

***UNIVERSITY OF CHITTAGONG***

**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

**Thesis On**

**NOSQL DATABASE (COUCHDB) COMPARISION WITH SQL (MYSQL) DATABASE**

Course code: CSE-422

Supervised By

Md. Mahbubul Islam

Assistant Professor

Department of Computer Science & Engineering

University of Chittagong

Prepared by

Aseem Chakrabarthy

ID: 12205078

Session: 2011-2012

Date of Submission: 8 June, 2017

**Dedication**

**To**

**My Parents**

**&**

**All of My Teachers**

**Declaration**

I declare that this thesis is entirely my own work and has not been submitted in any form to anywhere for any degree.

…………………………………………

Aseem Chakrabarthy (Candidate)

ID: 12205078

Session: 2011-2012

**Approval**

The thesis titled “NOSQL DATABASE (COUCHDB) COMPARISION WITH SQL (MYSQL) DATABASE” submitted by Aseem Chakrabarthy, ID No: 12205078 session: 2011-2012 has been accepted as satisfactory for the fulfillment of the requirements for the degree of B.Sc. Engineering in Computer Science and Engineering, University of Chittagong

…………………………………………

Signature of Supervisor

Md. Mahbubul Islam

Assistant Professor

Computer Science and Engineering

University of Chittagong

Acknowledgements

The successful completion of this thesis work is not a result of my own effort, but is an aggregate of contributions from many other ranging from family members to teachers of Department of Computer Science and Engineering. I’d like to thank my teacher and supervisor Assistant Professor Md. Mahbubul Islam who always inspires me to completed my thesis properly & efficiently. I especially thank all the teachers of CSE department for their excellent supervision and constant guidance throughout my thesis working period. I also thank all the members of my family.

Dated -4 th January, 2017 Aseem Chakrabarthy

**Abstract**

For quite some time relational database, such as MYSQL ,Oracle and Microsoft SQL Server, have been used to store data for most Application .With the Web growing rapidly , spawning enormous ,user-generated content websites such as Facebook ,Twitter, fast database that can handle huge amounts of data are a must .For this purpose new databases management systems collectively called NOSQL are being develop .This thesis explains the NOSQL database ,types of NOSQL ,advantage and disadvantage comparison with MYSQL.NOSQL database is an emerging alternative to the most widely used relational databases.

The massive amounts of data collected today by software in fields varying from academia to business and many other fields, is increasingly becoming a huge problem due to store-age technologies not advancing fast enough to provide the performance scalability needed. This is even more true for data which are highly organized and require analysis while being stored in databases and being accessed by various applications simultaneously. Databases also have the added advantage of providing failover mechanism in case of disruptions or one node failing.

As database vendors struggle to gain more market share new technologies emerge at-tempting to overcome the disadvantages of previous designs while providing more features. Two popular database types, the Relational Database Management Systems and NoSQL databases are examined. The aim of this project was to examine and compare two databases from these two database models and answer the question of whether one performs and scales better than the other.

From the comparison of the results it was found that CouchDB can perform much better for complicated queries at the cost of data duplication which in turn results to a larger database. Also the database size did not appear to be a deciding factor as performance was not crippled significantly for larger database sizes. Writing benchmarks were also run which showed that MySQL performs best at deletion whereas CouchDB excels at inserting documents. The last comparison showed that using sharding to split up the database in CouchDB did not provide a performance advantage which may be related to the routing done by the CouchDB system.

Keywords

Couchdb , couchdb –Java , NoSQL ,Documentation database ,JSON ,DBMS ,MYSQL, Database.

**Table of Contents**

Title Page 1

Dedication 2

Declaration 3

Approval 4

Acknowledgement 5

Abstract 6

Contents ………………………………………………………………………………...7-8

Chapter 1. Introduction……………………………………………………………………9

1.1 Nature and Significance of the Problem ……………………………………………10

1.2 Objective of the Study……………………………………………………………….10

1.3 Study Questions……………………………………………………………………...10

1.4 Limitations of the Research………………………………………………………….10

Chapter 2. Background and Review of Literature

2.1 Database ……………………………………………………………………………...11

2.2 Relational Databases………………………………………………………………..11-12

2.3 ACID properties ………………………………………………………………………..13

2.4 NoSQL Databases ……………………………………………………………………………………………….13-14

2.5 Other databases

2.5.1 Navigation Databases ……………………………………………………14

2.5.2 Object-oriented databases ……………………………………………14-15

2.6 Database Replication ……………………………………………………………15-16

2.7 Database Sharding ……………………………………………………………………17

Chapter [3. Methodology](#page22)

3.1 Java CouchDB Connectivity ………………………………………………………18-25

3.2 CouchDB –Attaching Files

3.2.1 Attaching Files using cURL ………………………………………….25-27

3.2.2 Attaching Files using Futon …………………………………………...27-28

3.3 CouchDB Create Document

3.3.1 Create a document in CouchDB database using Fauxton ……………..27-30

3.3.2 CouchDB create document using cURL utility………………………..30-32

3.4 CouchDB update document

3.4.1 Update (edit) document using fauxton ………………………………………………..32-35

3.4.2 Update (edit) document using utility ……………………………………35-36

3.5 CouchDB delete document

3.5.1 CouchDB delete document using fauxton ………………………………36-38

3.5.2 CouchDB delete document using cURL utility …………………………38-40

3.6 MySQL create database ………………………………………………………….40-41

3.7 MySQL SELECT database ………………………………………………………….42

3.8 MySQL CREATE TABLE ……………………………………………………………...43-45

3.9 Comparison on MySQL with CouchDB database ………………………………45-46

Chapter 4. Conclusion and Recommendations

Conclusion…………………………………………………………………………….47

Reference………………………………………………………………………………48

Chapter 1

INTRODUCTION

Database are defined as a collection of data .Although when using the term database we refer to the complete database system, the term actually refers only to the collection and data. The system which handle Big data, transactions, problems or any other aspect of the database is the Database Management System (DBMS). What follows is a description of the two database types which will be compared in this dissertation .Early designs and implementations were not standardized and required extensive training in order to make efficient use of them. These models and other important types are explained briefly as well.

Databases were created in order to satisfy this need of storing and finding data in an good manner .After their inception in the 1960's different types have invented ,each using its own representation of data. The new generation of applications like Business Intelligence, Web 2.0, Social networking requires processing of terabytes and even petabytes of data. This is achieved by distributed processing. This is one of major reasons for the power of web companies such as Google, Amazon, Salesforce.com, Facebook, Twitter and Yahoo!. Relational databases processing involving very large number of servers and handling Big Data application .There are several reasons for distributed processing. On one hand, programs should be scalable and should take advantage of multiple systems as well as multi-core CPU architectures. On the other end, website servers have to be globally distributed for low latency and failover.We believe that RDBMS systems still have their place and use cases. However, the web has changed the requirements of storage databases systems for the next generation of applications.

Relational Database Management Systems (RDBMSs) can be used to efficiently store and query large amounts of data. However, the performance of an RDBMS can be negatively affected by the requirement of full transactional consistency, where a set of properties known as ACID (for: atomicity, consistency, isolation and durability) are guaranteed by the system. In contrast to RDBMSs, non-relational data stores (NoSQL) are often designed to allow only what is known as eventual consistency to further improve scalability and performance. A comparison between the performance of RDBMSs and NoSQL database systems was conducted, where relaxed consistency was used in the relational systems to decrease overhead. This overhead was previously found to be evenly divided among four components of a RDBMS: logging, locking, latching, and buffer management. However, whether the performance for persisting and querying logs could be enhanced by utilizing a weaker consistency model had not been explored before.

**1.1 Nature and Significance of the Problem**

The problem stems from the fact that NOSQL follows a different approach that is yet to take off unlike ACID, which has been known to provide consistency and reliability to transactions in relational databases. On the other hand, RDBMS (Relational Database Management System) has been around for a while whereas NOSQL is comparably, a new phenomenon. This study will make light into the advantages and disadvantages of using Couchdb and also point out any data integrity in following it.

**1.2 Objective of the Study**

The objective of this study is to highlight compare NoSQL and SQL that with ACID. The study will enlist the positive and negative effects of this approach in NOSQL databases. Any negative effects found will also be evaluated in terms of data integrity of the NOSQL database as a whole.

**1.3 Study Questions**

How does Couchdb NoSQL databases compare to MySQL? How do these differences affect consistency, performance or scalability among others? Do these differences, if any, compromise data integrity more than ACID would?

**1.4 Limitations of the Research**

This research is strictly restricted to existing studies that range from scholarly articles to research white papers to blogs and other credible sources. The research’s aim is not to conduct any specific lab tests or experiments.

Chapter 2

Background

**2.1 Databases**

Databases are defined as organized collections of data. Although when using the term database we refer to the entire database system, the term actually refers only to the collection and the data. The system which handles the data, transactions, problems or any other aspect of the database is the Database Management System (DBMS). What follows is a description of the two database types which will be compared in this dissertation.

Early designs and implementations were based on the use of linked lists to create re-lations between data and to find specific data. These models were not standardized and required extensive training in order to make eﬃcient use of them. These models and other important types are explained briefly as well.

**2.2 Relational Databases**

Relational databases use the notion of databases separated into tables where each column represents a field and each row represents a record. Tables can be related or linked with each other with the use of foreign keys or common columns. On an abstract level tables represent entities, such as users, customers or suppliers. This abstraction is helpful when designing the database schema as real world objects need to be mapped to the database in addition with the relations between them. The design of a database schema can be visualized using diagrams such as the one in Figure 2.1.

UseU

Items

+item\_id: number

+item\_name: text

+price : number

+category: text

+description: text

Users

+user\_id: number

+username: text

+password: text

#email: text

1

Purchases

+sale\_id: number

+item\_id: number

+user\_id: number

+sale\_date: date

+quantity: number

many many

Figure 2.1: Database schema example

An important design aspect of relational databases is the normalization of the schema. This involves 3 steps which are described below:

1. First Normal Form (1NF): Eliminate groups of repeating data by creating a new table for each group of related data which is identified by a primary key.

2. Second Normal Form (2NF): If a set of values are the same for multiple records move them to a new table and link the two tables with a foreign key.

3. Third Normal Form (3NF): Fields which don’t depend on the primary key of a table must be removed and if necessary be put into another table.

It is also necessary for a database schema to satisfy the 2NF to first satisfy 1NF and the same applies for 3NF correspondingly. While there are other forms a schema is considered normalized if it satisfies the above 3 conditions.

**2.3: ACID properties**

An important aspect of relational databases which guarantees the reliability of transactions is their adherence to the ACID properties: Atomicity, Consistency, Isolation, Durability [6,p.405]. Each property is explained below in the context of databases.

Atomicity: Either all parts of a transaction must be completed or none.

Consistency: The integrity of the database is preserved by all transactions. The database is not left in an invalid state after a transaction.

Isolation: A transaction must be run isolated in order to guarantee that any incon-sistency in the data involved does not aﬀect other transactions.

Durability: The changes made by a completed transaction must be preserved or in other words be durable.

**2.4 NoSQL Databases**

NoSQL databases started gaining popularity in the 2000’s when companies began investing and researching more into distributed databases [7]. For this reason the category of NoSQL databases grew and included many subtypes each better suited to specific datasets than others. The most common NoSQL database categories are the following:

**Document stores:** The notion of "documents" is the central concept here with documents being the equivalent of records in relational databases and collections being similar to tables.

**Key-value stores:** Data is stored as values with a key assigned to each value similarly to hash-tables. Also depending on the database a key can have a collection of values.

**Graph databases:** Like graph theory the notion of nodes and edges is the primary concept in graph databases. Nodes correspond to entities such as a user or a music record and edges represent the relations between the nodes. An important aspect which diﬀerentiates graph from relational databases is the use of index-free adjacency, this means each element contains a pointer to its adjacent element and does not require indexing of every element.

An important aspect of NoSQL databases is that they have no predefined schema, records can have diﬀerent fields as necessary, this may be referred to as a dynamic schema. Many NoSQL databases also support replication which is the option of having replicas of a server, this provides reliability as in the case one goes oﬄine the replica would become the primary server. All servers execute the same transactions and synchronize their data in order to eliminate any errors. Also an important diﬀerence between relational databases and NoSQL databases is they do not fully guarantee ACID properties. Their lack of ACID guarantees is attributed to their deployment architecture which typically involves having multiple nodes in order to achieve horizontal scalability and recovery in case of failover. This deployment, which is also referred to as replication, creates issues with synchronization which can result in a secondary node becoming primary but not have an up-to-date copy of the data. NoSQL databases, apart from using an Application Programming Interface(API) or query language to access and modify data, may also use the Map Reduce method which is used for performing a specific function on an entire dataset and retrieving only the result.

**2.5 Other databases**

Beyond relational and NoSQL databases other types have emerged in the past as well.These types have not become as successful as the types examined in this project but have been an important part of the evolution of databases nonetheless.

**2.5.1 Navigational Databases**

Navigational Databases were the first generation of databases developed which worked by using pointers from one record to another. The major downside of navigational databases was the fact that the user would have to know and understand the underlying physical structure of the database in order to query for records. In order for an extra field to be added to a database the entire underlying storage scheme had to be rebuilt. In addition the lack of standardisation among vendors was a disadvantage as it became diﬃcult to choose a suitable implementation.

**2.5.2 Object-oriented databases**

An important part of the database evolution are object-oriented databases which emerged in the 1980s and their main usage was in the object-oriented field such as in conjunction with object-oriented programming languages. In this model information was stored as objects which could accommodate for a large number of data types and could also oﬀer advanced features such as inheritance and polymorphism which were characteristic of object-oriented programming languages as well. The disadvantages which prevented object-oriented databases from gaining more popularity was the need to rebuild an entire database in order to migrate from another database management system and also the issue that most of these databases were bound to a specific programming language. Object Relational Database Management System & Object Relational Mapping .In order to overcome the disadvantages of object-oriented database and use the advantages of relational databases the Object Relational Database Management System emerged. This system attempted to bring together the features of RDBMSs and object-oriented modelling techniques which were used in popular programming languages. This however was not successful with a more popular approach being the Object Relational Model whereby a relational model database was used with an object relational mapping software which allowed developers to integrate their own types and their methods into the DBMS.

**2.6 Database Replication**

Database Replication is the practice of deploying multiple servers which are clones of each other. This practice is used in NoSQL databases often in order to provide higher reliability and performance. CouchDB recommends deploying a replica set, which is a set of replica servers, in all production deployments.

In CouchDB replication is deployed using a primary-secondary server configuration whereby one server is the primary and all others are secondary. The primary server, or primary replica, handles all write operations and logs them in a special collection where the secondaries read and apply them. Secondary replica servers can also read the operations from another secondary thus limiting the amount of load on the primary server. This processes are shown in Figure 2.2. By having a copy of the data replica servers not only provide more reliability but can also be used for read operations from client applications, this can have a massive performance eﬀect but also carries with it the chance that the data provided will not be the most up-to-date. The reason that data is not always up to date is due to the asynchronous replication model which CouchDB uses. This asynchronous replications provides better reliability as secondary members can continue to function when another member of the replica set is unavailable but prohibits the guarantee of the ACID properties as a secondary replica which becomes primary may not be up to date thus violating the Consistency and Durability properties.

Couchdb

Instance

Master

Couchdb

Instance

slave

D

Synchronization

Write Read Read

User application

Figure 2.2: CouchDB Replication deployment and usage

In the case when the primary replication fails for any reason and goes oﬄine voting takes place to promote a secondary replica to primary, this is when the concept of an arbiter server becomes relevant. An arbiter server does not handle any load or store data but only exists to provide a vote in these elections, this is necessary in cases where there is an even number of secondary replicas and the voting could result in a draw.

MySQL Cluster also supports replication but is currently limited to a main replica server and a slave replica server at maximum. This limitation can be alleviated with the use of node clusters which work as replicas of other node clusters. In addition, the default configuration does not allow direct connections to the replication in order to distribute the load and also the use of replication in clustering instead of simple Master-Slave model requires the use of the NDB engine to cluster successfully. The MySQL implementation of the Master-Slave model is similar to CouchDB as it is used to send write operations to the

**2.7 : Database Sharding**

Sharding is the term used to describe practice of using multiple servers of the same database and configuring them in order for the data stored in the database to be split or separated to diﬀerent machines. This allows increased performance as each server handles diﬀerent sets of data thus if a single database becomes too large its performance may diminish due to the increased time a query takes. Despite the obvious advantage of added performance database, replication is recommended over sharding as it provides both performance and reliability. Specifically CouchDB developers suggest to deploy databases without sharding and only shard when the data set increases in size. In addition, CouchDB shards can be accessed directly but the transactions which can be performed are limited to the collections which are sharded on the specific shard.

MySQL Cluster handles sharding automatically but is limited to using the NDB storage engine. The NDB storage engine is limited mainly due to its lack of foreign keys support and a storage limit of 3TB compared to 64TB of the the storage engine making it unsuitable for the benchmarking harness which uses foreign keys.

**Chapter 3**

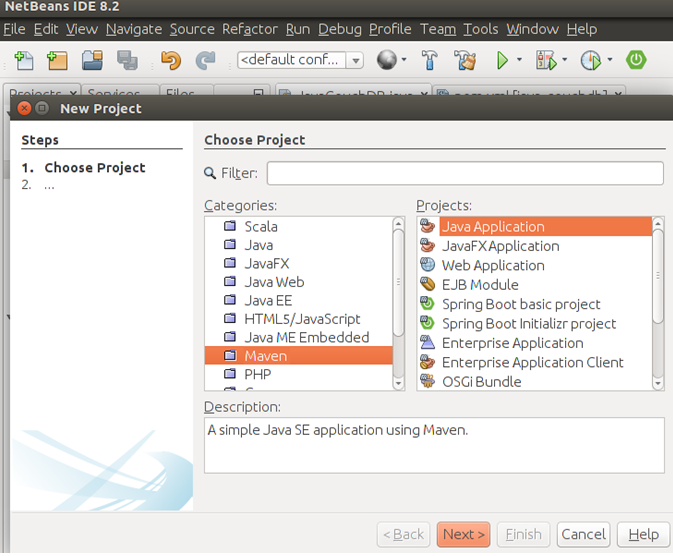
**Methodology**

**3.1 Java CouchDB Connectivity**

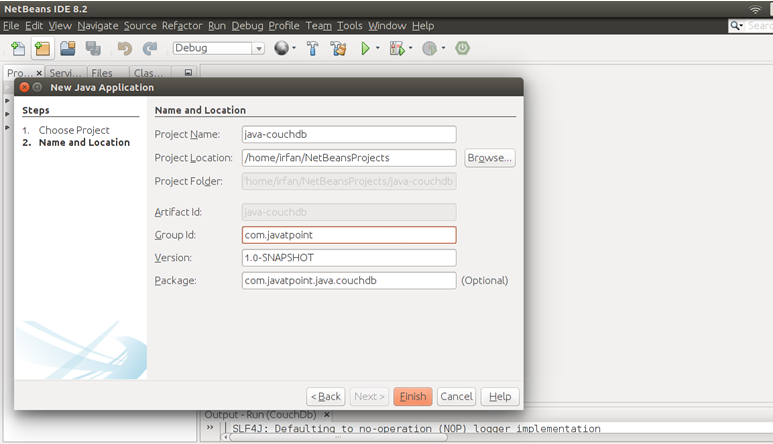
We can connect to the CouchDB with the Java Programming language. To connect, we are using Ektorp library that provides a persistence layer on the top of CouchDB.

Here, we are explaining an example, in which we are connecting, creating database etc.

This example is created in NetBeans IDE 8.2.First create a maven project and then follow the following steps.



Providing a name for our project….



After finishing, see, it has a pom.xml file. We need to add dependency in this file. let's do it first.

// Ektorp library dependency

<dependency>

<groupId>org.ektorp</groupId>

<artifactId>org.ektorp</artifactId>

<version>1.4.4</version>

</dependency>

// pom.xml

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.javatpoint</groupId>

<artifactId>java-couchdb</artifactId>

<version>1.0-SNAPSHOT</version>

<packaging>jar</packaging>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<maven.compiler.source>1.8</maven.compiler.source>

<maven.compiler.target>1.8</maven.compiler.target>

</properties>

<dependencies>

<dependency>

<groupId>org.ektorp</groupId>

<artifactId>org.ektorp</artifactId>

<version>1.4.4</version>

</dependency>

</dependencies>

</project>

After adding dependency, create a Java file for connection. Our Java file contains the necessary connection code .

// JavaCouchDB.java

package com.javatpoint.java.couchdb;

import java.net.MalformedURLException;

import org.ektorp.CouchDbConnector;

import org.ektorp.CouchDbInstance;

import org.ektorp.http.HttpClient;

import org.ektorp.http.StdHttpClient;

import org.ektorp.impl.StdCouchDbConnector;

import org.ektorp.impl.StdCouchDbInstance;

import org.ektorp.support.DesignDocument;

public class JavaCouchDB {

public static void main(String[] args) throws MalformedURLException {

//--------------- Creating Connection--------------------------//

HttpClient httpClient = new StdHttpClient.Builder()

.url("http://localhost:5984")

.build();

CouchDbInstance dbInstance = new StdCouchDbInstance(httpClient);

//--------------- Creating database----------------------------//

CouchDbConnector db = new StdCouchDbConnector("javatpoint", dbInstance);

db.createDatabaseIfNotExists();

//--------------- Creating Document----------------------------//

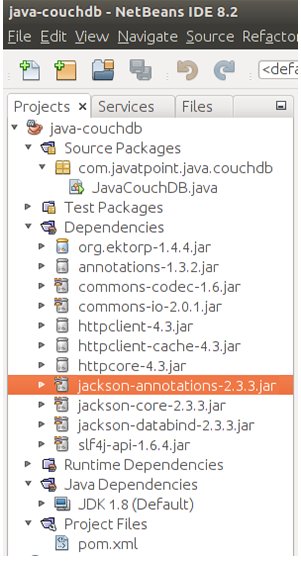
DesignDocument dd = new DesignDocument("light");

db.create(dd);

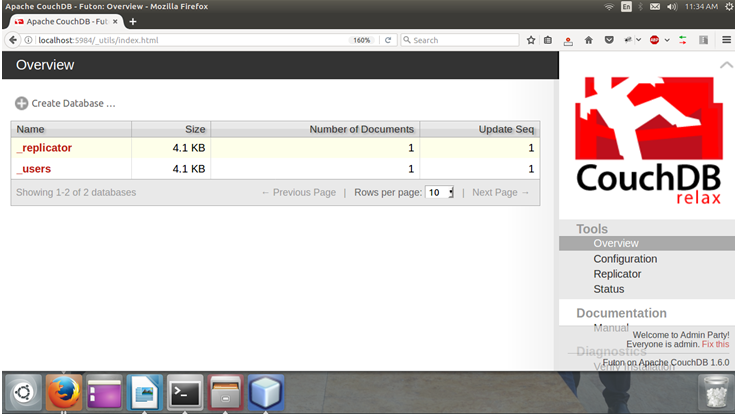
}

}

Our project looks like this:

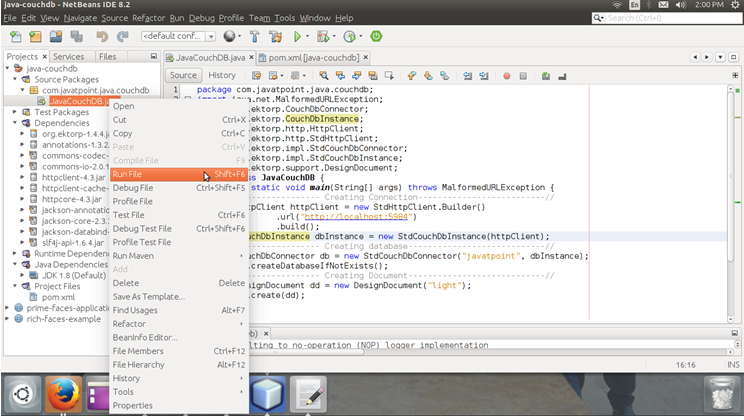


Now before executing Java code, check the CouchDB connection is working or not. To check it follow this URL **http://localhost:5984/\_utils/**. It will display all the available databases.

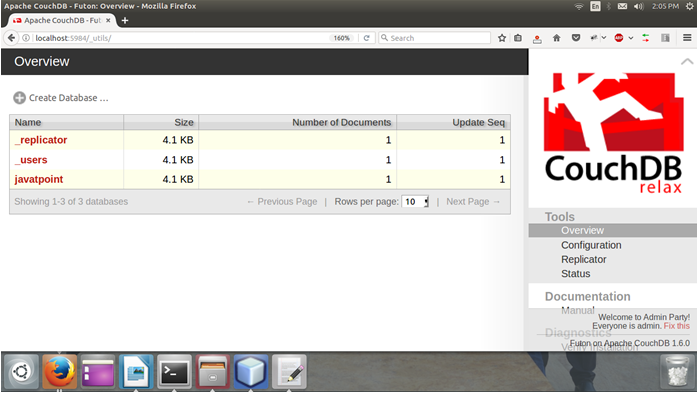


This screen-shot shows the CouchDB index web page.

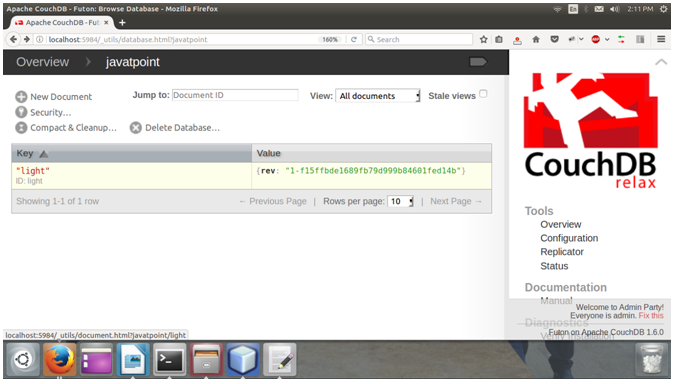
Now, execute the Java code that will create a database. We can see that database in the databases list shown at the index web page.



Check the CouchDB index web page.



See, there is three databases including **javatpoint** new one. It contains a document**light** shown as below.



**3.2 CouchDB – Attaching Files**

**3.2.1 Attaching Files using cURL**

You can attach files to CouchDB just like email. The file contains metadata like name and includes its MIME type, and the number of bytes the attachment contains. To attach files to a document you have to send PUT request to the server. Following is the syntax to attach files to the document −

$ curl -vX PUT http://127.0.0.1:5984/database\_name/database\_id

/filename?rev=document rev\_id --data-binary @filename -H "Content-Type:

type of the content"

The request has various options that are explained below.

–data-binary@ – This option tells cURL to read a file’s contents into the HTTP request body.

-H – This option is used to mention the content type of the file we are going to upload.

**Example**

Let us attach a file named boy.jpg, to the document with id 001, in the database named my\_database by sending PUT request to CouchDB. Before that, you have to fetch the data of the document with id 001 to get its current rev id as shown below.

$ curl -X GET http://127.0.0.1:5984/my\_database/001

{

"\_id": "001",

"\_rev": "1-967a00dff5e02add41819138abb3284d"

}

Now using the \_rev value, send the PUT request to the CouchDB server as shown below.

$ curl -vX PUT http://127.0.0.1:5984/my\_database/001/boy.jpg?rev=1-

967a00dff5e02add41819138abb3284d --data-binary @boy.jpg -H "ContentType:

image/jpg"

**Verification**

To verify whether the attachment is uploaded, fetch the document content as shown below−

$ curl -X GET http://127.0.0.1:5984/my\_database/001

{

"\_id": "001",

"\_rev": "2-4705a219cdcca7c72aac4f623f5c46a8",

"\_attachments": {

"boy.jpg": {

"content\_type": "image/jpg",

"revpos": 2,

"digest": "md5-9Swz8jvmga5mfBIsmCxCtQ==",

"length": 91408,

"stub": true

}

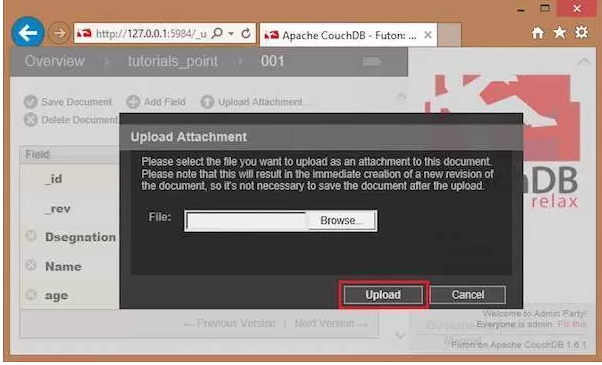
}

}

**3.2.2 Attaching Files using Futon**

Upload Attachment

Using this option, you can upload a new attachment such as a file, image, or document, to the database. To do so, click on the Upload Attachment button. A dialog box will appear where you can choose the file to be uploaded. Select the file and click on the Upload button.



The file uploaded will be displayed under \_attachments field. Later you can see the file by clicking on it.

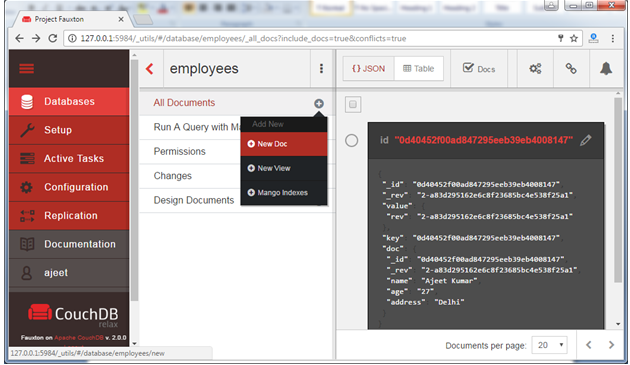
**3.3 CouchDB Create Document**

In CouchDB, data of the database is stored in the form of documents instead of tables.

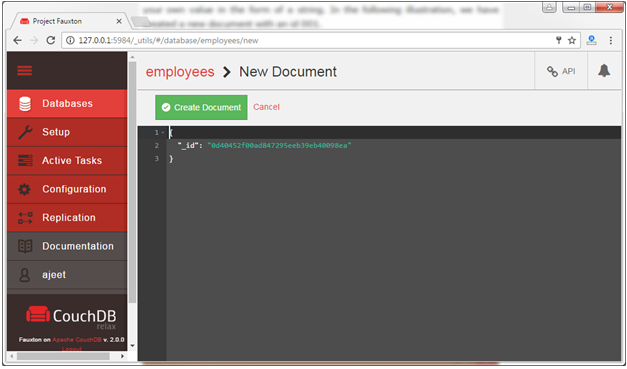
**3.3.1 Create a document in CouchDB database using Fauxton**

To create a document in CouchDB database open the Fauxton url:

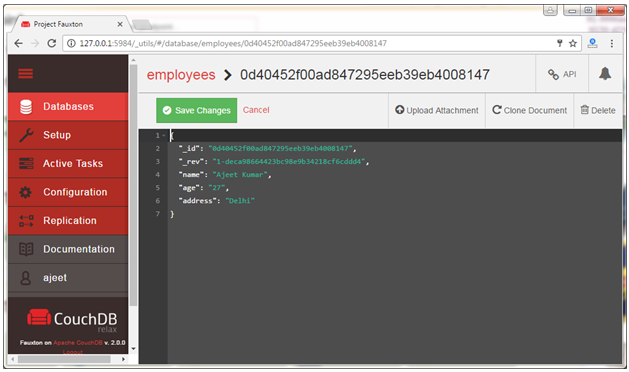
<http://127.0.0.1:5984/_utils/> Choose the specific database and put your cursor on all documents tab. Click on new Doc as shown in the below image:



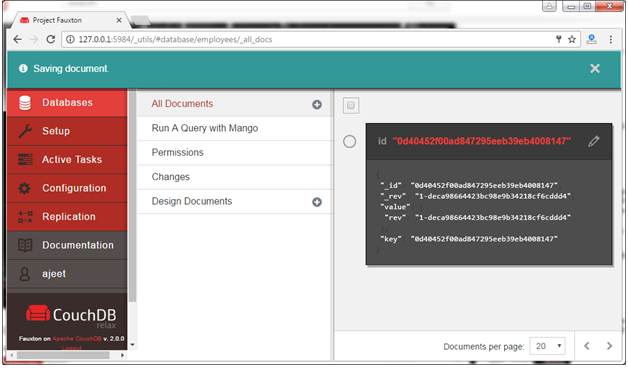
You will a page like:



Fill the entries which you want in your documents:



After all entries, click on the save changes tab. Now the document is created.



**3.3.2 CouchDB Create Document using cURL Utility**

To create a document in CouchDB, send an HTTP request to the server using PUT method through cURL utility.

Following is the syntax to create a document.

Syntax:

curl -X PUT http://127.0.0.1:5984/database name/"id" -d ' { document} '

Note: -X is used to specify a custom request method of HTTP that we use to communicate with the HTTP server. In this case, we are using PUT method. When we use the PUT method, the content of the url specifies the object name we are creating using the HTTP request.

i) The database name specifies the name of the database in which we are creating the document.

2) Id specifies the document id.

3) The data of the document. ?d option is used to send the data/document through HTTP request.

{

Name : Ajeet

age : 23

Designation : Designer

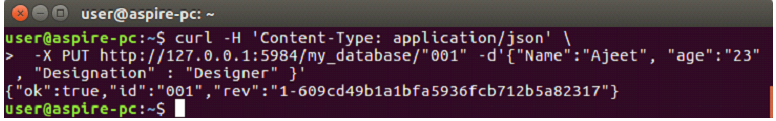
}

Example:

Let's create a document with id "001" with database name javatpoint.

curl -H 'Content-Type: application/json' \

-X PUT http://127.0.0.1:5984/my\_database/"001" -d'{"Name":"Ajeet", "age":"23" , "Designation" : "Designer" }'

The response of CouchDB for this command contains 3 fields:

i) "ok": It specifies that the operation is successful.

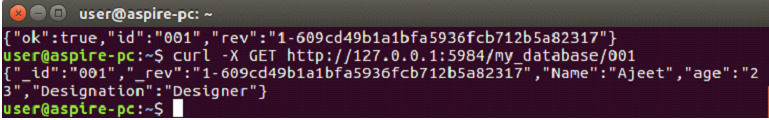
ii) "id": It stores the id of the document.

iii) "rev": This indicates the revision id. Every time you revise (update or modify) a document a \_rev value will be generated by CouchDB. If you want to update or delete a document, CouchDB expects you to include the \_rev field of the revision you want to change. When CouchDB accepts the change, it will generate a new revision number.

**Verification**

You can verify that your document is created by using the following command:

curl -X GET http://127.0.0.1:5984/my\_database/001

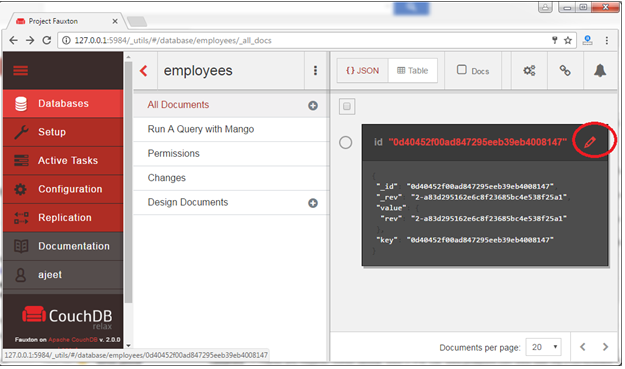
****

**3.4 CouchDB Update Document**

**3.4.1 Update (edit) document using fauxton**

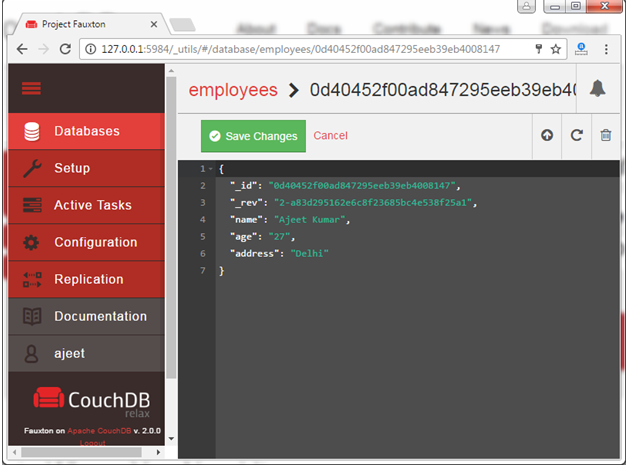
**Open the Fauxton url:http://127.0.0.1:5984/\_utils/**

You can also update/ change/ edit your document once you created. Click on the edit option (encircled in red).

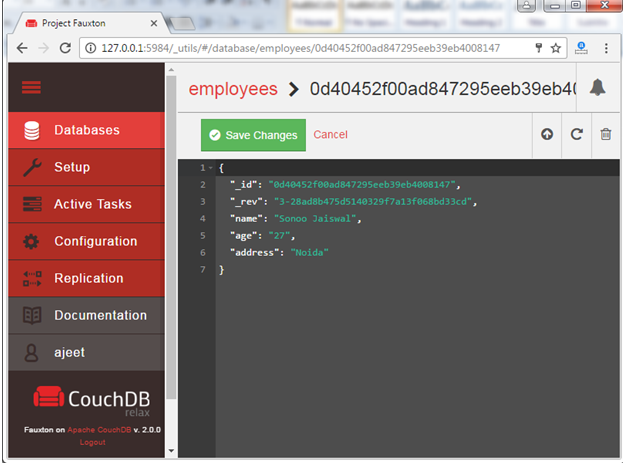
****

After clicking, you will get a new page where you can edit your entries. After editing click on the save changes tab and your document will be updated.

**For example:** In the below example employee name is Ajeet Kumar, Age is 27 and address is Delhi.



Now, update the name Ajeet Kumar to Sonoo Jaiswal and address Delhi to Noida.



Click on the "save changes" tab and your document will be updated.

**3.4.2 CouchDB Update Document Using cURL utility**

cURL facilitates users to update the document in CouchDB by sending an HTTP request to the server using PUT method through cURL utility.

**Syntax:**

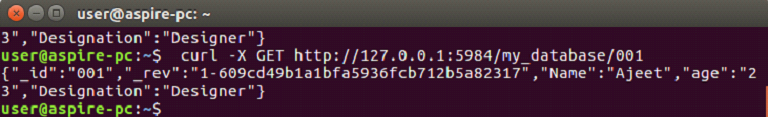
curl -X PUT http://127.0.0.1:5984/database\_name/document\_id/ -d '{ "field" : "value", "\_rev" : "revision id" }'

**Example:**

We have a document with id 001 in the database named my\_database.

First of all, retrieve the revision id of the document that is to be updated. You can find the \_rev of the document in the document by using the following command:

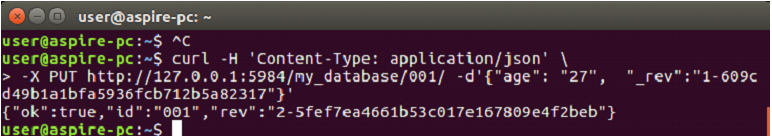
curl -X GET http://127.0.0.1:5984/my\_database/001



Use revision id \_rev from the document to change the data. Here we change the age from 23 to 27.

curl -H 'Content-Type: application/json' \

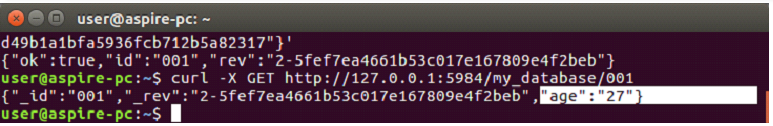
-X PUT http://127.0.0.1:5984/my\_database/001/ -d'{"age": "27",  "\_rev":"1-609cd49b1a1bfa5936fcb712b5a82317"}'



**Verification**

You can verify the document if it is updated or not by using again the GET request.

curl -X GET http://127.0.0.1:5984/my\_database/001



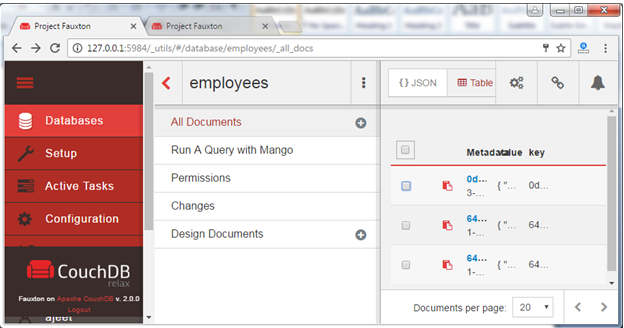
Here you can see that age is now updated.

**3.5 CouchDB Delete Document**

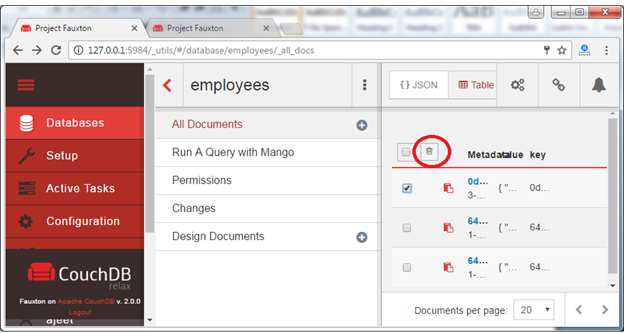
**3.5.1 CouchDB Delete Document using Fauxton**

Open the Fauxton url:http://127.0.0.1:5984/\_utils/

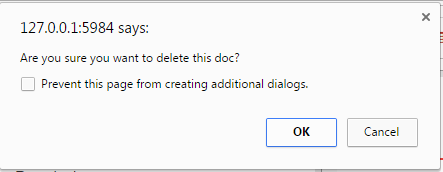
First select the database which contains all your documents. Here is all three documents in database "employees"



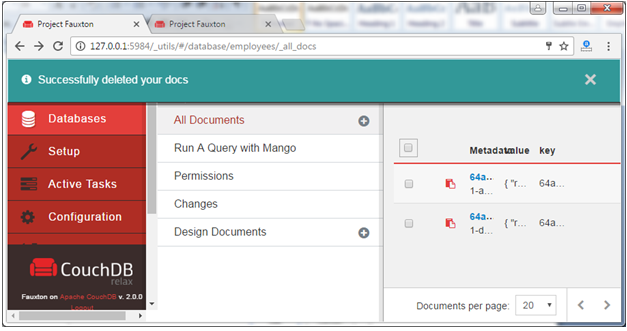
Select the specific document which you want to delete and click on the delete icon encircled in red.



You will see a pop-up message that "Are you sure you want to delete this doc?" Click ok.



Now the selected document is deleted. You can see here only two documents.



**3.5.2 CouchDB Delete Document Using cURL utility**

CouchDB facilitates you to delete a document by sending an HTTP request to the server using DELETE method through cURL utility.

**Syntax:**

curl -H 'Content-Type: application/json' \

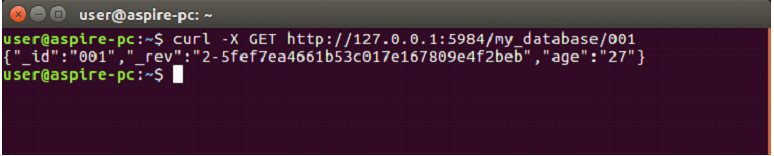
-X DELETE http://127.0.0.1:5984/database\_name/database\_id?\_rev

Note: -X is used to specify a custom request method of HTTP which is used while communicating with the HTTP server. To delete a dcoment, we have to pass the recent revision id through the url with /database\_name/database\_id/. "?" is used to mention attributes of any data structure.

**Example:**

We have a document in our database named "my\_database" with id 001. To delete this document first get the rev id of the document by using the following command:

curl -X GET http://127.0.0.1:5984/my\_database/001

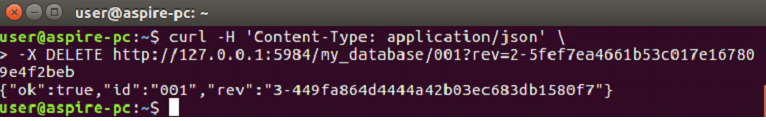


ere rev id is "2-5fef7ea4661b53c017e167809e4f2beb".

Now use the following command to delete the document having the id "001".

curl -H 'Content-Type: application/json' \

-X **DELETE** http://127.0.0.1:5984/my\_database/001?rev=2-5fef7ea4661b53c017e167809e4f2beb



The response of CouchDB for this command contains 3 fields:

i) "ok": It specifies that the operation is successful.

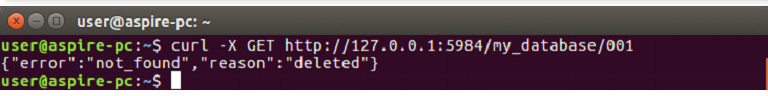
ii) "id": It stores the id of the document.

iii) "rev": This indicates the revision id. Every time you revise (update,modify or delete) a document a \_rev value will be generated by CouchDB. If you want to update or delete a document, CouchDB expects you to include the \_rev field of the revision you want to change. When CouchDB accepts the change, it will generate a new revision number.

**Verification:**

You can verify that your document is deleted by using the following command:

curl -X GET http://127.0.0.1:5984/my\_database/001

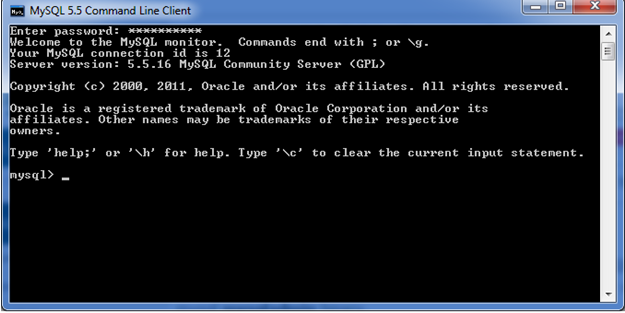
****

Here, you can't see the document because it is deleted.

**3.6 MySQL Create Database**

You can create a MySQL database by using MySQL Command Line Client.

Open the MySQL console and write down password, if you set one while installation. You will get the following:



Now you are ready to create database.

**Syntax:**

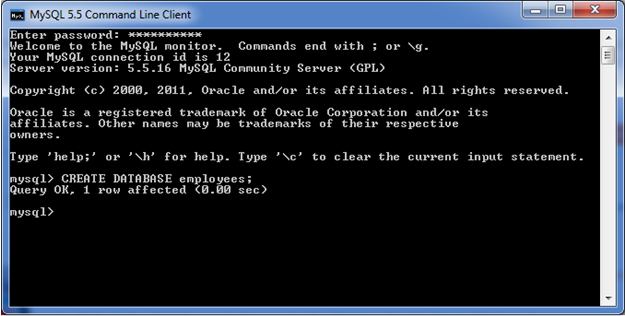
CREATE DATABASE database\_name;

**Example:**

Let's take an example to create a database name "employees"

CREATE DATABASE employees;

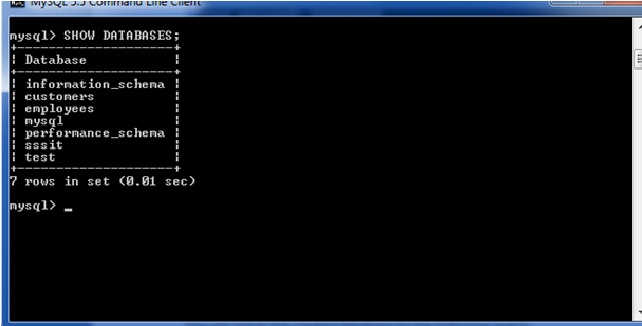
It will look like this:



You can check the created database by the following query:

SHOW DATABASES;

Output



Here, you can see the all created databases.

Note: All the database names, table names and table fields name are case sensitive. You must have to use proper names while giving any SQL command.

**3.7 MySQL SELECT Database**

SELECT Database is used in MySQL to select a particular database to work with. This query is used when multiple databases are available with MySQL Server.

You can use SQL command USE to select a particular database.

**Syntax:**

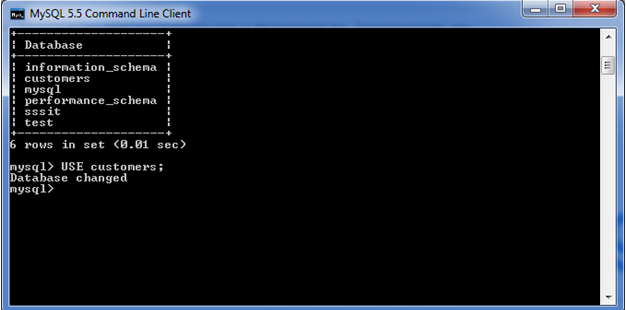
USE database\_name;

**Example:**

Let's take an example to use a database name "customers".

USE customers;

It will look like this:



Note: All the database names, table names and table fields name are case sensitive. You must have to use proper names while giving any SQL command.

**3.8 MySQL CREATE TABLE**

The MySQL CREATE TABLE command is used to create a new table into the database. A table creation command requires three things:

* Name of the table
* Names of fields
* Definitions for each field

**Syntax:**

Following is a generic syntax for creating a MySQL table in the database.

CREATE TABLE table\_name (column\_name column\_type...);

**Example:**

Here, we will create a table named "cus\_tbl" in the database "customers".

CREATE TABLE cus\_tbl(

cus\_id INT NOT NULL AUTO\_INCREMENT,

cus\_firstname VARCHAR(100) NOT NULL,

cus\_surname VARCHAR(100) NOT NULL,

PRIMARY KEY ( cus\_id )

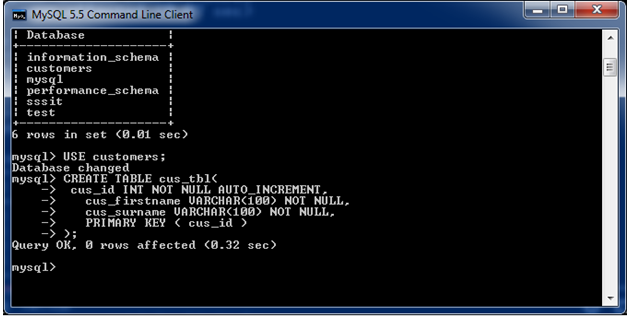
);

Note:

i) Here, NOT NULL is a field attribute and it is used because we don't want this field to be NULL. If you will try to create a record with NULL value, then MySQL will raise an error.

ii) The field attribute AUTO\_INCREMENT specifies MySQL to go ahead and add the next available number to the id field. PRIMARY KEY is used to define a column as primary key. You can use multiple columns separated by comma to define a primary key.

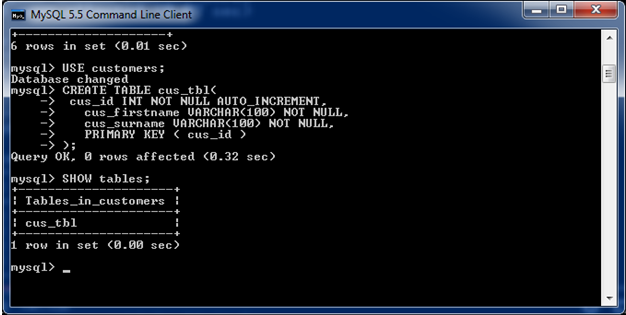
**Visual representation of creating a MySQL table:**

****

**See the created table:**

Use the following command to see the table already created:

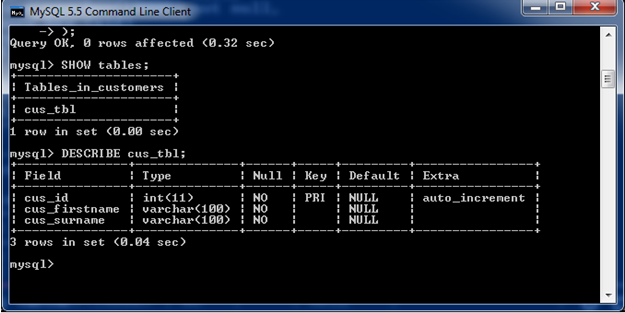
SHOW tables;



**See the table structure:**

Use the following command to see the table already created:

DESCRIBE cus\_tbl;



**3.9 Comparison on MySQL with CouchDB Database**

CouchDB MySQL

Data Model Document-Oriented (JSON) Relational

Large Object (Files) Yes (attachments) Blobs

Horizontal Partioning CouchDB Lounge Partitioning

Scheme

Replication Master-master (with developer Master-slave,multi-master and circular

Supplied conflict resolution) replication

Data Types string, number, boolean,array,object link

Query method Map/reduce of javascript function Dynamic;SQL

To lazily build an index per query

Object storage One large repository Table-based

Secondary Indexes Yes Yes

Interface REST Native Drivers

Written in Erlang c++

Concurrency Control MVCC

Consistency concepts Eventual Consistency Intermediate Consistency

Foreign keys no Yes

Transaction concepts no ACID

Durability yes yes

In-memory capabilities no yes

Supported Programming C,C#,ColdFusion,Erlang,Haskell,Java, C,C++,C#,D,Delphi,Eiffel,Erlang,Java,

Language Javascript,Perl,PHP,PL/SQL,Ruby,Python Perl,Python,Ruby,Scheme,oCaml,

Smalltalk,Lua,Lisa,oCaml Javascript,Objective-C

**Chapter 4**

**Conclusion**

Relational Database Management system won’t go away for big databases application because have a drawback to our applications like time consuming and execution speed and scaling of queries, they are define compulsory. But the storage requirement for the new generation of applications are huge different from heritage applications. We can choose NoSQL (CouchDB) instead of MySQL because of two factors, ease of use and timing performance. We conclude that if your web application is data intensive and stores lots of big-data, queries lots of data, and generally lives database, then you better do that efficiently or have resources (i.e. money) to burn .it’s a worth doing for all organizations for developing applications Lastly, the report concludes by method define a database integration method by using a middleware between the two layers. In this method, application does not have to consider about the complexity of underlying database layer with CouchDB Databases .there data distribution and storage.They have to use the basic SQL query language to get result from the database and all the format conversion rules will be done by the Metadata because data fetch from database in the form of JSON format. The system was proposed because CouchDB has newly come into existence, whereas the standard SQL language has been over years and, therefore if we merge the two we can use the features of both the database. Although, NoSQL(CouchDB) has the advantage of horizontal expansion, but for complex SQL requests, it cannot support them very well. For the Query based on KEY/VALUE and massive data storage requirements, NOSQL is a very worth doing choice for me and all other developers and organizations who’s developed big applications.

Reference

[1] S. Weber, “NoSQL Databases,” University of Applied Sciences HTW Chur, Switzerland, 2010.

[2] V. Sharma and M. Dave, “SQL and NoSQL Databases,” International Journal of Advanced Research in Computer Science and Software Engineering, vol. 2, no. 8, pp. 20 - 27, 2012.

[3] N. Leavitt, “Will NoSQL Database Live Up to Their Promise?,” IEEE computer society, vol. 10, no. 9162, pp. 12 - 14, 2010.

[4] C. Nance and T. Losser, “NOSQL VS RDBMS - WHY THERE IS ROOM FOR BOTH,” in Proceedings of the Southern Association for Information Systems Conference., Savannah, GA, USA, 2013.

[5] “NoSQL databases,” [Online]. Available: nosqldatabase.org. [Accessed 10 6 2013].

[6] P. W. Kriha, “NoSQL Databases,” [Online]. Available: www.christof-strauch.de/nosqldbs.pdf. [Accessed 2 2013].

[7] Lior Okman, Nurit Gal-Oz, Jenny Abramov,”Security Issues in NoSQL databases”,IEEE, 978-0-7695-4600-1, 2011

[8] Berg K., Seymour T., and Coel R. History of Databases. International Journal of Management and Information Services, 17, 2013.

[9] https://www.mysql.com/

[10] https://en.wikipedia.org/wiki/CouchDB