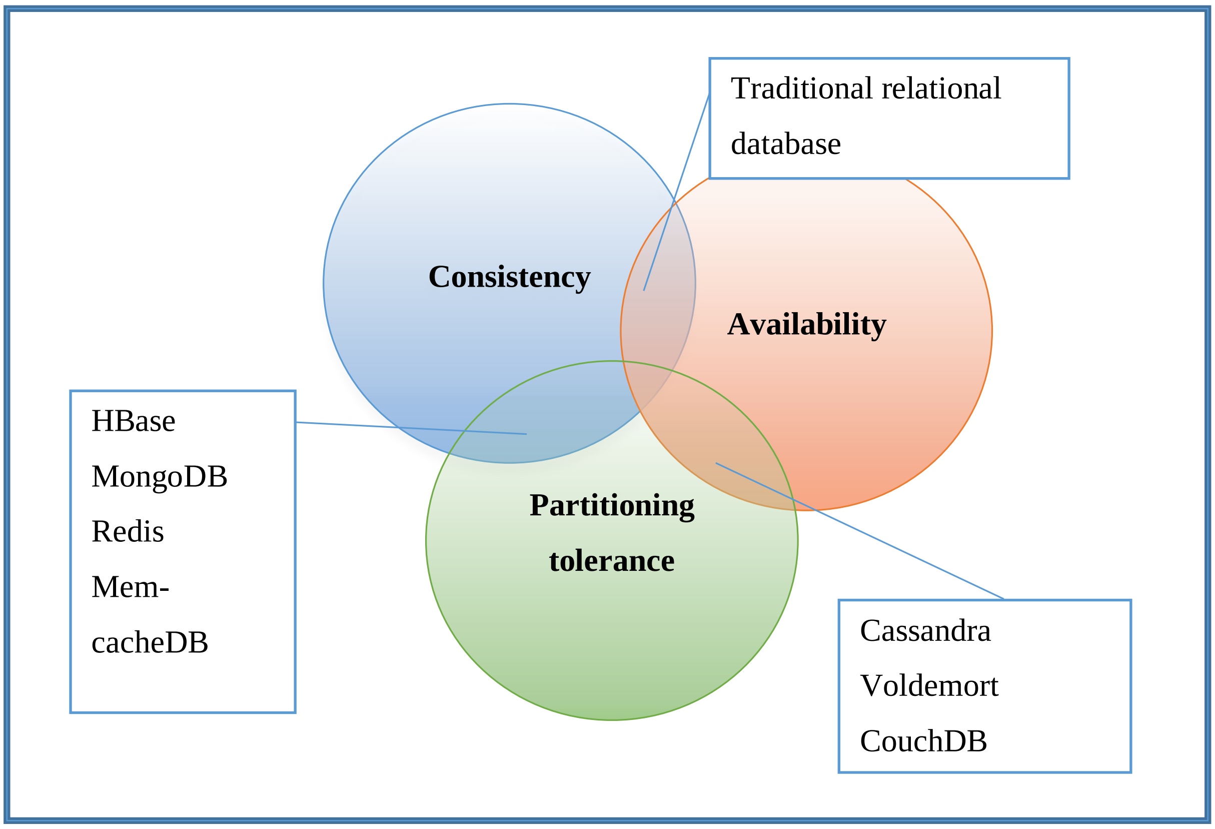


FIGURE 9 GAP theory

AP database brings the system into fully functional, even without any node connectivity it would be possible to attempt operations in a database. For example in a system of two nodes the connection faults, both nodes still should continue working with clients as the system has to be available. As two nodes are working separately without connection they are unable to see each other or exchange data. Until partitioned link is restored the data inside stays inconsistent. Here a weak consistency takes place. [48] [49]

CAP theorem can be misleading for most in a “pick two out of three” concept. From the last examples it can be seen that it does not in any case set consistency to 0% when availability and partitioning tolerance are 100%. It means, that by analyzing the most needed qualities of a database, current database solution are not able to provide all of them at the same time. Therefore every improvement in one sector will cause some interruptions inside the other two components. [48] [49]

Simultaneously CAP theorem itself cannot fully identify all the needs of every DB. There exist other design philosophies for DBs such as ACID and BASE.

3.2 ACID

ACID is used for little less than a DB classification but it shows the requirements for a transaction, a single DB operation. ACID is an acronym for **Atomicity, Consistency, Isolation and Durability**. To identify «**A**» value a DB should be able to follow the rule of all tasks are performed or nothing. In an example of a bank single payment transaction, if one account *a* wants to pay a 100$ so it gets minus that value when on another account *b* where the payment should be transfer will get +100$. If (in any case) the second part of the whole transaction fails the whole operation should be rolled back. For «**C**», **consistency** and hold is the same idea as in CAP theorem for having only one version of the file across the whole system. «**I**» comes with a DB able to handle data locks within isolation levels. In respect of keeping data being concurrent when one operation is performed other operations should not be able to access (have read/write permissions) until the operation is committed (finished). «**D**» is set to bring a DB into a solid state as if an operation was committed it cannot be undone anymore. Although, later it can be changed by the next operation. Durability is handled by the databases by keeping a log file where the history of all the operations is written. Thus, even if server is restarted the transaction log persists. [50]

The whole picture can be explained in a hotel booking example. On a travelling agency or airplane company it is possible to book room. If there are about 100 users searching for a room, while some of them are ordering the room it is not possible to see it for others. And this happened because until the accurate number of inventory is seen for all the users that data set will be locked by the system. This service will work with delays and many customers will get angry and leave the page. To change the system will require to violate integrity of data and can be a wise solution in the end. Amazon sets an operating example of using cached data – users do not see the count at that moment, instead they see a snapshot from couple of hours before. Sometimes there could be mistakes in book numbers in a digital warehouse and some users will not be able to purchase the book. But for a company with a customer base all over the world it is easier to apologize to couple of customers than to lose the client base. [50]

3.3 BASE

The BASE (**B**asically **A**vailable, **S**oft state, **E**ventual consistency) is mostly used as an opposite to ACID concept. As for ACID it is important that data is consistent, a BASE system it does not make any requirements - it sacrifices consistency for availability, which makes it wise to think about for an online application where availability is everything. [50]

Brewer points out in this presentation, there is a continuum between ACID and BASE. One can decide how close to be to one end of the continuum or the other according to system priorities. [50]

3.4 Software Used

In this work open source DBMS are used. Below is a summary table, Table 2, for different types of database storage which were presented in this paper. All four databases MariaDB, MongoDB, Cassandra, Neo4j) which are presented in the table are used in and testing. In the table the main data provided are names of the databases with a quick summary. Main characteristics such as the model type of a database (if it is relational of a part of NoSQL), language it is writer in, the protocol each database uses to access the data, storage type and a description from their vendors, what the database stands for and their aims.

TABLE 2 Used Software Summary

Name	Model/type	Language	Protocol	Storage	Description
MariaDB	One of the relational	C, C++	DDL, DML	tables	“MariaDB is a drop-in replacement for MySQL.” – MariaDB Foundation.
MongoDB	document-oriented	C++	BSON	document memory mapped b-trees	“MongoDB was designed for how we build and run applications today… It is an agile database that allows schemas to change quickly as applications evolve…” – Mongo DB, Inc.
Cassandra and CassandraSE*	Wide Column	Java	HTTP/ REST or TCP/Thrift	memtable/SSTable	"In terms of scalability, there is a clear winner throughout our experiments. Cassandra achieves the highest throughput for the maximum number of nodes in all experiments."- University of Toronto
Neo4J*	Graph	Java	HTTP/ REST	File System Volatile memory	“embedded, disk-based, fully transactional Java persistence engine that stores data structured in graphs rather than in tables" - Neo4j developers

**Originally, all the databases presented in the table were planned to be tested. In this work CassandraSE and Neo4j are only viewed and evaluated for the future development.*

Modern database systems are not viewed as single servers due to the cloud computing and massive data stores. The databases MariaDB and MongoDB used in this work are presented in examples of clustered database implementations.

4 CLUSTER AND DATABASE CLUSTER

This chapter gives an overall view for a term of a cluster in computer and data system. Here also different types of clusters will be shown with simple examples, such as Master-Master relations or Shared-Disk architecture, and how they can be used to optimize a DB system. [51] [52]

Cluster in a computer science can mean and be connected to different aspects of performance and architecture. In overall idea a cluster itself presents a system of computer devices connected together and work together so they all can be viewed as a one whole system. [51] [52]

A database cluster is a specified cluster which is built only for database usage. It is not just storage connected together. A database cluster is designed to be fully operational across distributed environment. Or in other words it is a system which is shared by being simultaneously mounted on multiple servers. [51] [52]

Parallel file system – a type of clustered filed system that spread data across multiple storage nodes. Usually such actions are taken in order to increase redundancy or performance. In this paper the term cluster is to be understand only as a cluster build as a part of DBMS or inside of it. [51] [52]

For a DB system a cluster solution where a full DB keeps the whole information on multiple components connected together makes the system scalable increase availability with lower cost. For every DBs’ working requirements there is in somewhat unique need architecture. [51] [52]

A database build on a top of a cluster can bring a system to faster performance or failover backup. At the same time every development comes with the price, in this case features to sacrifice and some to be put in first order. [51] [52]

4.1 Cluster Types

In a clustered system all the parts should be connected together in order to share the data. The connection is build based on hierarchy within a network line, high-speed connection such a fiber channel and shared disks.

The ways how server nodes can be connected can be classified into two separate groups: disk sharing dependency (Shared-Nothing, Shared Disk) and relation-role dependency (Master-Master, Master-Slave).

4.1.1 Shared-Nothing

Shared-Nothing architecture is the first option presents disk-sharing group. Shared-Nothing brings a full set of data into different separate locations. Those disks are still connected to each other and can be seen as a complete system but each node has its own part of data. On the Figure 10 a big DB with information about Users, Orders and Products is split into 3 equal server "memory cells". Servers can be separated according to type of data stored as on the figure or a geographical location, as Headquarter, South Office or North Office. When such a distribution happens an architect designer has to figure out what king of data is used more in each location. As Orders for example can be mostly used by Accounting Department when Products are needed for a Supplier Manager. In a situation when another node from the cluster needs to access a part of a DB which it does not possess, it makes a request to other nodes in a cluster. After the communication is established and nodes with needed information set are identified the needed part of data can be transferred to needed location. Although every single node can work very fast and efficient with own data, the communication part brings a delay. Thus, such an architecture requires smart partitioning and communication is based on data – shipping. [52]

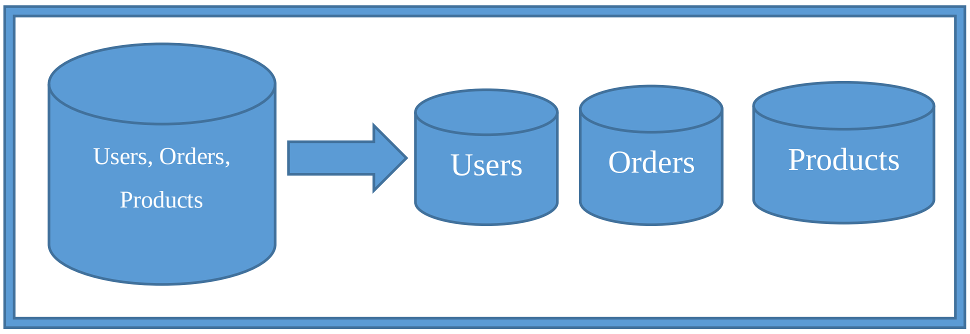


FIGURE 10 Shared-nothing cluster

An example with a company for this architecture is a case where there exists multiple servers across the world for a single web site. There is no master-server and all the servers are equal, however they do communicate but at the same time do not share any memory or disk space. When every node in this system is working as an independent and self-sufficient the data load is separated between them and every additional node makes the system to work faster. [52]

Shared-nothing architecture is also partly known as **sharding**. Sharding is a method breaking data into parts – "shards" and storing them across multiple servers. This technique allows the database to gain more performance and scalability. As big data is spread across multiple machines it is handled with more computing power.

4.1.2 Shared-Disk

Shared-Disk architecture is the second type of what cluster can be. In a Shared-Disk cluster all the nodes have a shared storage space which can be accessed by any node, any time when the data is needed. As it is shown on Figure 11 from one big database the information is moved into a fast, easy accessed storage space. Then as many servers' machines can be connected to the storage (in this example 3) usually the number of servers depends on system requirement. Such a solution brings each node in the cluster to act as if it has its own single collection of data. According to connection and disk speed this architecture designed to bring speed into the system. The users can work with a file at the same time therefore redundancy becomes a weak point. [53]

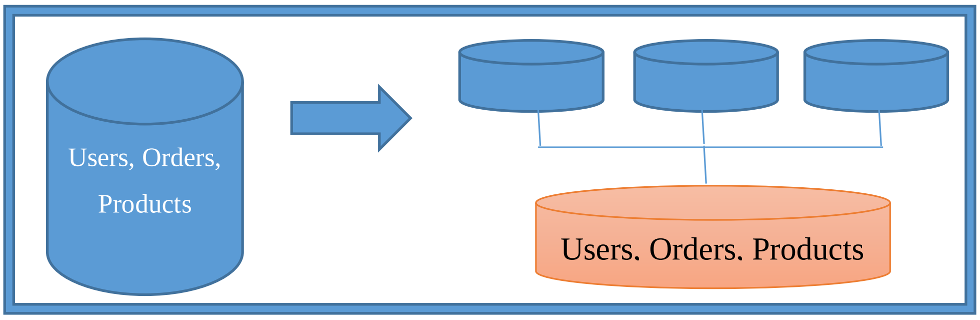


FIGURE 11 Shared disk cluster

For a shared-disk architecture it is also usual to have an inter-nodal messaging, it includes locking, buffering and general status information about the nodes. Those messages are fixed-size and amount grows linearly according to the number of nodes. [53]

Such architecture is easier to set up than shared nothing – it does not require additional partitioning or routing tables. On contrary to SN in future maintenance the system will work freely, without additional partitioning, which is time, performance and money consuming. In most cases shared-disk is used for applications which are dynamic or could have temporal toggling changes. [53]

Table 3 shows the basic differences between two architectures and allows to identify and compare two different architectures. Every row identifies the main opposites of Shared-Disk and Shared-Nothing.

TABLE 3 Shared-disk and shared-nothing comparison

Shared-Disk	**Shared-Nothing**
Quick adaptive in changing workloads	Can exploit simpler, cheaper hardware
High availability	High-volume, read-write environment
Dynamic load balancing	Fixed load balancing
Data need not be partitioned	Data is partitioned across the cluster
Performs best in a heavy read environment	Almost unlimited scalability
Messaging overhead limits total number of nodes	Depends on partitioning, data shipping can kill scalability

Disk and data sharing between the machines are a very efficient way to increase the performance of a database. However, it is not the only way how data can be stored and alone it is not enough to create a fully backed up system.

4.1.3 Master-Slave and Master-Master

A Master-Slave cluster is an example of a role-dependency architecture. As it can be seen from the name there are two main roles –Master and Slave. Master has a full access to data so it can read and write data. When a slave is only a copy of a master state (similar to backup solution) has only granted permission to read the data. Thus, as it is shown on Figure 12 each slave has a replication of data from master. As changes can be made only by master system becomes more reliable. In case if one of the Master node fails over and loses data everything can be restored from the copy. [54]

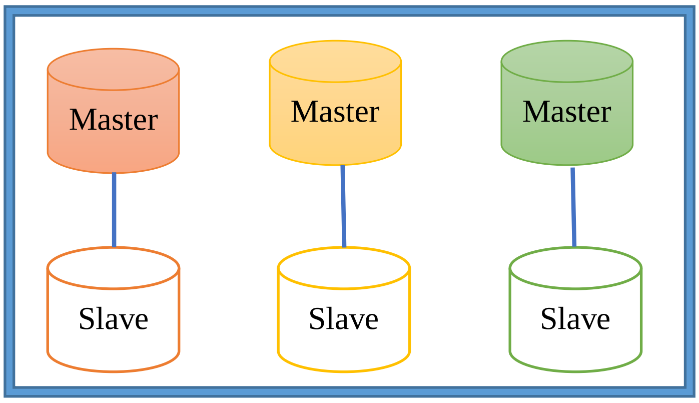


FIGURE 12 Master-Slave architecture

Sometimes it is not enough to have only one slave in such a situation the full data set can grow in a whole “slave tree” as on Figure 13. Here Slaves still have no writing privileges but they make a second layer backup which can bring even higher failover system. Additionally it makes higher performance to a system where reads occur more often than updates. [55] [56]

Apart from Master-Slave architecture, Master-Master is the relation between nodes where all the DBs in a cluster are granted with permissions to equally read and modify data. Here any node can act as a back-up to every other node. It bring to the system

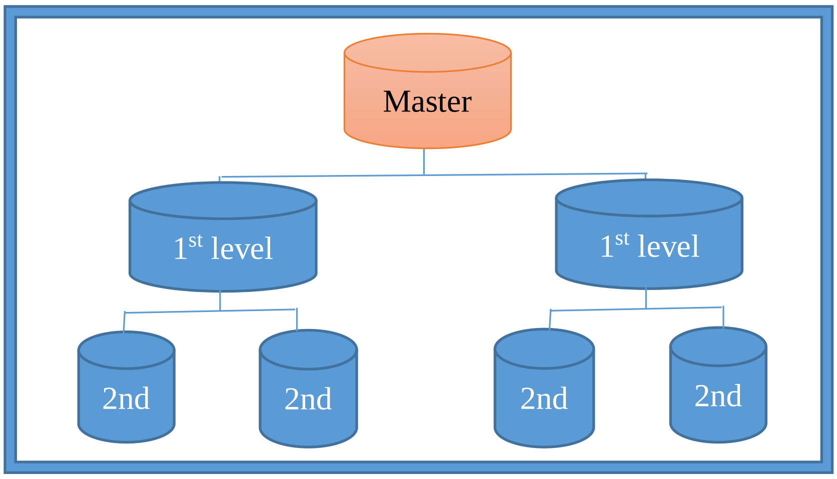


FIGURE 13 Master-slave tree

increased fail-over solution. On the other hand every node being a master means reads and writes are ought to be committed fast – high performance, expensive software is required. In the end it will guarantee full speed, easy planning and installation as no partitioning is needed and Load Balancing of the data would be done efficiently. [55] [56]

Comparing all the architectures there is no correct answer from which is better as every system has its advantages and drawbacks. SN is cheap to install but it takes time to design and maintain. If it is done correctly SN would be a good solution and add higher availability to the system. SD would require more investments but easier to set up and scale in the future. The choice for the system totally depends on personal preference, system size and demands. [55] [56]

Example of how both types of architecture can used together can be shown effectively in a case of unplanned downtime (connection is lost or server fails). Shared-nothing would work in a master-slave architecture system which means it relies on a single node failure. If it is a master which is down then a slave would take over and become a master. For SD after a failover node the next node/server would take place, therefore there would be no need for master-slave replica on the back nodes. Although, for the improved security it can be wise to back-up the main storage itself as a master-master replica. [55] [56]

4.2 Clusters in Database Solution

The previous chapter presented problems connected with low scale and performance connected with relational database management system. When a DB grows in size and more users get connection it becomes difficult to maintain and support. There are two ways to solve the problem: scale up or scale out.

When scaling up or vertical scaling, large numbers of CPUs and RAM are added. But capacity cannot be increased continuously. At first it makes system too massive, expensive and also every single machine has a limit for increasing capacity. A cloud provider can be a solution, although in this case a company needs to think if it is secure for personal data and records. Scaling up with relational technology. Shared everything architecture, bigger servers with more CPUs, more memory. As amount of users can vary from 10 to 1000 so the prediction and building environment is risky – “Too much and you’ve overspent; too little and users can have a bad experience or the application can outright fail.” [44]

Distributed system is a way how data can be scaled horizontally or in other words **sharding**. During sharding process all the nodes contain inside identical schemas for each data block. This technique shows the best practice in a place where data does not need to be provided in one set but can be delivered to the application or user separately. A smaller database is shown easier to handle, it is faster and can reduce costs compared to vertical scaling technique. [57]

Both types of scaling gives a database an ability to scale and support more users and data. Extending the useful scope of relational database – sharding, denormalizing, distributed caching solves the first problem of relational database. As all of those actions deal with the problem but in time create another. Relational database cannot support too many nodes on a cluster, moving between the nodes for a system means bringing the operational status down. For a relational database it also does not assure redundancy and availability of data. [54] [57]

4.3 Clusters Examples in Open Source Databases

MariaDB Galera Cluster is a synchronous multi-master fully free supported cluster solution for MariaDB. Read and write can be operated by any node same as client can connect to any node. MariaDB developers point that for better performance and node recovery minimum 3 nodes in a cluster. Operations between nodes are load balanced. Has a potential being setup on WAN scale to connect different data centers. [51]

MariaDB Replication cluster is an example for master-slave architecture. Master node writes about every event occurred into the binary log, then slaves read the data from each master so the data is replicated. In replication scenario nodes do not need to be in a constant connection with each other. If a slave node for some reason lost communication after it reconnects all the data would be mirrored from the master node. [58]

While RDB were searching and creating different solutions how to move into the cluster, making updates and changing structure, NoSQL are created from the very beginning to work on a distributed system. They are easy to scale out, build to tolerate failures. When working with NoSQL does not matter how many nodes system has or will have in the future - the application will always see system as a whole, single database. Due to its structure NoSQL can support auto-sharding and work as fast on hundred distributed servers. Auto-sharding means that the database spreads data across the servers automatically without application knowing it. As a result data is handled without application interruption and if a server goes down, it will be recovered from the other node. If more servers are added (and NoSQL can work with virtual servers) the data is spread and balanced into a larger cluster. [59]

MongoDB Replication is a solution which allows store synchronized data on multiple servers. It is an example of master-slave cluster architecture. Replication in MongoDB increases data availability and recover from hardware failure. As one of the servers can be disaster recovery node, reporting or a backup node. [60]

MongoDB Sharding shows shared-nothing architecture in practice. Data is distributed across shards by shard keys. Shard key is a specified field (index) which exists in every document in a collection. Shard key values are divided into chunks (range group) and then they are evenly spread across the shards. There are two ways how the shards are grouped together: **range based partitioning** and **hash based partitioning**. In the first type the numeric index shows to which chunk data belongs. For example for the range of serial numbers some specified numeric sets will form a chunk (Figure 14). [60]

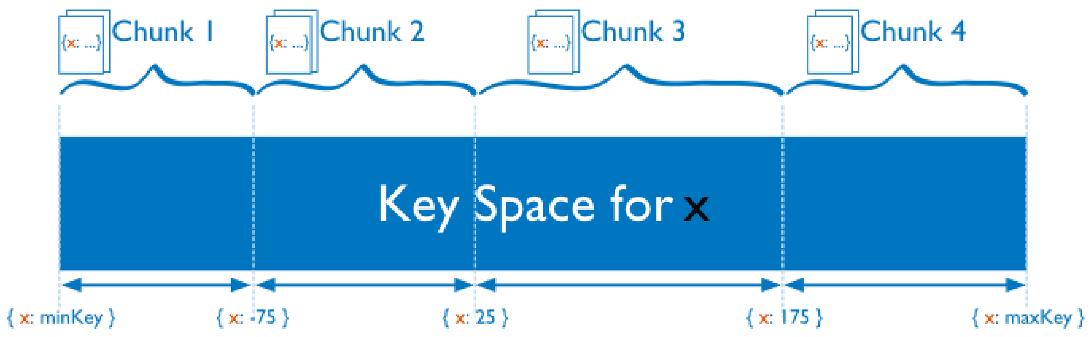


FIGURE 14 Range Based Sharding [60]

For hash based partitioning MongoDB computes a hash value for every document and then the chunks are creating using those values. In this type of partitioning unlike the first one, document with neighboring key index will most likely not be in the same chunk. Such logic ensures very random distribution across the servers.

In Chapter 7 shards usage reduces the load of a single server by distributing read and write operations between each node. [60]

5 ENVIRONMENT OVERVIEW AND INITIAL CONFIGURATION

This part describe hands on testing cases for database types and clusters covered in the theoretical part. On the testing scope in this part there are a relational MariaDB, NoSQL solution MongoDB and their clustered replications. The goal of the chapter is to build the databases and clusters, evaluate the installation process, weight each limitations, advantages and if possible to test performance on a sample dataset.

As it was show in a table 2 all the software used is open-sourced and easily accessible to download freely in the web. Every database chosen has full documentation, white papers and command manuals in order to set ups systems correctly and us it. In this mostly used official documentation provided by developer companies and tutorials from independent developers as a base for the configurations bases. There is already a number of software presented in the web with GUI for building and administrating database clusters. However, in this work every configuration is made from a scratch, without any third-party software in use.

Both databases and their clustered solutions are tested for performance on relatively sizable scale data. Stress-test for multi-user access, running queries and transactions. Big part of testing is to evaluate how databases would be able to react for rapid changes and workload. For MariaDB and MongoDB I would like to get more into the clustered solutions, evaluate implementation difficulties and their performance on manageable scale.

Syntax for practical part:

```
#command and output used in terminal OS
>command used in database and sql files
Test inside of config files
```
//any other comment

5.1 OSA Project and Testing Environment

In the project the system is based on RedHat environment. It provides an interface to deploy virtual machines and have remote access to them. In the system there are 3 virtual machines in the same LAN segment, no firewalls between the servers. The project plans to use CentOS for database purposes. For the thesis testing purposes all the installation have been done in a classroom lab on 3 PCs connected through LAN network with a switch. Virtualization is done with VMware Player. Every VM is created on separate PC, dual core, 2048MB RAM and 50GB disk, Bridged Interface Network interface card which uses the host interface. On VMs Linux CentOS is running. Therefore, if other OS is used commands/installations may vary for every other system/configuration. Such configurations are chosen in order to get the testing environment to the real project as close as possible.

5.2 MariaDB Environment

MariaDB newest release 10.0.10 (claimed to be stable by MariaDB Foundation) shown itself operating on singe machine in MariaDB server and MariaDB-client, and Galera-cluster 10.0 even being in Alpha state seems operational already. Replication and Galera are clustered solution for MariaDB database. Stress testing on MariaDB is done with mysqlslap. It is a built-in diagnostic tool for testing the database performance. Mysqlslap allows to perform an auto-generated sql queries or custom user defined queries. In the configuration for mysqlslap it is also possible to define simultaneous number of users, number of queries and precision of every test. For MariaDB testing data used is taken from MySQL AB, SAMPLE EMPLOYEE DATABASE. The file is given as SQL script with .dump files. The database was creating for testing purposes. It contains 300,000 employee records and 2.8 million salary entries. Which makes the database heavy enough for testing. The database structure is presented in APPENDIX Figure13.

Additional testing for MariaDB can be done with MySQL Testing Suite. It allows over 4000 different tests for database performance, security and optimization check. Additionally using Testing Suite Framework a user can create its own customized tests for the database. Testing suite is installed in this thesis together with database. The location of the environment is under /usr/share/mysql/mysql-test.

MongoDB is presented in this work with the latest release of 2.6 presented on April, 8 in 2014. “Key features include aggregation enhancements, text search integration, query engine improvements, a new write operation protocol, and security enhancements.” [Release Notes for MongoDB 2.6] For MongoDB system clustering is performed with Replica and Sharding solutions. Testing for MongoDB is planned to be done with mongoperf and mongo-perf tools.

Data is download from https://launchpad.net/test-db/employees-db-1/1.0.6/+download/employees_db-full-1.0.6.tar.bz2 and untar employees_db-full-1.0.6.tar.bz2 source to the needed location. In employees_db directory edit employees.sql file as follows:

Leave only “storage_engine=InnoDB” from the storage engines. For every .dump file write the full PATH, as example:

```
SELECT 'LOADING departments' as 'INFO';
source /home/student/employees_db/load_departments.dump ;
```

Create database in MariaDB:

```
>CREATE DATABASE employees;
```

In shell locate and import the data file:

```
#mysql –u root employees < employees.sql
```

5.3 MongoDB Environment

As MongoDB is a totally different architecture database the same data set cannot be used for all the cases. For this reason a new data is taken in json format from MongoDB sample data, named zips. The sample data contains City name, longitude and latitude, state and zipcode. Full data set is 29470 objects. Into mongo the object are imported with mongoimport.

```
#mongoimport –d thesisdb –c zips - -type json - -file /home/student/zips.json
```

For MongoDB there exist mongoperf built-in package. Although it is designed as MongoDB performance testing tool for now it cannot be used on mongod (database) or mongos (shards), only for system disk performance. In future developers are announcing to add support for testing the database itself.

There are other development testing tools in Mongo suit. For example, there are various different test available as part of smoke.py. While the idea and description of tests seems helpful in database performance evaluation, the suit does not fit into the required system. The main driver for every suite is Pymongo, which is run with specific python, both of which are not included with CentOS and require the installation. Although with the correct python installation location for the suits is not found for this MongoDB installations.

5.4 Cluster Nodes Configuration

Pre-configuration: For cluster configurations every node need to have communication between each other, without any security or firewall setting in the way. Therefore, hosts has to be configured for networking and in case of problem with connection between nodes security is shut down.

Firewall: for testing purposes it is better to turn off all the firewalls so it will not create interference.

```
#service iptables stop chkconfig iptables off
```

Security-Enhanced Linux: Linux kernel security module provides high process control security and it restricts database engines. As this work is not concentrated on security SELinus is disabled here.

```
#nano /etc/sysconfig/selinux
```

SELINUX=disabled

```
#setenforce 0
```

Hosts: in this work ip configurations are static and are same for every node configuration.

```
#nano /etc/hosts
200.100.1.30 node1 //always here the main/master node
200.100.1.20 node2
200.100.1.10 node3
```

6 CLUSTER BUIDING AND PERFORMANCE TESTS

For every cluster built in this practical work the working structure is the same. At first the installation of the machine cluster is documented through step by step with notes and explanations. Loading of data is skipped as it is already guided in part 5 PRACTICAL PART. Next, stress and performance testing is done in different environments (where it is possible) for MariaDB and MongoDB. System backup, failover, recovery is performed/explained in the end of each testing part.

6.1 Single Server MariaDB

MariaDB is a case of relational database. Its source is based on MySQL, which makes those two databases very close in structure and processing. SQL queries for both databases can be adopted to each other. Therefore if a user is familiar with MySQL then MariaDB will not bring difficulties.

Installation: For MariaDB single server it is very straight forward. Depending on the operating system repo script for yum installation can be found from https://downloads.mariadb.org/mariadb/repositories/#mirror=netinch. The commands to place it in right directory looks like this:

```
#nano /etc/yum.repos.d/MariaDB.repo
```

The command will create MariaDB.repo and open, where the repo can be placed inside:

```
# MariaDB 10.0 CentOS repository list - created 2014-04-09 12:42 UTC
# http://mariadb.org/mariadb/repositories/
[mariadb]
name = MariaDB
baseurl = http://yum.mariadb.org/10.0/centos6-amd64
gpgkey=https://yum.mariadb.org/RPM-GPG-KEY-MariaDB
gpgcheck=1
```

Next is installation with yum:

```
#sudo yum install MariaDB-server MariaDB-client MariaDB-test
```

NOTE! If there is a conflict during the installation with mysql then all the components of mysql server need to be removed.

```
#yum remove mysql-server mysql-libs mysql-devel mysql*
```

About my.cnf file. More ready-made cnf templates can be found in location /usr/share/mysql/. There are 5 different configurations. For all the installations in this work I have been using my-huge.cnf (full my-huge.cnf can be found in APPENDIX). For every installation I have replaced it into the /etc directory with name my.cnf.

After the package is installed and the configuration is ready to run the database server:

```
#sudo /etc/init.d/mysql start
```
or
```
#service mysql start
```

This work is not focused on running a secure server therefore the login into database can be done with root user without password or any additional settings for one server.

```
#mysql –u root
```

Running tests: As soon as queries are loaded testing can start. As it was pointed out earlier the tool used for testing is mysqlslap. Commands are entered from shell with identified parameters.

```
# /usr/bin/mysqlslap  --user=root --auto-generate-sql --
concurrency=100 --iterations=20 --number-of-queries=100
        Average number of seconds to run all queries: 0.075 seconds
        Minimum number of seconds to run all queries: 0.048 seconds
        Maximum number of seconds to run all queries: 0.174 seconds
        Number of clients running queries: 100
        Average number of queries per client: 1

# /usr/bin/mysqlslap  --user=root --auto-generate-sql --
concurrency=100 --iterations=20 --number-of-queries=5000
        Average number of seconds to run all queries: 1.965 seconds
        Minimum number of seconds to run all queries: 1.854 seconds
        Maximum number of seconds to run all queries: 2.115 seconds
        Number of clients running queries: 100
        Average number of queries per client: 50
```

For running test1 query (APPENDIX) on Employees database:

```
[root@localhost employees_db]# /usr/bin/mysqlslap  --user=root --
query=/home/student/test1.sql --concurrency=100 --iterations=20 --
number-of-queries=5000 --create-schema=employees
        Average number of seconds to run all queries: 2.078 seconds
        Minimum number of seconds to run all queries: 1.988 seconds
        Maximum number of seconds to run all queries: 2.554 seconds
        Number of clients running queries: 100
        Average number of queries per client: 50
```

```
[root@localhost employees_db]# /usr/bin/mysqlslap  --user=root --
query=/home/student/test1.sql --concurrency=1000 --iterations=5 --
number-of-queries=100000 --create-schema=employees
        Average number of seconds to run all queries: 2.120 seconds
        Minimum number of seconds to run all queries: 2.097 seconds
        Maximum number of seconds to run all queries: 2.194 seconds
        Number of clients running queries: 1000
        Average number of queries per client: 100
```

For the write operation I have chosen the queries for the longest table salaries by adding an additional column and populating it with data from employees table:

```
>ALTER TABLE salaries ADD name varchar(14);
Query OK, 0 rows affected (24.70 sec)
>UPDATE salaries JOIN employees ON (salaries.emp_no = employees.emp_no) SET sala-
ries.name = employees.first_name;
Query OK, 2844047 rows affected (1 min 11.07 sec)
Rows matched: 2844047  Changed: 2844047  Warnings: 0
```

For a single server failover and backup is highly irrelevant as data is contained on one machine. Therefore, in case of disaster, break-in or “drop table” command all the configurations and data are lost.

Result: If no complication appears the installation is straight forward. Errors can come from errors in depositories or previously installed packages. Configuration is minimum as only one server is required however with a growth of users and data the performance gets low. There is no backup or disaster recovery procedure with a single node.

6.2 Single Server MongoDB

Installation for MongoDB is very straight forward. Everything is followed by the tutorial from MongoDB. The procedure resembles the installation of MariaDB (or any "Red Hat" like installation):

Create repo `#nano`/etc/yum.repos.d/mongodb.repo

```
[mongodb]
name=MongoDB Repository
baseurl=http://downloads-distro.mongodb.org/repo/redhat/os/x86_64/
gpgcheck=0
enabled=1

#sudo yum install mongodb-org
```

Running tests: Performance and stress testing run into complication due to underdevelopment of mongoperf.

Result: The installation is easy and along multiple installations there have been no complications for single MongoDB server. For the backup and recovery the result is same like in MariaDB case, as the system is formed by a single node.

6.3 Replication Cluster MariaDB

In order to build the replication cluster in MariaDB configurations from 5.1 a) on 3 servers except are done for each node. The data is imported only into master node. The cluster topology is presented in Figure 13. The replication is done with 3 nodes:

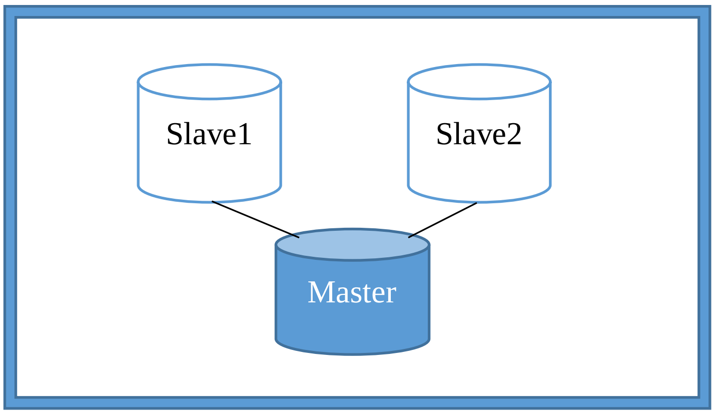


FIGURE 15 MariaDB Replication Topology

one is Master and two others are Slaves. This type of clustering is expected to get higher read performance, compared to single server, with balancing the queries between nodes when master node gets overloaded. In replication cluster of MariaDB slaves can be synchronous or asynchronous.

For Replication cluster in MariaDB main connection idea is that the nodes get the data from the binary log. In order for the cluster to function the master server is needed to be configured first. In the config file my.cnf these lines are changed:

`bind-address=200.100.1.30` //ip of the master node binder to the working machine

`server-id=1` //this configuration is already in place, make sure the id number for each node is unique

`Log-bin =/var/lib/mysql/mysql-bin.log` //the exact path for MariaDB binary log files

`binlog_do_db=employees` //name of the database to be replicated, can be multiple at the same time

MariaDB server is restarted from database shell grant privileges to the replication user can be assigned:

`>GRANT REPLICATION SLAVE ON *.* TO ‘slave_user’@’%’ IDENTIFIED BY ‘psw’;` //so slave nodes in replication set can access the database as replication users

`>FLUSH PRIVILEGES;` //used in order the privilege settings got in effect

. - revokes Global privileges so nodes can be added into the cluster

Next step is to distribute all the data to the slaves so they would become identical to master. On master node the table to be shared is locked and a fixed location for the master status is recorded, for the later slave replication.

```
>FLUSH TABLES WITH DEAD LOCK;
>SHOW MASTER STATUS;
```

```
+----------------------+------------+-------------------+-----------------------+
| File                 | Position   | Binlog_Do_DB | Binlog_Ignore_DB |
+----------------------+------------+-------------------+-----------------------+
| mysql-bin.000005    | 1863       |    employees      |                       |
+----------------------+-------------+------------------+-----------------------+
```

Next step is to get a snapshot of the master database. This command has to be done in new terminal window so the position will stay locked:

```
#mysqldump –u root –opt employees > employees.sql
```

Dump file can be found from the directory terminal. After the file is ready on master database tabled can be released:

```
>UNLOCK TABLES;
```

For now master configuration is ready and slaves can be configured. For each slave on MariaDB servers database is created and the dump file loaded:

```
>CREATE DATABASE employees;
# mysql -u root employees < PATH/employees.sql
```

Once the data is imported my.cnf can be changed. For every slave a unique id is needed and log location pointed:

```
Server-id=3
Relay-log=/var/lib/mysql/mysql-replay-bin.log
Log_bin=/var/lib/mysql/mysql-bin.log
Binlog_do_db=employees
```

To apply the configurations slave servers are restarted. Master-server information in MariaDB shell for slaves:

```
>CHANGE MASTER TO
  MASTER_HOST='200.100.1.30',
  MASTER_USER='slave_user', //replication user name
  MASTER_PASSWORD='psw'
  MASTER_PORT=3306, //default port is 3306, change if another port is in use
  MASTER_LOG_FILE='mysql-bin.000005',
  MASTER_LOG_POS=1863;
```

With this command slave server knows the location/address of the master node, gets the correct login credentials and tells the server where to start the replication from binary log.

Now each slave can be started:

```
>START SLAVE;
```

Slave status can be seen with command:

```
>SHOW SLAVE STATUS\G;
```

```
Slave_IO_State: Waiting for master to send event //the connection is set but binaries currently
are not transmitted
Master_Host: 200.100.1.30
Master_User: slave_user
Master_Port: 3306
Connect_Retry: 60
Master_Log_File: mysql-bin.000006
Read_Master_Log_Pos: 326
Relay_Log_File: mysql-relay-bin.000003
Relay_Log_Pos: 613
Relay_Master_Log_File: mysql-bin.000006
Slave_IO_Running: Yes
Slave_SQL_Running: Yes
…
```

On Master node slaves stats can be checked with command:

```
>SHOW SLAVE HOSTS;
+-----------+------+--------+--------------+
| Server_id | Host | Port | Master_id      |
+-----------+------+---------+-------------+
|          3 |       | 3306 |              1 |
|          8 |       | 3306 |              1 |
+-----------+------+-------+---------------+
```

Running tests:

Tests for write operations are performed in the same order as for singe server.

```
# /usr/bin/mysqlslap  --user=root --auto-generate-sql --
concurrency=100 --iterations=20 --number-of-queries=100
        Average number of seconds to run all queries: 0.054 seconds
        Minimum number of seconds to run all queries: 0.041 seconds
        Maximum number of seconds to run all queries: 0.083 seconds
        Number of clients running queries: 100
        Average number of queries per client: 1

# /usr/bin/mysqlslap  --user=root --auto-generate-sql --
concurrency=100 --iterations=20 --number-of-queries=5000
        Average number of seconds to run all queries: 1.955 seconds
        Minimum number of seconds to run all queries: 1.831 seconds
        Maximum number of seconds to run all queries: 2.294 seconds
        Number of clients running queries: 100
        Average number of queries per client: 50

# /usr/bin/mysqlslap  --user=root -- query=/home/student/test1.sql --
concurrency=100 --iterations=20 --number-of-queries=5000 --create-
schema=employees
        Average number of seconds to run all queries: 2.007 seconds
        Minimum number of seconds to run all queries: 1.998 seconds
        Maximum number of seconds to run all queries: 2.653 seconds
        Number of clients running queries: 100
        Average number of queries per client: 50

# /usr/bin/mysqlslap  --user=root --query=/home/student/test1.sql --
concurrency=1000 --iterations=5 --number-of-queries=100000 --create-
schema=employees
        Average number of seconds to run all queries: 2.025 seconds
        Minimum number of seconds to run all queries: 2.012 seconds
        Maximum number of seconds to run all queries: 2.047 seconds
        Number of clients running queries: 1000
        Average number of queries per client: 100
```

Write operation has shown exactly the same stats as one server case, as only Master can perform write operations, so the queries cannot be balanced. What is peculiar here, that after the change is done, the command is committed on Master it automatically passes the binary log to the slaves and the same changes can be seen from the slave nodes.

Failover testing: For every additional node added into the cluster the process goes the same ways in building the cluster. If a slave node goes down, in order to recover it the process goes the same as adding new node into the cluster. It the master node goes down it can be recovered from one of the slave nodes in a condition that the node contains the newest version of data. After the databases are loaded into the master node the whole replication process starts from the beginning.

Even though a replication cluster is viewed by many users as a backup solution only replication is not enough to provide a behavior a stable backup node. If a regular slave node in a system is considered to be as a backup it cannot stay connected to the cluster the whole time. As in that case any accident drop table or database query can damage the data on every slave depending on one master node. Therefore, the replication scenario is to create synchronizing schedule and keep the backup node offline in regular usage.

Result: Each machine installation is easy, linking them together is more challenge. Some difficulties can come from networking issues, firewalls or security systems. Additionally is it very important to make the configurations in my.conf correctly, for example bind address and replication name. Database migrations are manual therefore set up is needed for each database. In order to keep the cluster operational changes and logs have to check on regular bases therefore routine maintenance and support are required. Later when the databases become too big it might take time for synchronization on startup and when new nodes are added. Cluster does not affect write operations but it performs well in term of Replication set and read operations. Recovery option is tricky in case of the master node failover and, additionally, only replication is not enough for fully reliable system.

6.4 Galera Cluster MariaDB

As Galera cluster is Master-Master performance increase is expected. Write and read operations can be done form any node. The topology of the installation is three equal master nodes connected together.

Installation: According to official MariaDB page for Galera cluster installation is the same as for single MariaDB-server as the Galera package is included into the repository. The yum command only is needed to be changed:

```
#sudo yum install MariaDB-Galera-server galera MariaDB-client Mari-
aDB-test setroubleshoot
```

However, during this work the package was not presented in the repository and gave the notification **‘No package available MariaDB-Galera’**. In this case other packages are to be downloaded still such as MariaDB-client, galera and others.

In this scenario the download and installation of MariaDB-Galera should be done manually with rpm download. Source for the package - http://mirror.netinch.com/pub/mariadb/mariadb-10.0.8/yum/centos6-amd64/rpms/. Name for the package “*MariaDB-Galera-10.0.7-centos6-x86_64-server.rpm*”. The file is downloaded it and installed with terminal. To install rpm:

```
#rpm –Uvh MariaDB-Galera-10.0.7-centos6-x86_64-server.rpm
```

Then the operation is repeated for every machine is added into the cluster and start the server. If /etc/init.d/mysql ‘*)’ problem comment the line 429 in the file script. If a SELinux prompt an error #setenforce 0 command fixes it. If SELinux (appears as ERROR! The server quit without updating the file..) is needed to be on restart CentOS then run MariaDB and setroubleshooot (was installed with yum package) will report 3 errors. In every error report →Details there are commands which needed to run to allow MariaDB access.

For the first master configurations here we need to set user and the password. Any additional user can be created but in this work all the configurations are done with root user:

```
>GRANT ALL ON *.* TO  ‘root’ @’ localhost’  IDENTIFIED BY  ‘psw’  WITH GRANT OPTION;
>GRANT ALL ON *.* TO  ‘root’ @’ %’  IDENTIFIED BY  ‘psw’  WITH GRANT OPTION;

>FLUSH PRIVILEGES;
```

Next, firewall network ports have to be opened. SQL server firewall adjustments:

```
#system-config-firewall-tui
```

Ports 3306, 4567 and 4444 must be enabled with tcp for MariaDB cluster. Theoretically, this option should be enough but seems that Galera uses another ports as until all firewall settings are disabled. It is not the smartest solution but this subject can be used by further research. Therefore all the firewalls are needed to be turned off.

Galera Cluster ports:

3306 – Mysql client connections

4567 Galera Cluster replication traffic

4568 – Incremental State Transfer

4444 – all SSTs, Full State Transfer, used wsrep

Next configurations need to be done in my.cnf file (path /etc/my.cnf). Configurations for node1, main (configurations different from my-huge.cnf):

```
[mysqld]
bind-address=0.0.0.0
socket=/var/lib/mysql/mysql.sock
query_cache_size=0
binlog_format=ROW
default_storage_engine=innodb
innodb_autoinc_lock_mode=2
datadir=/var/lib/mysql
wsrep_provider=/usr/lib64/galera/libgalera_smm.so //default galera
files
wsrep_provider_options="gcache.size=32G; gcache.page_siz=1G"
wsrep_cluster_address=gcomm://200.100.1.30 //cluster location address
wsrep_cluster_name='magatest' //name of the cluster
wsrep_node_address='200.100.1.30'
wsrep_node_name='node1'
wsrep_sst_method=rsync
wsrep_sst_auth=root:psw
```

For the second and other nodes the configurations are the same except for the node address and node name should specified for that node. **Note!** `wsrep_cluster_address` stays the same as that is the address for the cluster itself, so the nodes know where to connect. Cluster name has to be same on every node, or the configuration is failed.

After all the configurations are done the cluster can be started. Before that making sure that mysql server is not running:

```
#service mysql stop
```

On the first node command is run in order to start creating a cluster:

```
#mysqld --wsrep_new_cluster --user=mysql
```

This command will tell the node that it is the main one, create a new cluster and uses the logging data from mysql server. After other outputs and configuration process if all the configurations are done right the first node should show this:

```
mysqld: ready for connections.
Version: '10.0.7-MariaDB-wsrep' socket: '/var/lib/mysql/mysql.sock'
port: 3306 MariaDB Server, wsrep_25.9.r3928
```

On the next and every other node in a cluster connection is also opened, cluster address is pointed with user credentials:

```
#mysqld --wsrep_cluster_address=gcomm://200.100.1.30 --user=mysql
```

After the command execution the connection between nodes is established and they are ready to communicate. In order to check the connection table modifications can be made and then they will be seen on the other nodes.

The terminal windows should stay open and for the access to the database new terminal window should be opened. Galera cluster status can be viewed from any node. As soon as something changes in the cluster the updated configurations are seen.

```
WSREP: Quorum results:
version    = 3,
component  = PRIMARY, //master-master replication every node is primary
conf_id    = 13,
members    = 3/3 (joined/total),
act_id     = 238264,
last_appl. = 0,
protocols  = 0/5/2 (gcs/repl/appl),
group UUID = 3511cbf1-c3c7-11e3-ab44-e757256286d8
```

For my.cnf file a lot more options can be defined for security, better data transfer and more complex networking.

Loading data: Although in the cluster there are 3 nodes, the database is loaded only on one node and while it is loading it is also parallel distributed to other nodes. For the future data grow data can loaded from any node in the system and other members synchronize it.

Running tests:

```
# /usr/bin/mysqlslap  --user=root -p --auto-generate-sql --
concurrency=100 --iterations=20 --number-of-queries=100
        Average number of seconds to run all queries: 2.102 seconds
        Minimum number of seconds to run all queries: 2.000 seconds
        Maximum number of seconds to run all queries: 2.390 seconds
        Number of clients running queries: 100
        Average number of queries per client: 1

# /usr/bin/mysqlslap  --user=root -p --auto-generate-sql --
concurrency=100 --iterations=20 --number-of-queries=5000
        Average number of seconds to run all queries: 13.593 seconds
        Minimum number of seconds to run all queries: 13.086 seconds
        Maximum number of seconds to run all queries: 14.103 seconds
        Number of clients running queries: 100
        Average number of queries per client: 50
```

For the next tests the execution time took too long and the query had to be stopped. Such performance can be explained from the nature of the database. As SQL database originally was not created for distributed data even new suite in Galera form is not enough to provide full distributed performance. The system is built presents itself as three equal communicating nodes – there is no master as in previous replication. Which means due to the structure all the nodes are overloaded and trying to split the data across for better performance but none of them can split the data into correct sets, so they communicate blindly. In order to fix the problem an external load balancer is needed for the system. Galera Cluster used to have a separate load balancer - Galera Load Balancer (at the moment it is missing for download neither in the depository).

Failover testing: Apart from mysqlslap testing for the cluster it is possible to test its architecture reliability. All the nodes have been disconnected and then put together, it takes less than couple of seconds to bring them back into the cluster. The same situation if one node gets disconnected form the cluster (except for the main node) after establishing the connection back it will be synchronized with the cluster changing state from JOINER to JOINED and then SYNCED. |**Note!** If data on the cluster is massive it can take a while| If the main primary node gets disconnected the cluster still stays operational. However if and when the cluster is restarted the main node is needed to be configured again as if the cluster is just being built. In my.cnf it is possible to set parameters so every server will be connecting automatically into the cluster as soon as it is turned on.

Result: As Galera cluster continues update and development the latest versions can be unstable and have problems with repositories and newest versions. During installation complications can appear form the configurations – wsrep options, user access, addressing and firewall. Only cluster installation and operations are not enough to bring the system to function with all the benefits of a cluster. Only with additional load balancer write and read operations can be balanced across the nodes for better performance. If two nodes trying to commit changes for the same database the cluster will prevent the data conflict and set a deadlock for the table. However, if query is being executed from two or more nodes simultaneously and perform with different databases, performance increases and deadlock will not occur. Nodes are managed automatically, so if server drops it is deleted from members list. When new databases or tables are added to one server the data is automatically copied to other nodes in a cluster.

Galera architecture holds a very promising WAN scale system potential. Plus, when the system grows every master node could hold its own slave replication with its every benefit such as local read performance and backup.

6.5 Replication Cluster MongoDB

For building a cluster in MongoDB there is no need to download any additional package as everything is inside of the basic installation package. For NoSQL developers it is essential to assume that the database is going to be used on multiple machines. Minimum number of nodes is 3 - 1 is primary and 2 others are secondaries. Topology follows MariaDB replication Figure 13. Primary node is the only node to accept write operations, after the changes are recorded into the oplog. Secondary nodes get replicates of the log and apply the changes in their own datasets, additionally only accept external read operations.

Installation: Before configuring MongoDB or starting the server configurations are modified in Operating System for SElinux and firewall. |**Note!** MongoDB supports hosts names (Local or DNS)|. In order for every node to work MongoDB server needs to be installed on every machine. For replication MongoDB uses mongod package which stores files in different location from mongo. The storage has to be created manually on each node:

```
#mkdir /data/db
#chmod 777 /data/db
```

Next step is to change configuration file for every node. The file can be found in /etc/mongod.conf. Initial configuration can be found in APPENDIX. In configuration file sets for various replica configuration:

`port=27017` //can be manually specified port
`bind_ip=127.0.0.1` // comment the line for the interface listens to other ports
`replSet=rs0` //sets the replication name – has to be identical for every node

Same configuration is done for each node.

Try `mongod --replSet "rs0"` // possibly, does not work

Next command starts mongod with forced configuration and points out to the replication name:

```
[root@node1 /]# mongod --config /etc/mongod.conf
about to fork child process, waiting until server is ready for con-
nections.
forked process: 25405
child process started successfully, parent exiting
```

With this the command mongo service is started and now it is listening for incoming connections.

Next step is to connect to mongo

```
[root@node1 /]# mongo
MongoDB shell version: 2.6.0
connecting to: test
Welcome to the MongoDB shell.
For interactive help, type "help".
For more comprehensive documentation, see
        http://docs.mongodb.org/
Questions? Try the support group
        http://groups.google.com/group/mongodb-user
```

As node1 is a primary node it is the first to start replica initiation. With rs.initiate() command replica configurations built locally and the node becomes primary.

```
> rs.initiate()
{
        "info2" : "no configuration explicitly specified -- making
one",
        "me" : "node1:27017",
        "info" : "Config now saved locally.  Should come online in
about a minute.",
        "ok" : 1
}
```

With rs.config() configuration status for the cluster can be viewed. For now it holds only one host.

```
> rs.config()
{
        "_id" : "rs0",
        "version" : 1,
        "members" : [
                {
                        "_id" : 0,
                        "host" : "node1:27017"
                }
        ]
}
```

In order to add other members into the cluster mongod should be started with new configurations holding replica name:

```
[root@node3 /]# mongod --config /etc/mongod.conf
about to fork child process, waiting until server is ready for con-
nections.
forked process: 2695
child process started successfully, parent exiting
```

When the hosts are ready for connection on the main node from Mongo shell each cluster member can be added using host names:

```
rs0:PRIMARY> rs.add("node2")
{ "ok" : 1 }


rs0:PRIMARY> rs.add("node3")
{ "ok" : 1 }
```

For monitoring cluster activity rs.status() command can be issued for nodes id, node and cluster states, uptime and cluster updates timing.

```
rs0:PRIMARY> rs.status()
{
        "set" : "rs0",
        "date" : ISODate("2014-04-23T14:31:32Z"),
        "myState" : 1,
        "members" : [
                {
                        "_id" : 0,
                        "name" : "node1:27017",
                        "health" : 1,
                        "state" : 1,
                        "stateStr" : "PRIMARY",
                        "uptime" : 179,
                        "optime" : Timestamp(1398263485, 1),
                       "optimeDate" : ISODate("2014-04-23T14:31:25Z"),
                        "electionTime" : Timestamp(1398263344, 2),
                        "electionDate" : ISODate("2014-04-
23T14:29:04Z"),
                        "self" : true
                },
                {
                        "_id" : 1,
                        "name" : "node2:27017",
                        "health" : 1,
                        "state" : 5,
                        "stateStr" : "STARTUP2",
                        "uptime" : 14,
                        "optime" : Timestamp(0, 0),
                       "optimeDate" : ISODate("1970-01-01T00:00:00Z"),
                        "lastHeartbeat" : ISODate("2014-04-
23T14:31:32Z"),
                        "lastHeartbeatRecv" : ISODate("2014-04-
23T14:31:31Z"),
                        "pingMs" : 0,
                        "lastHeartbeatMessage" : "initial sync need a
member to be primary or secondary to do our initial sync"
                },
                {
                        "_id" : 2,
                        "name" : "node3:27017",
                        "health" : 1,
                        "state" : 5,
                        "stateStr" : "STARTUP2",
                        "uptime" : 7,
                        "optime" : Timestamp(0, 0),
                       "optimeDate" : ISODate("1970-01-01T00:00:00Z"),
                        "lastHeartbeat" : ISODate("2014-04-
23T14:31:31Z"),
                        "lastHeartbeatRecv" : ISODate("2014-04-
23T14:31:31Z"),
                        "pingMs" : 0,
                        "lastHeartbeatMessage" : "initial sync need a
member to be primary or secondary to do our initial sync"
                }
        ],
        "ok" : 1
}
```

After a while later the heartbeat message will be sent over to the STARTUP nodes and they will change the configurations to SECONDARY. When refreshing the rs.config()

```
rs0:PRIMARY> rs.status()
{
        "set" : "rs0",
        "date" : ISODate("2014-04-23T14:42:44Z"),
        "myState" : 1,
        "members" : [
                {
                       "_id" : 0,
                       "name" : "node1:27017",
                       "health" : 1,
                       "state" : 1,
                       "stateStr" : "PRIMARY",
                       "uptime" : 851,
                       "optime" : Timestamp(1398263485, 1),
                      "optimeDate" : ISODate("2014-04-23T14:31:25Z"),
                       "electionTime" : Timestamp(1398263344, 2),
                       "electionDate" : ISODate("2014-04-
23T14:29:04Z"),
                       "self" : true
                },
                {
                       "_id" : 1,
                       "name" : "node2:27017",
                       "health" : 1,
                       "state" : 2,
                       "stateStr" : "SECONDARY",
                       "uptime" : 686,
                       "optime" : Timestamp(1398263485, 1),
                      "optimeDate" : ISODate("2014-04-23T14:31:25Z"),
                       "lastHeartbeat" : ISODate("2014-04-
23T14:42:43Z"),
                       "lastHeartbeatRecv" : ISODate("2014-04-
23T14:42:42Z"),
                       "pingMs" : 0,
                       "syncingTo" : "node1:27017"
                },
                {
                       "_id" : 2,
                       "name" : "node3:27017",
                       "health" : 1,
                       "state" : 2,
                       "stateStr" : "SECONDARY",
                       "uptime" : 679,
                       "optime" : Timestamp(1398263485, 1),
                      "optimeDate" : ISODate("2014-04-23T14:31:25Z"),
                       "lastHeartbeat" : ISODate("2014-04-
23T14:42:44Z"),
                       "lastHeartbeatRecv" : ISODate("2014-04-
23T14:42:42Z"),
                       "pingMs" : 0,
                       "syncingTo" : "node1:27017"
                }
        ],
        "ok" : 1
}
```

Failover: In Mongo Replica Set Secondary nodes are not only the one which replicates data but also in case of primary node failure they elect new primary node, according to data/update state, location or priority. Arbiter node however is another type of node which does not hold the data but play a role in case of new primary node election but cannot become primary. BackUp node holds the data and can be a part of the election process but will never become primary. As in the architecture map the BackUp node is in a hidden state. There can be up to 12 members in a Replica Set.

Result: Installation of the cluster is very logically built, failover and backup solutions provide high reliability, secondary nodes can increase read capacity as they have an ability to send read and write operations to other nodes. Additional nodes can be configured easily, failover process is automatic. Replica Set has a possibility to be geographically distributed for providing fault tolerance if one of the datacenters is not available. For such architecture a secondary node with priority 0 which cannot become primary is installed in different data center form original data. Thus the data can be fully recovered from even if the whole data center falls down.

6.6 Sharding Cluster MongoDB

Sharding allows to store data across multiple nodes. For correct and successful configuration each Sharding cluster must have 3 different types of nodes:

Shard nodes store the data. For the production environment best performance each shard is a separate replica set.

Query Routers run mongos instance. It is a link between client and appropriate shards. In a sharded cluster there can be more than one query router. Mongos instance does not need much resources therefore can be run together with mongod instance on the same machine.

Config Servers contain cluster’s metadata. It is used to target operations to the correct shard. In a production there has to be three config servers.

Sharding Cluster topology is presented in Figure 14. The figure is taken form MongoDB Sharding tutorial.

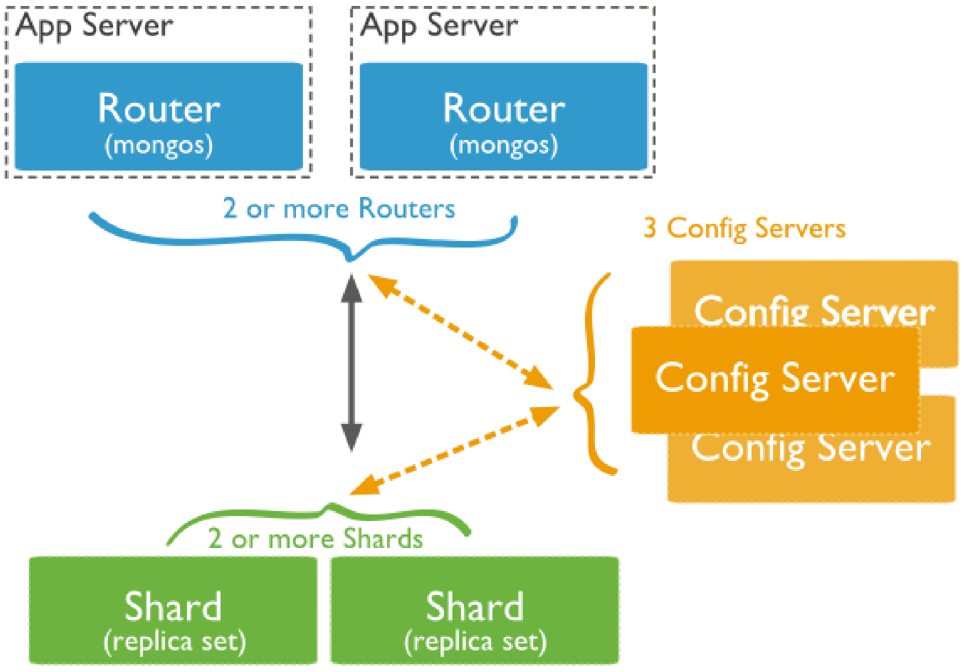


Figure 16 Sharding MongoDB

Installation:

Nodes has to be able to communicate with each other. Therefore firewall, selinux and hosts configurations are needed. MongoDB server is installed on each machine. After the installation file on each machine in mongod.conf comment out bind_ip so the servers listen to all interfaces. First one to configure is Config Server (node 1). There should should be 3 Servers per cluster but in this work for testing purposes only one is used (in a smaller development for mongo servers multiple instances can be run from the same machine). The directory with permissions is created for Server instance.

```
#mkdir /data/ins1
#chmod 777 /data/ins1
```

This command will start server instance for the selected location, on a required port.

```
#mongod - -configsvr - -dbpath /data/ins1 - -port 27023
```

After the server started running the configurations are ready and the Server is waiting for connections. Next to add to the cluster are Mongos instances (Router) node2. In order to run correctly Router node has to get connection and path for the Config Server. There can be more than one router in a cluster – the installation procedure is the same.

`#Mongos - -configdb node1` (if there are 3 config Servers then for each starting mongos they have to be listed in the same order).

At last adding Shards From node3

```
[root@node3 ~]# mongo --host node2 --port 27017
MongoDB shell version: 2.6.0
connecting to: node2:27017/test
```

At this moment node3 is the master shard – place where the configuration is starting. When issuing status command we can see that there is no shards yet added.

```
mongos> sh.status()
--- Sharding Status ---
  sharding version: {
          "_id" : 1,
          "version" : 4,
          "minCompatibleVersion" : 4,
          "currentVersion" : 5,
          "clusterId" : ObjectId("5358d65ac0cfed7b558f3396")
}
  shards:
  databases:
          {  "_id" : "admin",  "partitioned" : false,  "primary"
: "config" }
```

In order to add shards on each shard the router needs to be identify to open the connection:

```
# mongo --host node2 --port 27017
```

After that on the main shard new members can be added. In this work node2 is added as a shard to test how it can be running along with router. For that configuration is modified in mongod.conf file for the port line. As in mongos 27017 port is taken, I changed for mongod port 27018.

Shards are added from the first shard node (in this case node3), it is important to identify the ports if they are different:

```
mongos> sh.addShard("node2:27018")
{ "shardAdded" : "shard0000", "ok" : 1 }
```

For this testing I have additionally added VM number 4 with a host name noden for more sharded connections.

```
mongos> sh.addShard("noden")
{ "shardAdded" : "shard0001", "ok" : 1 }
```

After shards are added successfully status shows us different update.

```
mongos> sh.status()
--- Sharding Status ---
  sharding version: {
          "_id" : 1,
          "version" : 4,
          "minCompatibleVersion" : 4,
          "currentVersion" : 5,
          "clusterId" : ObjectId("5358d65ac0cfed7b558f3396")
}
  shards:
          {  "_id" : "shard0000",  "host" : "node2:27018" }
          {  "_id" : "shard0001",  "host" : "noden:27017" }
  databases:
          {  "_id" : "admin",  "partitioned" : false,  "primary"
: "config" }
```

After the shard cluster is build and totally functional data is need to be loaded and allowed to be sharded in the cluster. This will allow to distribute the collection across the shards, with specified key parameter. Shard key will be used to from the chunks.

{ shardCollection: "<database>.<collection>", key: <shardkey> }

Chunk control and balancing data and chunk migrations run in the background of sharded cluster. When a chunk size exeedes the maximum it is split and if needed migrated. [http://docs.mongodb.org/manual/core/sharding-introduction/]

Result: Installation of sharded cluster is complicated because of the nodes roles and cluster structure. As a single server load is distributed across the shards it allows high throughput applications better performance. If the system is geographically dependent each shard can be specified for the location. Although the cluster does not allow back-up by itself but if each shard uses a Replica Set instead of single server the system becomes reliable and high available.

7 CONCLUSIONS

This chapter concludes all the result which was found out during the thesis work. The aim of the study was to get familiar with database cluster solutions, find out different options and give recommendations for OSA project.

Clustered databases can help to overcome a number of limitations of a single server. As it was explained in chapters 2, 3 and 4 clustered solution together with NoSQL movement are a logical step into the database development. Together they can solve the problems of performance, reliability and availability compared to the relational model. During this work four different clusters have been built and from the testing it can be viewed that clustered solutions make difference in data organization. Each of them carries on different architecture, requirements and purposes. For a small company a single replication is enough so it achieves backup function and load balancing for read operations. For another company it would be efficient not only to balance the read but also write operations. And the third company wants to achieve concurrency on a WAN level.

Not every goal of the research was achieved. As the database behavior depends and varies a lot on a number of configurations there was no possibility to carry out full statistical dependencies and analysis. As for OSA project there is no concrete solution can be given of which cluster is the best to build as their project is a unique case. From the archive point of view the best system would be if a user can upload any type of data inside of the archive without limitations, after the upload the processed data would be structured and contained with a help of relational model based storage. Along the research it was found out that such a solution is being developed and shown an operational potential with CassandraSE.

This thesis presents clustered database example implementations and clustered logic references for the future and current state of the OSA project. As the project not only focuses on performance but also reliability and backup solutions replication clusters are able to fulfill the project requirements. In the future as the data store and users pool grows it could be helpful to implement Galera and Sharded clustes. In addition, replication or Galera can bring advantages into the system with coexistence with CassandraSE.

CassandraSE and Neo4j were originally planned to be tested in this work but I have found them difficult to implement for current environment stage. CassandraSE, as it was mentioned in the newest release MariaDB 10.0.10, gives the user a possibility to connect relational and non-relational (column) systems. As the developers assign it is a window from SQL into NoSQL. While conducting the research I have found some successful stories about Cassandra and MariaDB implementation but mostly only for testing. From what I have read most people are using Ubuntu as OS and also conducting the installations by using third party tools such as Vargant. Example of such installation can be found in Julien Duponchelle blog [61]

Neo4j is another type of the database with graph structure for which no testing project data have been found. As a NoSQL database Neo4j operates with its own language - Cypher there is no easy way to implement data from other sources. I still think that in future Neo4j can be helpful for the development, for example system architecture map or client → files fast dependencies.

While I have been following tutorials from both vendors (MongoBD and MariaDB) I could not skip the documentation evaluation. For both systems there exist official and developers tutorials and documentations. Sadly I have grown an opinion that they alone are not enough. Along the installations I have been reading lots of different courses as was not able to get a full picture form the official documentation. Although in the end of the research after reading the files again I found them quite clear and well-written. MariaDB misses some initial system requirements, clear view about the cluster, easier to read configuration options and also some working examples of an operational system. In MongoDB everything seems being in its place. They bring the information in a very compact way and easy to follow. Even though there could be more pointers into config file organization and mongodb contributers.

Open Source is always more challenging to handle than other types of software. In this case it was my first experience with Open Source databases and clusters. Building the system was not as time consuming as trying and find all the answers for every coming error during the installation process. After thousand times of trying when all the possible complication are found then new ones can appear. Sometimes the problem appear as a fact that the software is still in a development state. When I had just started my work there were some difficulties in initial installation stage of MariaDB but upon the thesis finalization seems that developers have fixed some issues.

Thesis reliability and limitations

The main work limitation is that the thesis practical part have not been tested in actual project environment. Clusters building and operations have been carried on tested data which was not able to show the working scope as the files stored in archive differ in size and types therefore different from testing data. The full data set would be too complicated to virtualize for testing. There are number of points which were not considered before the thesis process such as OS issues, additional testing and virtualizing programs. Software for testing brought challenges as good freeware testing programs is difficult to find. As it comes to no surprise any system optimization is a consuming and expensive work. Unfortunately, testing with inside built systems is not always enough. Therefore there can be several ways to overcome the dilemma – to test the performance in a real time working environment or to create specified testing tools which would support all the tests required for the system.

Additionally for system overview CentOS might not be the best decision for running a database cluster. It is considered very reliable and full Linux distribution it has limitations in installations and configuration for open source databases.

Future research

The topic is Database Cluster and NoSQL connection to SQL is quite new. In this work I have completed raw configurations of multiple clustered servers. The systems are crude as there is not real data with it, they are not connected to the actual system, and there is no security configured inside or proper user names. The systems which were built are not the last step in how a databases or a database cluster can be optimized. Thus for the future research it would be really necessary and relevant to implement user access, security customization, implement the system into the real environment and see how it performs with archive data. For even further research development of SQL and NoSQL implementations needs to be followed.

BIBLIOGRAPHY

[1] Sharma, Neeraj (ed). 2010. Database Fundamentals. IBM Canada.

[2] Sumanti, S., Esakkirajan, S. 2007 Fundamentals of Relational Database Management Systems. Studies in Computational Intelligence, Volume 47. Heidelberg: Springer.

[3] Educational Portal | Data Management. 2013. http://education-portal.com/academy/topic/data-management.html Referred: 10.11.2013.

[4] Java Programming Tips and Hints. http://www.erpgreat.com/java/difference-between-database-and-management-system.htm Referred: 26.03.2014.

[5] Tech Terms Open db Connectivity. 2013. http://www.techterms.com/definition/odbc. Referred: 10.12.2013.

[6] Computer History Museum | History of Databases. 2013. http://www.computerhistory.org/revolution/memory-storage/8/265/2207 Referred: 20.9.2013.

[7] Codd, E. F. 1970. A Relational Model of Data for Large Shared Data Banks. Communications of the ACM: 5.06.1970, Volume 13.

[8] Stanford University Lecture Notes. Chapter 8. http://infolab.stanford.edu/~ullman/focs/ch08.pdf.

[9] Hernanndes, Michael J. 2013. Database Design For Mere Mortals. Michigan: Edwars Brothers Malloy.

[10] Database Management Wikia. Relational Database: Keys. http://databasemanagement.wikia.com/wiki/Database:_Keys. Referred: 10.11.2013.

[11] The IT Education Site. Candidate Key. 2013. http://www.techopedia.com/definition/21/candidate-key. Referred: 10.11.2013.

[12] What are Relational Database? http://computer.howstuffworks.com/question599.htm Referred: 9.12.2013.

[13] IBM WebSphereStudio Application Developer Version 5.1.2 Product Overview, Defining Schema. Second Edition. 2002. IBM Canada.

[14] Worboys, M. F. 2001. Chapter 26, Relational databases and beyond. In a book Longley P.A. (ed), Geographical Information Systems. England: John Wiley & Sons, 373 – 384.

[15] Rischert, Alice. 2002. Oracle® SQL Interactive Workbook, Second Edition. Prentice Hall.

[16] Database Management Wikia. Relational Database: Metadata http://databasemanagement.wikia.com/wiki/Metadata Referred: 10.11.2013.

[17] IT knowledge Exchange | Eye on Oracle. 2014. http://itknowledgeexchange.techtarget.com/eye-on-oracle/oracle-the-clear-leader-in-24-billion-rdbms-market/. Referred: 20.03.2014.

[18] Market Share MySQL. 2014. http://www.mysql.com/why-mysql/marketshare/ Referred: 26.03.2014.

[19] Example of relational database. Indiana. http://www.cs.indiana.edu/classes/a114-dger/lastYear/flights.pdf. Referred: 22.03.2014.

[20] How are relational databases utilized in accounting? 2014. http://www.examiner.com/article/how-are-relational-databases-utilized-accounting. Referred: 22.03.2014.

[21] Hughes, Arthur Middleton. 2014. How a Relational Database helps Marketers. Database Marketing Institute. http://www.dbmarketing.com/articles/Art223.htm. Referred: 2.04.2014.

[22] SkySQL. 2012. MySQL vs. MariaDB. Whitepaper.

[23] Lith, Adam, Mattsson, Jakob. 2010. Investigating Storage Solutions for Large Data. Chalmers University of Technology

[24] Harrison, Guy. 2010. NoSQL and Document-Oriented Databases. Database Trends and Applications. Referred: 3.02.2014.

[25] MongoDB. Document Database. 2014. http://www.mongodb.com/document-databases. Referred: 20.02.2014.

[26] Tiwari, Shashank. 2011. Professional NoSQL. John Willey & Sons.

[27] Burd, Greg. 2011. NoSQL. [WhynoSQL]

[28] Your Ultimate Guide to the Non-Relational Universe! http://nosql-database.org/. Referred: 20.02.2014.

[29] Strauch, Christof. NoSQL. Stuttgart Media University.

[30] http://maxgrinev.files.wordpress.com/2010/07/twitterschema-tweets.png. Referred: 28.03.2014.

[31] Introducing JSON. http://json.org/. Referred: 3.4.2014.

[32] MongoDB. JSON and BSON. http://www.mongodb.com/json-and-bson. Referred: 3.4.2014.

[33] XML from inside out O'Reilly Media, Inc. http://www.xml.com/pub/a/98/10/guide0.html?page=2#AEN58. Referred: 3.4.2014.

[34] REST API Tutorial. 2012. http://www.restapitutorial.com/lessons/whatisrest.html. Referred: 3.4.2014.

[35] MongoDB. About MongoDB. 2014. https://www.mongodb.com/mongodb-overview. Referred: 20.02.2014.

[36] Vaish, Gaurav. 2013. Getting started with NoSQL. Packt.

[37] Dyson, Larry. 2012. The Four Horsemen of NoSQL. http://www.modelmetrics.com/technology-viewpoint/the-four-horsemen-of-nosql/. Referred: 16.02.2014.

[38] Neo4j. 2.1 What is a Graph Database? 2014. http://docs.neo4j.org/chunked/stable/what-is-a-graphdb.html Reffered: 16.02.2014.

[39] The four categories of NoSQL databases. 2013. http://rebelic.nl/2011/05/28/the-four-categories-of-nosql-databases/. Referred: 16.02.2014.

[40] MongoDB. 2013. Architecture Guide 2.4.

[41] Rick Osborne, rickosborne.org

[42] Harrison, Guy. 2010. 10 things you should know about NoSQL databases. http://www.techrepublic.com/blog/10-things/10-things-you-should-know-about-nosql-databases/#. Referred: 3.02.2014.

[43] MongoDB. 2014. NoSQL Databases Explained. Whitepaper.

[44] Couchbase. 2013. Why NoSQL. Whitepaper.

[45] Facebook Newsroom http://newsroom.fb.com/company-info/

[46] MongoDB. 2014. Cisco Case Studies. Whitepaper.

[47] MongoDB. 2014. Quantifying Business Advantages. Whitepaper.

[48] Gilbert, Seth, Lynch, Nancy. Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services.

[49] Foundationdb. CAP Theorem. https://foundationdb.com/white-papers/the-cap-theorem. Referred: 16.02.2014.

[50] ACID versus BASE for database transactions. 2009. http://www.johndcook.com/blog/2009/07/06/brewer-cap-theorem-base/. Referred: 16.02.2014.

[51] SkySQL. 2013. MySQL and MariaDB Clustering with Galera. Whitepaper.

[52] Safari Books Online. 2012. Shared-Nothing Architecture. http://blog.safaribooksonline.com/2012/08/21/shared-nothing-architecture/ Referred: 10.11.2013.

[53] ScaleDB. Hogan, Mike. Cluster Architecture. Whitepaper

[54] CodeFutures. 2008. Database Sharding. Whitepaper.

[55] Versant. 2007. Database Scalability and Clustering. Whitepaper.

[56] Song, Huaiming, (ed). A Hybrid Shared-nothing/Shared-data Storage Architecture for Large Scale Databases. Illinois Institute of Technology, Chicago Texas Tech University, Lubbock. [Hybrid Poster ccgrid]

[57] An Introduction to SQL Server Clustes. 2012. http://www.brentozar.com/archive/2012/02/introduction-sql-server-clusters/. Referred: 1.04.2014.

[58] MariaDB Replication Overview https://mariadb.com/kb/en/replication-overview/ Referred: 10.4.2014

[59] MongoDB. Manual. Sharding. http://docs.mongodb.org/manual/core/sharding-introduction/. Referred: 20.03.2014.

[60] MongoDB Manual http://docs.mongodb.org/manual/core/replication-introduction/ Referred: 10.4.2014

[61] Julien Duponchelle, Cassandra MariaDB Virtual Box http://julien.duponchelle.info/Cassandra-MariaDB-Virtual-Box.html Referred: 4.5.2014.

In this part configuration scripts and database schemas can be found.

MariaDB my.cnf derivated form standart my-huge.cnf

```
[client]
#password       = your_password
port            = 3306
socket          = /var/lib/mysql/mysql.sock
# Here follows entries for some specific programs
# The MySQL server
[mysqld]
port            = 3306
socket          = /var/lib/mysql/mysql.sock
skip-external-locking
key_buffer_size = 384M
max_allowed_packet = 1M
table_open_cache = 512
sort_buffer_size = 2M
read_buffer_size = 2M
read_rnd_buffer_size = 8M
myisam_sort_buffer_size = 64M
thread_cache_size = 8
query_cache_size = 32M
# Try number of CPU's*2 for thread_concurrency
thread_concurrency = 8
# Point the following paths to a dedicated disk
#tmpdir         = /tmp/
# Don't listen on a TCP/IP port at all. This can be a security enhancement,
# if all processes that need to connect to mysqld run on the same host.
# All interaction with mysqld must be made via Unix sockets or named pipes.
# Note that using this option without enabling named pipes on Windows
# (via the "enable-named-pipe" option) will render mysqld useless!
#skip-networking
# Replication Master Server (default)
# binary logging is required for replication
log-bin=mysql-bin
# required unique id between 1 and 2^32 - 1
# defaults to 1 if master-host is not set
# but will not function as a master if omitted
server-id       = 1
```

```
# Replication Slave (comment out master section to use this)
# To configure this host as a replication slave, you can choose between
# two methods :
# 1) Use the CHANGE MASTER TO command (fully described in our manual) -
#    the syntax is:
#    CHANGE MASTER TO MASTER_HOST=<host>, MASTER_PORT=<port>,
#    MASTER_USER=<user>, MASTER_PASSWORD=<password> ;
#    where you replace <host>, <user>, <password> by quoted strings and
#    <port> by the master's port number (3306 by default).
#    Example:
#    CHANGE MASTER TO MASTER_HOST='125.564.12.1', MASTER_PORT=3306,
#    MASTER_USER='joe', MASTER_PASSWORD='secret';
#  OR
# 2) Set the variables below. However, in case you choose this method, then
#    start replication for the first time (even unsuccessfully, for example
#    if you mistyped the password in master-password and the slave fails to
#    connect), the slave will create a master.info file, and any later
#    change in this file to the variables' values below will be ignored and
#    overridden by the content of the master.info file, unless you shutdown
#    the slave server, delete master.info and restart the slaver server.
#    For that reason, you may want to leave the lines below untouched
#    (commented) and instead use CHANGE MASTER TO (see above)
# required unique id between 2 and 2^32 - 1
# (and different from the master)
# defaults to 2 if master-host is set
# but will not function as a slave if omitted
#server-id      = 2
# The replication master for this slave - required
#master-host     =   <hostname>
# The username the slave will use for authentication when connecting
# to the master - required
#master-user     =   <username>
# The password the slave will authenticate with when connecting to
# the master - required
#master-password =   <password>
# The port the master is listening on.
# optional - defaults to 3306
#master-port     =  <port>
# binary logging - not required for slaves, but recommended
```

```
#log-bin=mysql-bin
# binary logging format - mixed recommended
#binlog_format=mixed
# Uncomment the following if you are using InnoDB tables
#innodb_data_home_dir = /var/lib/mysql
#innodb_data_file_path = ibdata1:2000M;ibdata2:10M:autoextend
#innodb_log_group_home_dir = /var/lib/mysql
# You can set .._buffer_pool_size up to 50 - 80 %
# of RAM but beware of setting memory usage too high
#innodb_buffer_pool_size = 384M
#innodb_additional_mem_pool_size = 20M
# Set .._log_file_size to 25 % of buffer pool size
#innodb_log_file_size = 100M
#innodb_log_buffer_size = 8M
#innodb_flush_log_at_trx_commit = 1
#innodb_lock_wait_timeout = 50
[mysqldump]
quick
max_allowed_packet = 16M
[mysql]
no-auto-rehash
# Remove the next comment character if you are not familiar with SQL
#safe-updates
[myisamchk]
key_buffer_size = 256M
sort_buffer_size = 256M
read_buffer = 2M
write_buffer = 2M
[mysqlhotcopy]
interactive-timeout
```

Configuration for standard mongod.conf

```
# mongod.conf
#where to log
logpath=/var/log/mongodb/mongod.log
logappend=true
```

```
# fork and run in background
fork=true
#port=27017
dbpath=/var/lib/mongo
# location of pidfile
pidfilepath=/var/run/mongodb/mongod.pid
# Listen to local interface only. Comment out to listen on all interfaces.
bind_ip=127.0.0.1
# Disables write-ahead journaling
# nojournal=true
# Enables periodic logging of CPU utilization and I/O wait
#cpu=true
# Turn on/off security.  Off is currently the default
#noauth=true
#auth=true
# Verbose logging output.
#verbose=true
# Inspect all client data for validity on receipt (useful for
# developing drivers)
#objcheck=true
# Enable db quota management
#quota=true
# Set oplogging level where n is
#   0=off (default)
#   1=W
#   2=R
#   3=both
#   7=W+some reads
#diaglog=0
# Ignore query hints
#nohints=true
# Disable the HTTP interface (Defaults to localhost:27018).
#nohttpinterface=true
# Turns off server-side scripting.  This will result in greatly limited
# functionality
#noscripting=true
# Turns off table scans.  Any query that would do a table scan fails.
#notablescan=true
# Disable data file preallocation.
#noprealloc=true
```

```
# Specify .ns file size for new databases.
# nssize=<size>
# Replication Options
# in replicated mongo databases, specify the replica set name here
#replSet=setname
# maximum size in megabytes for replication operation log
#oplogSize=1024
# path to a key file storing authentication info for connections
# between replica set members
#keyFile=/path/to/keyfile
```

SQL QUERY TEST1

```
SELECT employees.*
FROM employees
JOIN dept_emp ON (dept_emp.emp_no = employees.emp_no)
JOIN salaries ON (salaries.emp_no = salaries.emp_no)
WHERE employees.first_name
LIKE '%Jo%'
AND salaries.from_date > '1993-01-21'
AND salaries.to_date <'1998-01-01'
LIMIT 0, 100
;
```

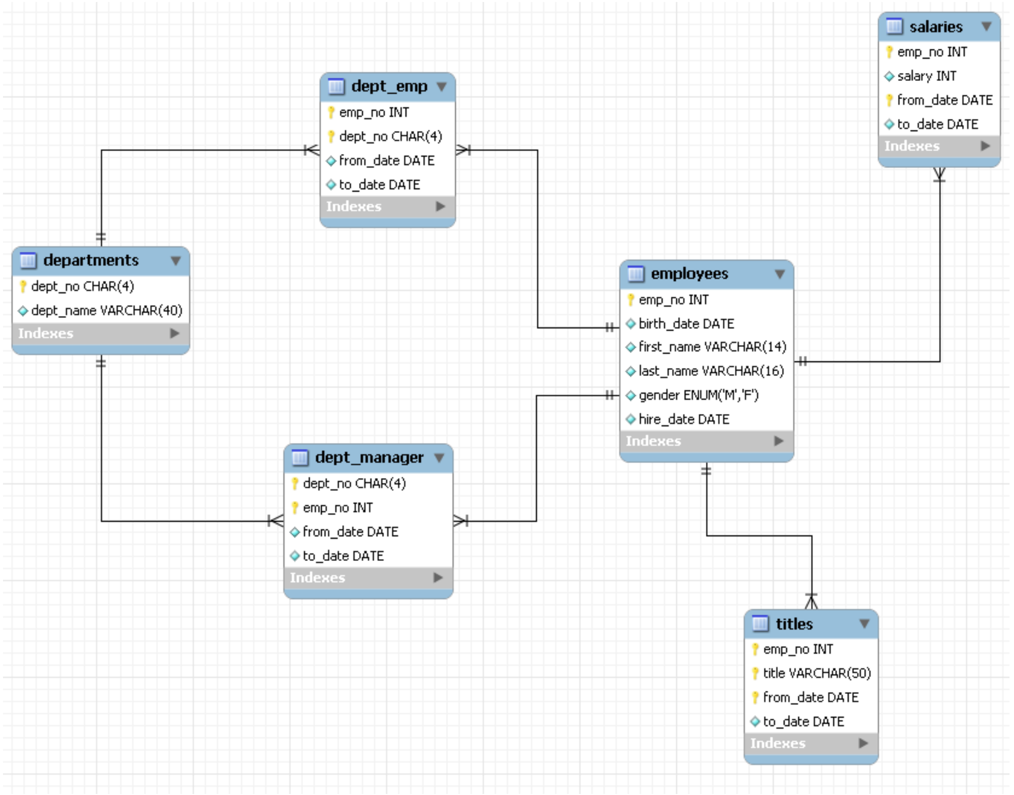


FIGURE 17 Employees Schema