2018

# Document stores using CouchDB

ADVANCED DATABASE PROJECT

APARNA KHIRE, MINGRUI DONG

aparna.khire@vub.be, mingdong@ulb.ac.be

# Table of Contents

# 1. Introduction

With the onset of different kinds of applications working with different forms of data introduced on the web, it is required to introduce different kind of database, as well, to store and manage this data. With this, we also want databases that can be scalable and distributed to handle large amount of such data.

In this project, we study one such database, the document stores database using CouchDB. We compare this DB with the traditional DBMS like MySQL, in terms of the processing time for each type of query and the complexity of creating such queries.

We will consider the dataset from the food delivery app – Zomato. In this dataset, reviews for all registered restaurants are considered.

# 2. Background

If the data requirements are not clear at the inception of the project or if we are dealing with massive amounts of unstructured data, it will be very difficult, if not impossible, to develop a relational database with clearly defined schema. For this reason, the non-relational databases came into picture. It offers greater flexibility than their traditional counterparts. Non-relational databases are like file folders, assembling related information of all types. In the below Fig. 1, we can see an example of an invoice and contact card. Both these documents may have different structured data for each customer.

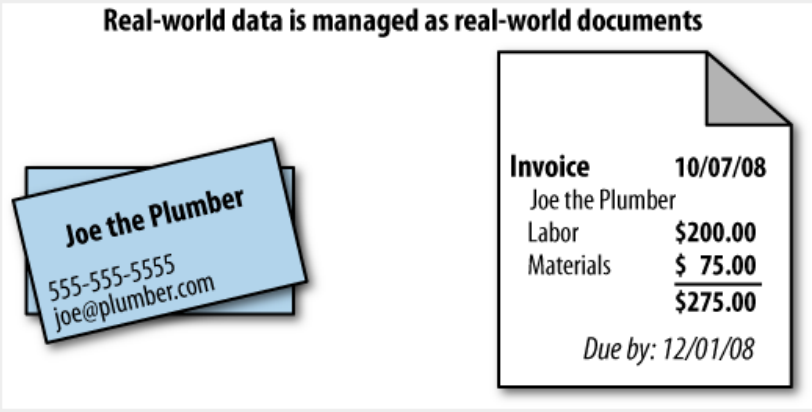



*Figure 1: Image Src - couchDB.org*

## 2.1 NoSQL Database

Unstructured data from the web can include social media sharing, personal settings, photos, location-based information, review metrics, usage metrics, and more. This type of unstructured data is stored, processed, and analysed using schema-less alternatives as NoSQL, meaning “Not only SQL.” While the term NoSQL encompasses a broad range of alternatives to relational databases, what they have in common is that the data can be maintained with more flexibility.

Instead of tables, NoSQL [1] databases are document-oriented. This way, non-structured data (such as articles, photos, social media data, videos, or content within a blog post) can be stored in a single document that can be easily found but is not necessarily categorized into fields like a relational database does.

NoSQL databases are widely recognized for their ease of development, functionality, and performance at scale. They use a variety of data models, such as:

- Key-value
- Column
- Document
- Graph
- In-memory
- Search

In this project, we will experiment with one such NoSQL database, the Document store using CouchDB.

# 3. Document Stores

## 3.1 What is Document-stores?

It is a database that uses a document-oriented model to store data.

Document [2] databases store each record and its associated data within a single document. Each document contains semi-structured data that can be queried using languages such as XQuery, XSLT, SPARQL, Java, JavaScript, Python, etc.

While each document-oriented database implementation differs, but in general, they all assume documents encapsulate and encode data (or information) in some standard format or encoding. Encodings mostly used are - XML, YAML, JSON, and BSON, as well as binary forms like PDF and Microsoft Office documents (MS Word, Excel, and so on). In our project, we are using the JSON format for data encoding.

```xml
<authors>
   <authorname>Sonja Yoerg</authorname>
   <books>
     <book>
       <bookname>True Places</bookname>
       <datereleased>2018</datereleased>
       <genre>Fiction</genre>
     </book>
     <book>
       <bookname>All the best People</bookname>
       <datereleased>2017</datereleased>
       <genre>Fiction</genre>
     </book>
     <book>
       <bookname>House Broken</bookname>
       <datereleased>2015</datereleased>
       <genre>Fiction</genre>
     </book>
   </books>
 </authors>
```

*Figure 2: XML form of a Document*

```json
{
    '_id' : 1,
    'authorName' : { 'Sonja Yoerg' },
    'books' : [
        {
            'bookname' : 'True Places',
            'datereleased' : 2018,
            'genre' : 'Fiction'
        }, {
            'bookname' : 'All the best People',
            'datereleased' : 2017,
            'genre' : 'Fiction'
        }, {
            'bookname' : 'House Broken',
            'datereleased' : 2015,
            'genre' : 'Fiction'
        }
    ]
}
```

*Figure 3: JSON form of the Document*

Some of the leading document store databases are MongoDB, CouchBase, CouchDB, MarkLogic, OrientDB, etc.

## 3.2 Document Store vs. Relational Database

### 3.2.1 Storage Structure

As mentioned in the above explanations, data is stored in a table, in relational database systems, whereas the data is stored in documents in the document stores.

### 3.2.2 Schemas

With relational databases, you must create a schema before you load any data. With document store databases, you have no such requirement. The data can be loaded without any predefined schema. Thus, with a document store, any two documents can contain a different structure and data type.

For example, if a user is supposed to fill a contact information form and he wishes to enter a Fax number along with a mobile number, then a document with both the numbers can be created. On the other hand, if another user does not have a Fax machine and he provides only the mobile number, the document created in this case will not have field called Fax number at all. But if this was a relational database, fax number would still be a field for both users – it would just contain a value or null in the above scenario. In other words, even if only one user has a value of fax number, this would be a field required in the table, having a lot of null values.

### 3.2.3 Scalability

Document databases can scale very well, horizontally. Data can be stored over a large distributed system and it will perform well.

Relational databases are not well suited to scale in this fashion. But are more suited towards scaling vertically (i.e. adding more memory, storage, etc). But this vertical scaling can reach up a certain limit.

### 3.2.4 Relationships

Document stores do not have foreign keys, like relational databases. Foreign keys are used by relational databases to enforce relationships between tables. If a relationship needs to be established with a document database, it would need to be done at the application level. However, in the document model, there is no need to establish this relationship, as any data associated with a record is stored within the same document.

## 3.3 Document Store vs Key-Value Databases

Document databases are similar to key-value databases, as in, there is a key and a value. Data is stored as a value. Its associated key is the unique identifier for that value.

The difference is that, in a document database, the value contains structured or semi-structured data. In key-value databases, these store all the data in multiple key-value pairs without having any relation between them. But, document databases maintain sets of key-value pairs within a document. That is, similar key-value pairs are stored in a single document. This is helpful when we have complex queries. For example, store all the orders having fields such as name, shipping address, order quantity, price, etc. within the same document.

Document databases also support indexing, which can improve query performance by using filter criteria. In the above example, a query like searching for all orders placed in the last 10 days.

# 4. CouchDB

CouchDB is an open source NoSQL database based on common standards to facilitate Web accessibility and compatibility with a variety of devices. CouchDB is written in Erlang, a computer language highly optimized for concurrency, distribution and fault tolerance.

Data in CouchDB is stored in the JavaScript object notification (JSON) format and organized as key-value pairs. The key is a unique identifier of the data and the value is the data itself or a pointer to the data's location. Standard database functions are all performed by JavaScript. OS-agnostic, device-independent Web standards allow databases to perform well for the widest variety of users.

## 4.1 Why should we use CouchDB?

CouchDB [3] has several features that make it powerful such as:

- Easy cross server replication through instances
- Its internal architecture is fault-tolerant, and failures occur in a controlled environment and are dealt with gracefully
- Single problems do not cascade through an entire server system but stay isolated in single requests
- CouchDB is designed to handle varying traffic. For instance, if a website is experiencing a sudden spike in traffic, CouchDB will generally absorb a lot of concurrent requests without falling over. It may take a little more time for each request, but they all get answered.
- Quick indexing and search and retrieval.
- RESTful Web interface.
- Documents are accessible through browsers and indices can be queried through HTTP.
- Index, combine, and transform operations performed with JavaScript.
- Simple create, read, update, delete (CRUD) operations.
- Uses advance MapReduce.
- No Locking:
  Regular RDBS like MySQL and as well NoSQL databases use locking to make sure that a table or row is not modified in the same time by another client. Instead CouchDB uses a concept called MVCC (Multi Version Concurrency Control). In the below figure 4, you can see the difference between locking and MVCC, basically reads do not wait for a table to be unlocked when the table is edited. This means that under heavy load the system will use the resources better.
  Eventual consistency [4] offers the ability to provide partition tolerance and availability.

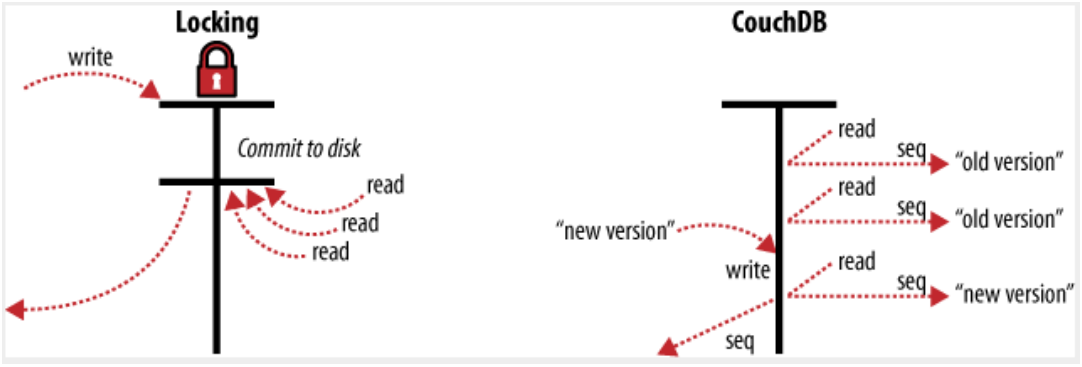



*Figure 4: MVCC (Image source: [4])*

Besides making databases and documents available to a wide audience of users, CouchDB also facilitates Web app development and makes it possible to serve apps directly from the database.

### 4.2 CouchDB vs MongoDB [5]

| Property | CouchDB | MongoDB |
|---|---|---|
| *Speed* | Slower in comparison | Faster in comparison |
| *Mobile Support* | Can run on iOS and Android devices | No mobile support provided |
| *Replication* | Master-master and master-slave replication | Master-slave replication |
| *Size* | CouchDB can grow with size of database | MongoDB better suited for rapidly growing database |
| *Syntax* | Queries use MapReduce functions | Query syntax closer to SQL |

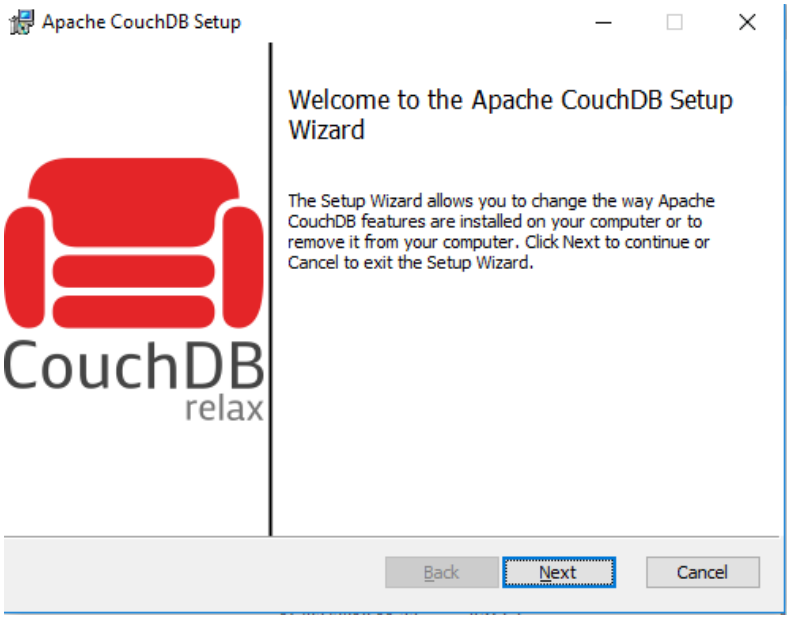
### 4.3 CouchDB vs CouchBase [6]

CouchBase is very similar to CouchDB

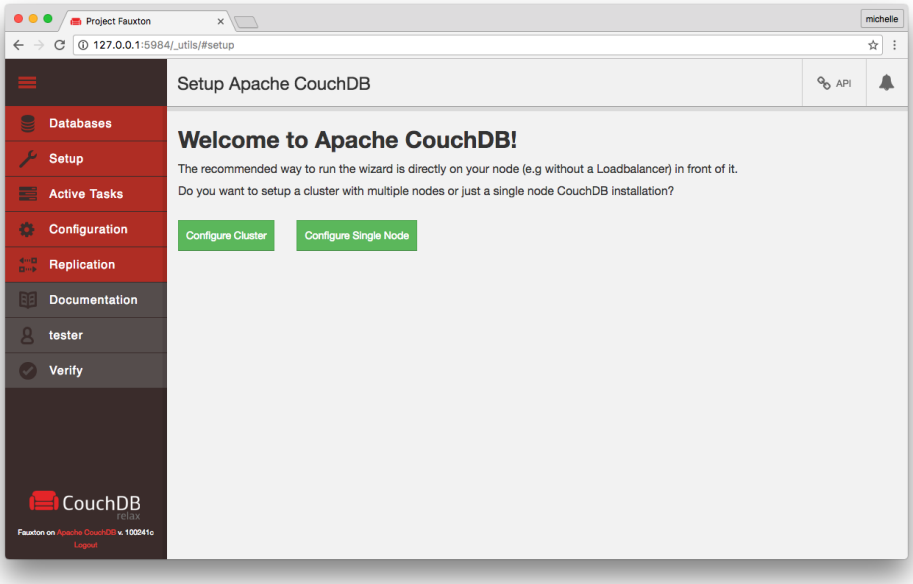
| Property | CouchDB | CouchBase |
|---|---|---|
| *Data models* | Document | Document, Key-Value |
| *Storage* | Append-only B-Tree | Append-only B-Tree |
| *Consistency* | Eventual | Strong |
| *Topology* | Replicated | Distributed |
| *Replication* | Master-Master | Master-Master |
| *Automatic failover* | No | Yes |
| *Integrated cache* | No | Yes |
| *Memcached compatible* | No | Yes |
| *Locking* | Optimistic with MVCC | Optimistic & Pessimistic |
| *MapReduce (Views)* | Yes | Yes |
| *Query language* | No | Yes, N1QL (SQL for JSON) |
| *Secondary indexes* | Yes | Yes |

## 4.4 Installation

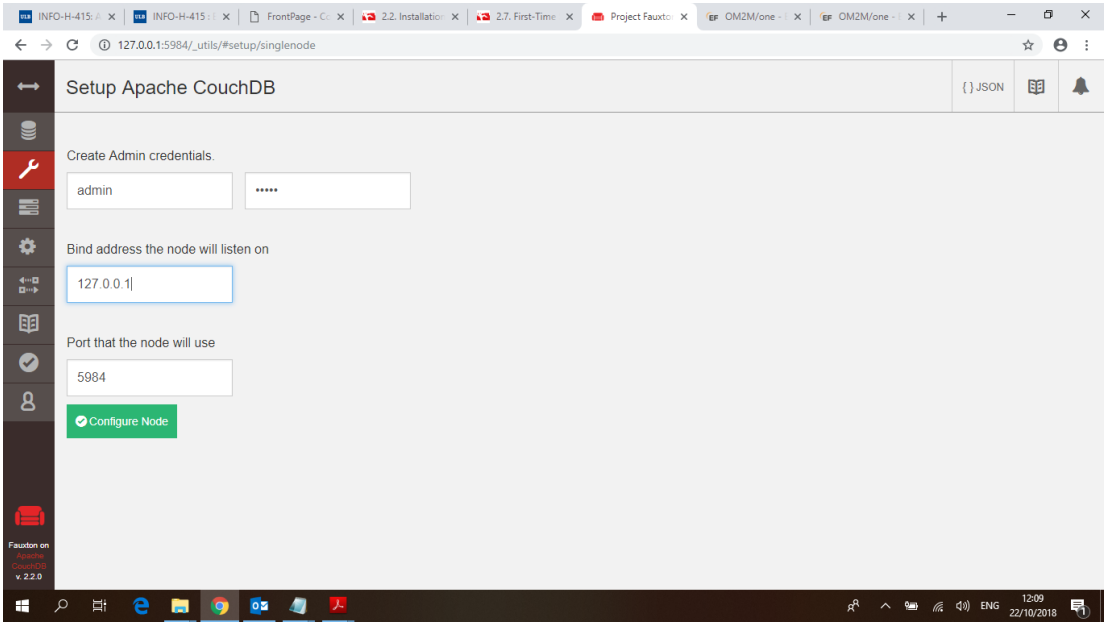
1. Download the Windows Installer from the CouchDB website, http://couchdb.apache.org/#download



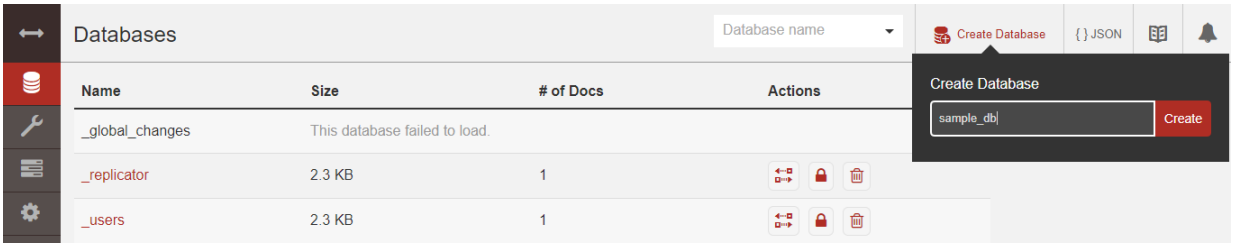

2. Once the installation [7] is complete, open CouchDB using Fauxton, http://127.0.0.1:5984/_utils/
3. Fauxton [8]: Fauxton is a native web-based interface built into CouchDB. It provides a basic interface to the majority of the functionality, including the ability to create, update, delete and view documents and design documents. It provides access to the configuration parameters, and an interface for initiating replication.
4. Fauxton Setup: We have setup CouchDB as a single node



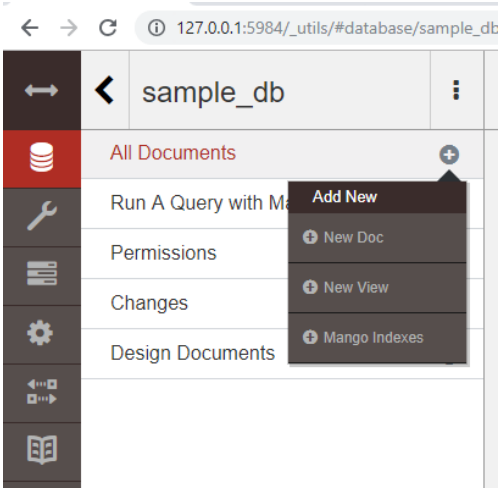
5. Setup credentials for login:

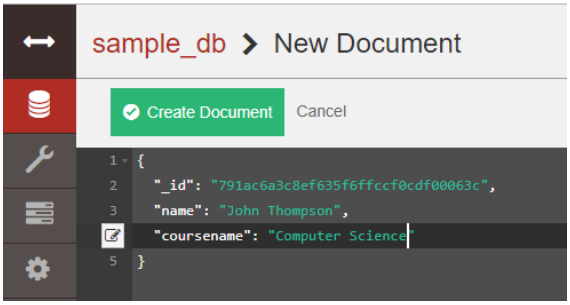



6. Create a new database: sample_db




7. Create a new document



8. We can query the database using the Mango Query API in Fauxton


9. But the applications can query the database using HTTP requests [9]

```
POST /movies/_find HTTP/1.1
Accept: application/json
Content-Type: application/json
Content-Length: 168
Host: localhost:5984

{
    "selector": {
        "year": {"$gt": 2010}
    },
    "fields": ["_id", "_rev", "year", "title"],
    "sort": [{"year": "asc"}],
    "limit": 2,
    "skip": 0,
    "execution_stats": true
}
```

*Figure 5: HTTP POST request*

```
HTTP/1.1 200 OK
Cache-Control: must-revalidate
Content-Type: application/json
Date: Thu, 01 Sep 2016 15:41:53 GMT
Server: CouchDB (Erlang OTP)
Transfer-Encoding: chunked

{
    "docs": [
        {
            "_id": "176694",
            "_rev": "1-54f8e950cc338d2385d9b0cda2fd918e",
            "year": 2011,
            "title": "The Tragedy of Man"
        },
        {
            "_id": "780504",
            "_rev": "1-5f14bab1a1e9ac3ebdf85905f47fb084",
            "year": 2011,
            "title": "Drive"
        }
    ],
    "execution_stats": {
        "total_keys_examined": 0,
        "total_docs_examined": 200,
        "total_quorum_docs_examined": 0,
        "results_returned": 2,
        "execution_time_ms": 5.52
    }
}
```

*Figure 6: HTTP Response*

## 4.5 Indexing in CouchDB

Views [10] in CouchDB are similar to indexes in SQL. Views in CouchDB can be used for filtering documents, retrieving data in a specific order, and creating efficient indexes so you can find documents using values within them. Once you have indexes, they can represent relationships between the documents. These view results are stored in a B-tree index structure.

### 4.5.1 MapReduce

CouchDB uses MapReduce, a two-step process that looks at all the documents and creates a map result consisting of an ordered list of key/value pairs. The mapping occurs once after a document is created. After that, it is not changed unless the document is updated.

The map reduce [10] functions are written in Javascript. Map functions take a document as argument and emit key/value pairs. CouchDB stores the emitted rows by constructing a sorted B-tree index, so the row lookups by key require a small memory and processing footprint, while writes avoid seeks. Generating a view takes O(N), where N is the total number of rows in the view. However, querying a view is very quick, as the B-tree remains shallow even when it contains many keys.

Reduce functions operate on the sorted rows emitted by map view functions. The reduce function is run on every node in the tree in order to calculate the final reduce value. At the end, the reduce function can be incrementally updated upon changes to the map function. The initial reduction is calculated once per each node (inner and leaf) in the tree.

When the reduce function is run on inner nodes, the rereduce flag is true. This allows the function to account for the fact that it will be receiving its own prior output. When rereduce is true, the values passed to the function are intermediate reduction values as cached from previous calculations. When the tree is more than two levels deep, the rereduce phase is repeated, consuming chunks of the previous level's output until the final reduce value is calculated at the root node.

## 4.6 Replication

CouchDB supports both master-master and master-slave replication. This allows low latency access to data regardless of location. Replication in CouchDB is as simple as sending HTTP requests to the database with a source and target.

CouchDB will start sending any changes that occur in the source to the target database. This is a unidirectional process. If you want a bidirectional process, you will need to trigger the replication on the destination server with it being the source and the remote server being the destination.

# 5. Application

## 5.1 Topic

In this project, we consider the dataset [11] from Zomato, which included a list of restaurants located in different cities of different countries and their rating based on the reviews given by the customers. We have a total of 9551 records for this data. Consider the Zomato.csv file, we have the following fields:

- Restaurant ID
- Restaurant Name
- Country Code
- City
- Address
- Locality
- Locality Verbose
- Longitude
- Latitude
- Cuisines
- Average Cost for two
- Currency
- Has Table
- Has Online
- Is delivering now
- Switch to Order Menu
- Price range
- Aggregate rating
- Rating color
- Rating text
- Votes

Using this data, we compare the structure of data storage in both CouchDB and MySQL. Also, we will compare the query performance between the two.

## 5.2 Schema for Relational Database

The dataset is already in the form of documents store, that is, each record corresponds to one document. But we had to create a schema corresponding to this dataset for storing this data in MySQL.

### 5.2.1 ER-Diagram

The schema created is as follows:

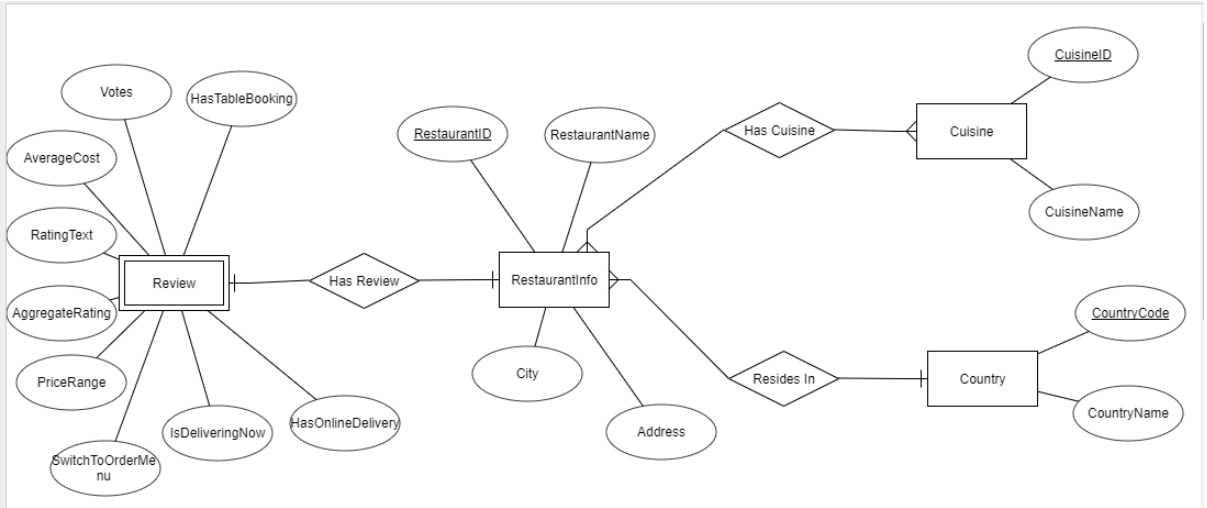



*Figure 7: ER Diagram for Zomato Dataset*

### 5.2.2 Relation Schema Diagram

The corresponding relational schema for the above ER diagram is as follows:

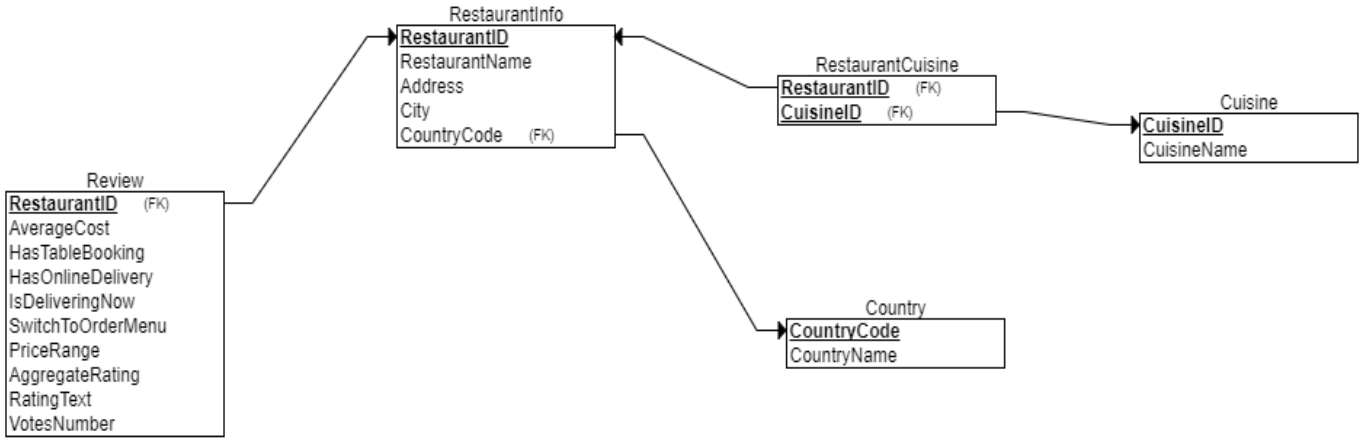



*Figure 8: Relational Schema for Zomato*

## 5.3 Structure for Document Store:

Each record for a restaurant review is stored in a single document. Each document has a unique _id as an identifier. In the below figure, we can see how a document looks. Here _id and restaurantID can be the same, but we have kept it different as more reviews can be added for the same restaurantID in future.

```
{
 "id": "41d66ce53d805cdd1344c7de54007770",
 "key": "41d66ce53d805cdd1344c7de54007770",
 "value": {
  "rev": "1-c4b9642e0e7d8c6cbc5505db2c302f23"
 },
 "doc": {
  "_id": "41d66ce53d805cdd1344c7de54007770",
  "_rev": "1-c4b9642e0e7d8c6cbc5505db2c302f23",
  "Restaurant ID": 6317637,
  "Restaurant Name": "Le Petit Souffle",
  "Country Code": 162,
  "City": "Makati City",
  "Address": "Third Floor, Century City Mall, Kalayaan Avenue, Poblacion, Makati City",
  "Locality": "Century City Mall, Poblacion, Makati City",
  "Locality Verbose": "Century City Mall, Poblacion, Makati City, Makati City",
  "Longitude": 121.027535,
  "Latitude": 14.565443,
  "Cuisines": "French, Japanese, Desserts",
  "Average Cost for two": 1100,
  "Currency": "Botswana Pula(P)",
  "Has Table booking": "Yes",
  "Has Online delivery": "No",
  "Is delivering now": "No",
  "Switch to order menu": "No",
  "Price range": 3,
  "Aggregate rating": 4.8,
  "Rating color": "Dark Green",
  "Rating text": "Excellent",
  "Votes": 314
```

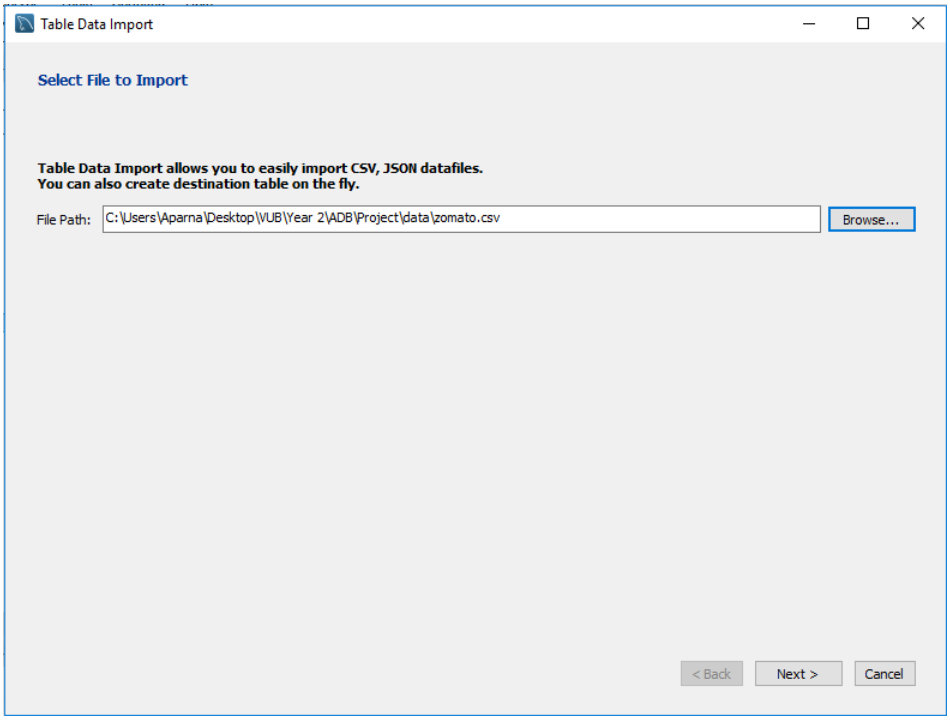

*Figure 9: Document Structure*

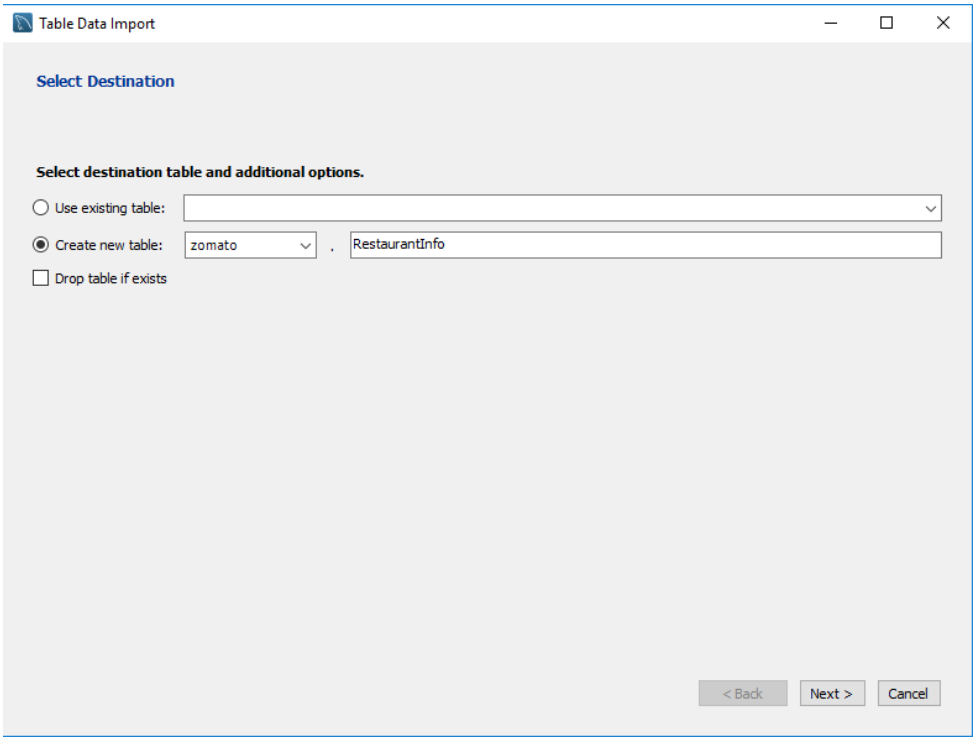
## 5.4 Importing Dataset

As the dataset is in a CSV form, we import this dataset into both MySQL and CouchDB.

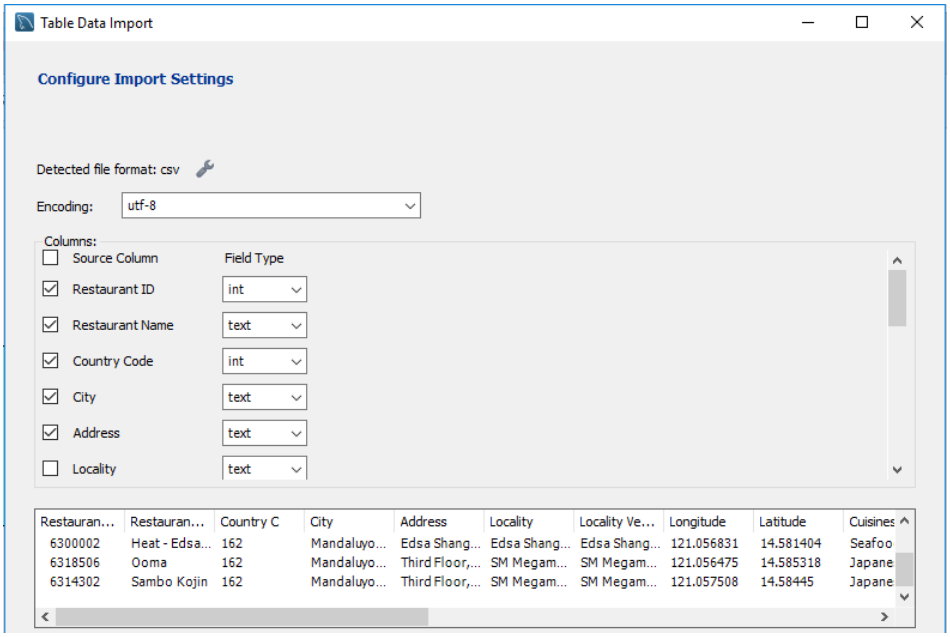
### 5.4.1 Importing Dataset into MySQL

Importing data in MySQL is simple. Just specify the csv file from which you want to import.




Specify the columns we want to keep in our RestaurantInfo Table using the above schema as below screenshot. Repeat the below procedure, to create five tables as in the relational schema.




### 5.4.2 Importing Dataset into CouchDB

To import the dataset into CouchDB, we use a python code which reads a json file as input and for each json object and creates a corresponding json document in CouchDB. We create a database called sample_db, where we will import all the data into.

Pre-requisites to run the code:

1. Python and pip already installed
2. Install couchdb package using pip command: pip install couchdb
3. Convert the csv file into a single json file using online tools

In the figure below, the python code [12] to bulk import the dataset:

```python
import sys
import couchdb
import json
import re

DB_STRING="http://127.0.0.1:5984"
regex = r"\{(.*?)\}"

def main(json_file, db_name, couchdb_address):

# 1. Create the database.. No error checking here because we want to get exception if db exists
    couch = couchdb.Server()
    db = couch[db_name]
    json_str = []

# 2 Read the json line by line and put into the db
    with open(json_file, encoding="Latin-1") as jsonfile:
        for row in jsonfile:
            for ch in row:
                if( "}" not in row):
                    json_str.append(row)
                    continue
                json_str.append("}")
                print(json_str)
                strl = "".join(json_str)
                db_entry = json.loads(strl)
                db.save(db_entry)
                json_str = []
```

*Figure 10: Code snippet to import json file in couchdb*

```
C:\Users\Aparna\Desktop\VUB\Year 2\ADB\Project>python json_to_couchDB.py csvjson.json sample_db
Call as <jsondb.json> <new_db> <optional db string>

C:\Users\Aparna\Desktop\VUB\Year 2\ADB\Project>
```

*Figure 11: Execute python code*

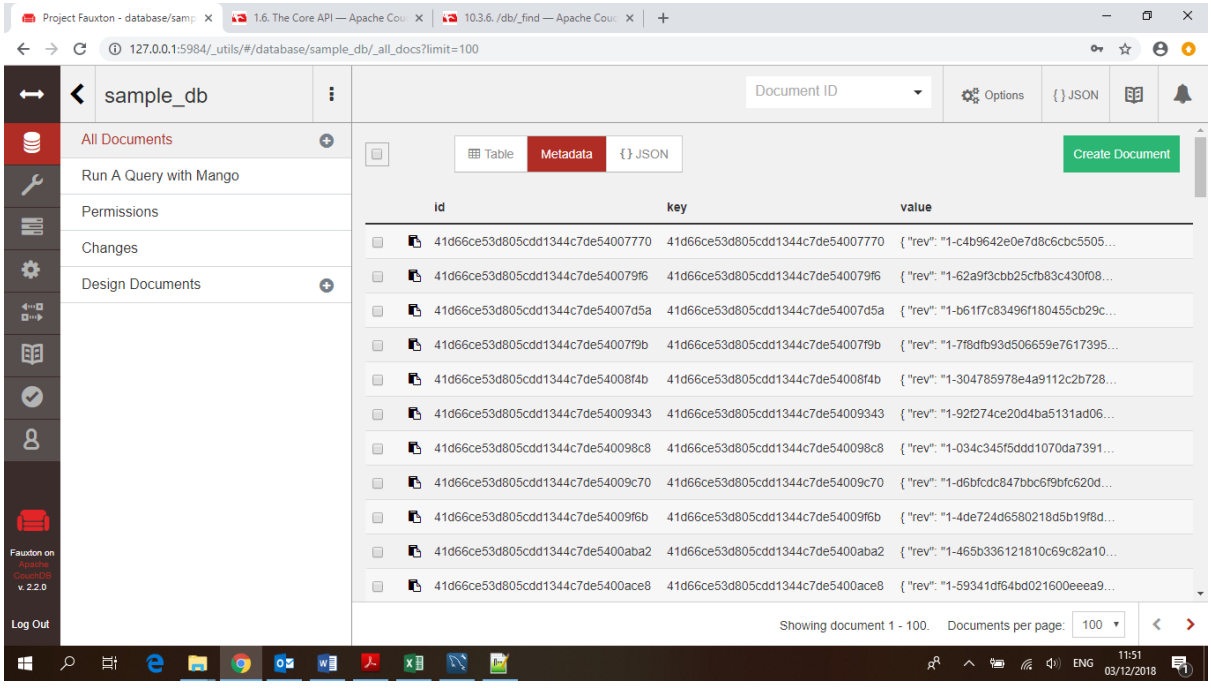


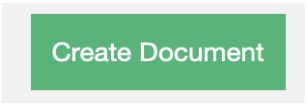
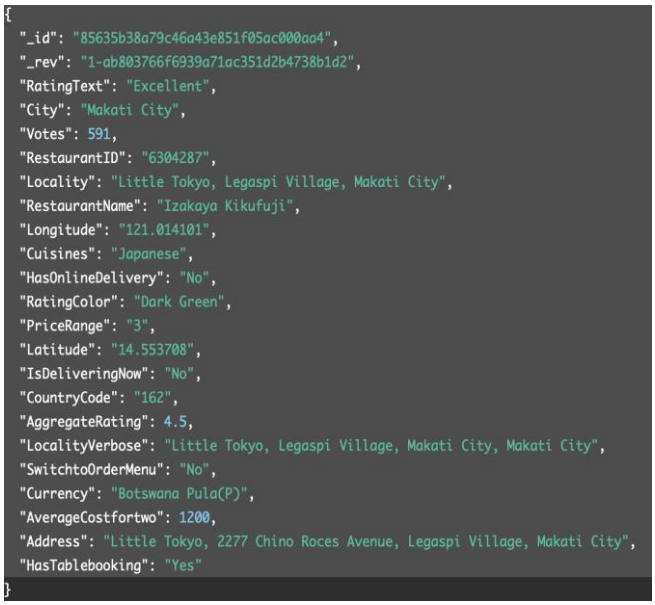

*Figure 12: Imported Data view in CouchDB*

## 5.5 Testing and Comparison

We are using Fauxton to perform the operations on the CouchDB as it provides a good UI to view queries. It can be used to perform basic and simple queries. But this can also be done using cURL commands for HTTP requests.

### 5.5.1 CRUD Operations:

1. Insert a new record for a new restaurant with all details.

<table>
<tr><th>CouchDB relax</th><th>MySQL</th></tr>
<tr>
<td>In CouchDB, inserting a new document is easy.<br>Firstly, we just go into a certain database where we want our new document located.<br>And click the “Create Document” button.<br><br>  <br><br>Then, we can just write any new key-value pairs as a json document. So, no query is required for this.<br>In case of bulk insert, we can use the python code above or HTTP requests for the same.<br><br>  </td>
<td>In MySQL, however, inserting a new data will be more difficult considering the relational schema it has, where the foreign key constraint could also be an obstacle.<br>For example, take the Restaurant Review schema we have. If we want to insert a review of a new restaurant, we firstly should insert the basic information into the RestaurantInfo table since the RestaurantID is the foreign key for others. Then we can insert the review information in the Review table.</td>
</tr>
</table>

2. Delete/ Update record

|  |  |
|---|---|
| In CouchDB, deleting or updating a record in database is easy, if we know the document that we want to update. To delete or update a record we already have, we first just need to find the record we want to change, then update it or delete it directly.<br><br>In case of bulk delete or update, we have to just write python code like above figure 11 or use HTTP requests with the doc_ids and rev_ids for the same. | Again, due to the relation schema of MySQL, the operation of deleting and updating is more complicated than CouchDB. To delete one record of one restaurant, for example, we need to delete all information about it in several tables according to the key value. But, if we want to add a new cuisine type which is not included in cuisine table to a restaurant, we firstly need to add the cuisine relation in certain table due to the foreign key constraint. It takes two steps to update the record in this situation. |

### 5.5.2 Selector Queries

1. Find all the restaurants which have Chinese cuisine.

CouchDB

```
{
   "selector": {
      "Cuisines": "Chinese"
   },
   "fields": [
      "RestaurantName",
      "City",
      "Cuisines",
      "AggregateRating"
   ]
}
```

| AggregateRat. | City | Cuisines | RestaurantNa. |
|---|---|---|---|
| 4.4 | Mandaluyong City | Chinese | Din Tai Fung |
| 3.8 | Albany | Chinese | House of China Restaura... |
| 4.2 | Cedar Rapids/Iowa City | Chinese | Ting's Red Lantern |
| 4 | Columbus | Chinese | Chef Lee's Peking Restau... |
| 4 | Davenport | Chinese | China Cafe |
| 4.1 | Des Moines | Chinese | Tsing Tsao South |
| 3.5 | Pocatello | Chinese | Chang Garden |
| 3.8 | Sioux City | Chinese | Hunan Palace |
| 3.8 | Waterloo | Chinese | Hong Kong Chinese Rest... |
| 3.7 | Waterloo | Chinese | Golden China |
| 4.2 | Abu Dhabi | Chinese | P.F. Chang's |
| 4 | Bhopal | Chinese | Chi Kitchen |
| 4.1 | Bhubaneshwar | Chinese | Silver Streak |
| 4 | Bhubaneshwar | Chinese | Taste Of China |

Showing 4 columns. Show all columns. Showing document 1 - 20. Documents per page: 20

MySQL

```
SELECT I.RestaurantName, I.City, C.CuisineName, R.AggRating
FROM Review R, RestaurantInfo I, RestaurantCuisine RC, Cuisine C
WHERE I.RestaurantID = R.RestaurantID AND
RC.CuisineID = C.CuisineID AND
RC.RestaurantID = I.RestaurantID AND
C.CuisineName = 'Chinese'
```

| RestaurantName | City | CuisineName | AggRating |
|---|---|---|---|
| The Forresta Kitchen & Bar | Jaipur | Chinese | 4 |
| Replay | Jaipur | Chinese | 4.3 |
| Mainland China | New Delhi | Chinese | 3.7 |
| Mamu's Infusion | Jaipur | Chinese | 4 |
| Monarch Restaurant - Holiday Inn Jaipur City Centre | Jaipur | Chinese | 4.5 |
| Chao Chinese Bistro - Holiday Inn Jaipur City Centre | Jaipur | Chinese | 4.3 |
| Apni Rasoi | Gurgaon | Chinese | 3 |
| Calzone- Dine & Rooftop Lounge | Jaipur | Chinese | 3.7 |
| Mini's Royal Cafe | Gurgaon | Chinese | 3.3 |
| Moti Restaurant | New Delhi | Chinese | 2.4 |
| The Delhi Heights | New Delhi | Chinese | 2.8 |
| Woks - The Lalit New Delhi | New Delhi | Chinese | 3.6 |
| Hot Pot | New Delhi | Chinese | 3.1 |
| Embassy | New Delhi | Chinese | 3 |
| Flaming Chilli Pepper | New Delhi | Chinese | 3.8 |
| Haldiram's | Gurgaon | Chinese | 3.6 |
| Kabir Restaurant | Ahmedabad | Chinese | 3.8 |
| Patang - The Revolving Restaurant | Ahmedabad | Chinese | 3.7 |
| The Golden Dragon | New Delhi | Chinese | 3.5 |
| Yanki Sizzlers | Ahmedabad | Chinese | 4.1 |
| 650 - The Global Kitchen | Ahmedabad | Chinese | 4.2 |
| Lotus Pond | New Delhi | Chinese | 4.2 |
| Castle 9 | New Delhi | Chinese | 3.1 |
| The Plaza Solitaire | Gurgaon | Chinese | 3.3 |
| Ridhi Sidhi | Gurgaon | Chinese | 3 |
| Gola Northend | New Delhi | Chinese | 3.1 |
| Shagun | New Delhi | Chinese | 3.9 |

| | | 1 | 2 | 3 | Average |
|---|---|---|---|---|---|
| **CouchDB** | Execution Time (ms) | 158 | 153 | 151 | 154 |
| **MySQL** | Execution Time (ms) | 41 | 11 | 13 | 21.67 |

2. Find all the restaurants whose aggregate rating is greater than 4 and country code is 1.

CouchDB

```
{
   "selector": {
      "$and": [
         {
            "AggregateRating": {
               "$gt": "4"
            }
         },
         {
            "CountryCode": {
               "$eq": "1"
            }
         }
      ]
   },
   "fields": [
      "RestaurantName",
      "City",
      "Cuisines",
      "AggregateRating"
   ]
}
```

| AggregateRat. | City | Cuisines | RestaurantNa. |
|---|---|---|---|
| 4.2 | Agra | North Indian, Chinese, M... | Pinch Of Spice |
| 4.3 | Agra | North Indian, Mughlai | Peshawri - ITC Mughal |
| 4.1 | Agra | South Indian, Desserts | Dasaprakash Restaurant |
| 4.9 | Agra | Cafe, North Indian, Chinese | Sheroes Hangout |
| 4.4 | Agra | Italian, Pizza | Pizza Hut |
| 4.2 | Agra | Chinese, Italian, Continen... | Tea'se Me - Rooftop Tea ... |
| 4.1 | Agra | North Indian, Fast Food | Thaaliwala |
| 4.2 | Ahmedabad | Chinese, Italian, North Ind... | 650 - The Global Kitchen |
| 4.5 | Ahmedabad | Ice Cream, Desserts, Con... | Huber & Holly |
| 4.1 | Ahmedabad | North Indian, Continental,... | @Mango |
| 4.3 | Ahmedabad | Pizza, Italian, Beverages, ... | Fozzie's Pizzaiolo |
| 4.4 | Ahmedabad | Pizza, Italian | La Pino'z Pizza |
| 4.4 | Ahmedabad | Cafe, Continental, Desserts | Mocha |
| 4.6 | Ahmedabad | Desserts, Ice Cream | Cryo Lab |

Showing 4 columns. Show all columns. Showing document 1 - 20. Documents per page: 20

MySQL

```
SELECT RestaurantInfo.RestaurantName, Review.AggRating, Cuisine.CuisineName, RestaurantInfo.City
FROM Cuisine, RestaurantCuisine, RestaurantInfo, Review
WHERE
RestaurantInfo.RestaurantID = Review.RestaurantID AND
RestaurantCuisine.RestaurantID = Review.RestaurantID AND
Cuisine.CuisineID = RestaurantCuisine.CuisineID AND
Review.AggRating >= 4 AND
RestaurantInfo.CountryCode = '1'
```

| RestaurantName | AggRating | CuisineName | City |
|---|---|---|---|
| Chokhi Dhani | 4.3 | Rajasthani | Jaipur |
| The Forresta Kitchen & Bar | 4 | North Indian | Jaipur |
| The Forresta Kitchen & Bar | 4 | Desserts | Jaipur |
| The Forresta Kitchen & Bar | 4 | Chinese | Jaipur |
| The Forresta Kitchen & Bar | 4 | Rajasthani | Jaipur |
| The Forresta Kitchen & Bar | 4 | Beverages | Jaipur |
| The Forresta Kitchen & Bar | 4 | Mexican | Jaipur |
| The Forresta Kitchen & Bar | 4 | Continental | Jaipur |
| Replay | 4.3 | North Indian | Jaipur |
| Replay | 4.3 | Chinese | Jaipur |
| Replay | 4.3 | Mexican | Jaipur |
| Replay | 4.3 | Italian | Jaipur |
| Replay | 4.3 | Continental | Jaipur |
| Tapri Central | 4.7 | Fast Food | Jaipur |
| Tapri Central | 4.7 | Street Food | Jaipur |
| Tapri Central | 4.7 | Cafe | Jaipur |
| On The House | 4.4 | Cafe | Jaipur |
| On The House | 4.4 | Mexican | Jaipur |
| On The House | 4.4 | Bakery | Jaipur |
| On The House | 4.4 | Italian | Jaipur |
| On The House | 4.4 | Continental | Jaipur |
| Taruveda Bistro | 4.2 | Cafe | Jaipur |
| Taruveda Bistro | 4.2 | Italian | Jaipur |
| Taruveda Bistro | 4.2 | Continental | Jaipur |
| Taruveda Bistro | 4.2 | Japanese | Jaipur |
| Mamu's Infusion | 4 | North Indian | Jaipur |
| Mamu's Infusion | 4 | Chinese | Jaipur |

| | | 1 | 2 | 3 | Average |
|---|---|---|---|---|---|
| **CouchDB** | Execution Time (ms) | 63 | 91 | 115 | 90 |
| **MySQL** | Execution Time (ms) | 18 | 14 | 11 | 14.33 |

3. List Top 20 restaurants having max votes in United States.

CouchDB

```
{
   "selector": {
      "Votes": {
         "$gt": 0
      },
      "CountryCode": {
         "$eq": "162"
      }
   },
   "fields": [
      "Votes",
      "RestaurantName"
   ],
   "sort": [
      {
         "Votes": "desc"
      }
   ],
   "limit": 20,
   "skip": 0
}
```

| RestaurantNa. | Votes |
|---|---|
| Silantro Fil-Mex | 1070 |
| Vikings | 677 |
| Spiral - Sofitel Philippine Plaza Manila | 621 |
| The Food Hall by Todd English | 618 |
| Izakaya Kikufuji | 591 |
| NIU by Vikings | 535 |
| Locavore | 532 |
| Buffet 101 | 520 |
| Mad Mark's Creamery & Good Eats | 488 |
| Guevarra's | 458 |
| Wildflour Cafe + Bakery | 392 |
| Ooma | 365 |
| Din Tai Fung | 336 |
| Le Petit Souffle | 314 |

Showing 2 columns. Show all columns. Documents per page: 20

MySQL

```
SELECT I.RestaurantName, R.VotesNumber
FROM RestaurantInfo I, Review R
WHERE I.RestaurantID = R.RestaurantID AND
I.CountryCode = '162'
ORDER BY R.VotesNumber DESC
LIMIT 20
```

| RestaurantName | VotesNumber |
|---|---|
| Silantro Fil-Mex | 1070 |
| Vikings | 677 |
| Spiral - Sofitel Philippine Plaza Manila | 621 |
| The Food Hall by Todd English | 618 |
| Izakaya Kikufuji | 591 |
| NIU by Vikings | 535 |
| Locavore | 532 |
| Buffet 101 | 520 |
| Mad Mark's Creamery & Good Eats | 488 |
| Guevarra's | 458 |
| Wildflour Cafe + Bakery | 392 |
| Ooma | 365 |
| Din Tai Fung | 336 |
| Le Petit Souffle | 314 |
| Silantro Fil-Mex | 294 |
| Heat - Edsa Shangri-La | 270 |
| Sambo Kojin | 229 |
| Sodam Korean Restaurant | 223 |
| Balay Dako | 211 |
| Hobing Korean Dessert Cafe | 118 |

| | | 1 | 2 | 3 | Average |
|---|---|---|---|---|---|
| **CouchDB** | Execution Time (ms) | 1580 | 1811 | 1917 | 1769.33 |
| **MySQL** | Execution Time (ms) | 28 | 1 | 0 | 9.67 |

4. List all restaurants whose name stat with “L”.



```
{
   "selector": {
      "RestaurantName": {
         "$regex": "^L"
      }
   },
   "fields": [
      "RestaurantName",
      "City",
      "PriceRange",
      "AggregateRating"
   ]
}
```

| AggregateRat. | City | PriceRange | RestaurantNa. |
|---|---|---|---|
| 4.8 | Makati City | 3 | Le Petit Souffle |
| 4.8 | Pasig City | 3 | Locavore |
| 4.2 | Rio de Janeiro | 2 | Leme Light |
| 4.6 | Sí£o Paulo | 4 | Les 3 Brasseurs |
| 3.5 | Albany | 2 | Locos Grill & Pub |
| 3.5 | Albany | 2 | Longhorn Steakhouse |
| 4.5 | Athens | 3 | Last Resort Grill |
| 4.1 | Athens | 3 | La Dolce Vita Ristor... |
| 4.4 | Boise | 1 | Los Beto's |
| 4.4 | Boise | 2 | Lucianos Italian Re... |
| 3.8 | Dalton | 1 | Las Palmas |
| 4.1 | Dalton | 1 | Los Pablos |
| 0 | Davenport | 2 | Los Agaves |
| 4.1 | Davenport | 2 | Los Agaves |

Showing 4 columns. Show all columns. Documents per page: 20

```
SELECT I.RestaurantName, I.City, R.AverCost, R.AggRating
FROM RestaurantInfo I, Review R
WHERE I.RestaurantID = R.RestaurantID AND
I.RestaurantName LIKE "L%"
```

| RestaurantName | City | AverCost | AggRating |
|---|---|---|---|
| Lotus Pond | New Delhi | 1800 | 4.2 |
| Lodi - The Garden Restaurant | New Delhi | 2600 | 4.2 |
| LPK Waterfront | Goa | 1500 | 3.7 |
| Le Plaisir | Pune | 1000 | 4.8 |
| Le Chef Restro Bar | Faridabad | 1000 | 2.7 |
| Lazeez Food | Gurgaon | 800 | 3.5 |
| Little Italy | Nashik | 800 | 3 |
| Lanterns Kitchen & Bar | New Delhi | 1200 | 3.8 |
| La Plage | Goa | 800 | 4.6 |
| La Trattoria of Lavandula | Hepburn Springs | 7 | 3.8 |
| Lake House Restaurant | Vineland Station | 70 | 4.3 |
| Leonard's Bakery | Rest of Hawaii | 10 | 4.7 |
| Lulu's Waikiki | Rest of Hawaii | 25 | 3.9 |
| Locos Grill & Pub | Albany | 25 | 3.5 |
| Longhorn Steakhouse | Albany | 25 | 3.5 |
| La Dolce Vita Ristorante | Athens | 40 | 4.1 |
| Last Resort Grill | Athens | 40 | 4.5 |
| Los Beto's | Boise | 10 | 4.4 |
| Lucianos Italian Restaurant | Boise | 25 | 4.4 |
| Los Pablos | Dalton | 10 | 4.1 |
| Las Palmas | Dalton | 10 | 3.8 |
| Los Agaves | Davenport | 25 | 4.1 |
| Los Aztecas | Dubuque | 10 | 3.5 |
| L. May Eatery | Dubuque | 40 | 3.8 |
| Longstreet Cafe | Gainesville | 10 | 4.3 |
| Leopold's Ice Cream | Savannah | 10 | 4.6 |
| Lulu's Chocolate Bar | Savannah | 10 | 4.3 |

| | | 1 | 2 | 3 | Average |
|---|---|---|---|---|---|
| **CouchDB** | Execution Time (ms) | 61 | 75 | 111 | 82.33 |
| **MySQL** | Execution Time (ms) | 61 | 13 | 9 | 27.67 |

5. Show all information of the restaurants in Canada.

CouchDB

```
{
    "selector": {
        "CountryCode": "37"
    }
}
```

| Address | AggregateRat... | AverageCostf... | City | CountryCode |
|---|---|---|---|---|
| 150 Richmond... | 3.7 | 25 | Chatham-Kent | 37 |
| 4931 50th Stre... | 3 | 25 | Consort | 37 |
| 3100 N Servic... | 4.3 | 70 | Vineland Station | 37 |
| 14 Second Av... | 3.3 | 25 | Yorkton | 37 |

Showing 5 of 23 columns. Show all columns. Documents per page: 20

MySQL

```
SELECT R.*, I.Address, I.City, I.CountryCode,
I.RestaurantName, RC.CuisineID, Cu.CuisineName, C.CountryName
FROM RestaurantInfo I, Review R, RestaurantCuisine RC, Cuisine Cu, Country C
WHERE I.RestaurantID = R.RestaurantID AND
I.RestaurantID = RC.RestaurantID AND
RC.CuisineID = Cu.CuisineID AND
I.CountryCode = C.CountryCode AND
C.CountryName = 'Canada'
```

| RestaurantID | AverCost | HasTableBooking | HasOnlineDelivery | IsDeliveringNow | SwitchToOrderMenu | PriceRange | AggRating | Rati |
|---|---|---|---|---|---|---|---|---|
| 16643459 | 25 | No | No | No | No | 2 | 3 | Aver |
| 16643459 | 25 | No | No | No | No | 2 | 3 | Aver |
| 16654702 | 70 | No | No | No | No | 4 | 4.3 | Very |
| 16654702 | 70 | No | No | No | No | 4 | 4.3 | Very |
| 16654702 | 70 | No | No | No | No | 4 | 4.3 | Very |
| 16659169 | 25 | No | No | No | No | 2 | 3.7 | Goo |
| 16659169 | 25 | No | No | No | No | 2 | 3.7 | Goo |
| 16668008 | 25 | No | No | No | No | 2 | 3.3 | Aver |

| | | 1 | 2 | 3 | Average |
|---|---|---|---|---|---|
| **CouchDB** | Execution Time (ms) | 900 | 1679 | 1602 | 1393.67 |
| **MySQL** | Execution Time (ms) | 1 | 0 | 1 | 0.67 |

6. Calculate the average rating for restaurants having Indian cuisine.



In CouchDB, it is not possible to conduct selector function with query in Mongo like listed above. To fulfil the request, we need to use map and reduce function with JavaScript. First, we create a view of the data we want to calculate.

```
function (doc) {
  if(doc.Cuisines.indexOf("Indian")!= -1){
    emit("Indian",doc.AggregateRating);
  }
}
```

Then, we run a reduce function to calculate the result.

```
function (keys, values) {
  return sum(values)/values.length;
}
```

We can then get both view of data and result we want but without any time being consumed.
This map-reduce functionality will be an advantage when our database is in a distributed structure.

```
SELECT AVG(R.AggRating)
FROM RestaurantInfo I, Review R, Cuisine C, RestaurantCuisine RC
WHERE I.RestaurantID = R.RestaurantID AND
I.RestaurantID = RC.RestaurantID AND
RC.CuisineID = C.CuisineID AND
C.CuisineName LIKE '%Indian%'
```

| AVG(R.AggRating) |
|---|
| 2.5423673291111553 |

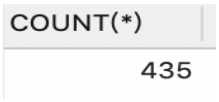
7. Count the number of restaurants which have table booking and online delivery.

<table>
<tr><td>Like mentioned above, we need map and reduce function to fulfill the set function. First, we create a view to select the restaurants matching our standard.<br><br>Then, we use the default COUNT function to count the number.<br><br>

```
function (doc) {
  if(doc.HasTablebooking == "Yes" && doc.HasOnlineDelivery == "Yes"){
    emit("Eligible Restaurant", null);
  }
}
```

</td><td>

```
SELECT COUNT(*)
FROM Review R
WHERE R.HasOnlineDelivery = 'Yes' AND
R.HasTableBooking = 'Yes'
```



</td></tr>
</table>

# 6. Results and Analysis

We did the operation in a laptop (Apple MacBook Pro 2017) with the features as:

Processor: 2.9 GHz Intel Core i5

RAM: 8GB 2133 MHz LPDDR3

We started from finding an appropriate dataset for our document store. After which we created the relational schema for MySQL. To insert the records into different tables from a single csv file, we split the original data into several tables according to our schema. We then imported the data into MySQL and created primary and foreign key constraint for MySQL. Importing the data into CouchDB was faster once we wrote our code for bulk importing the data.

The next step was to create queries to search data in two different type of database in order to compare them.

Based on the process and results listed above, our conclusion is that:

CRUD operations: 1 and 2, are inserting, updating and deleting, these operations can be conducted directly in graphical interface. Updating and deleting the documents can be done with this UI when we know the document that we want to update or delete. In case we want to bulk update or delete we can do the same with HTTP POST requests. But, of course, an additional knowledge of python, in our case, is required to write this program. To do the same in MySQL, there are various foreign key constraints to be considered. Thus, to bulk insert, update or delete, it is very important to have the complete knowledge of the relational schema. On the contrary, CouchDB does not require any pre-defined schema which provides more flexibility in performing CRUD operations. In other words, every data in CouchDB can have different schema, making it possible to have various details for different types of records.

Selector Query: 1 to 5, which are queries using certain criteria to find the results, show that MySQL performs better than CouchDB in searching. We tried different queries including many methods, like sorting, and joining in MySQL. According to the knowledge we had for CouchDB and MySQL, we thought searching a single document might be an advantage for CouchDB compared to the joining in MySQL where multiple tables need to be joined. But contrary to what we expected, MySQL performed better and ran faster in both queries we tested. Although we were testing on 10K records for both the databases, this size is not enough to illustrate the actual power of document stores. Maybe if we had billions of records, only then would the difference be visible. Also, when we require these records multiple times, it will be faster with CouchDB with the help of views.

Aggregate Query: 6 and 7 perform aggregate functionality in the two databases. In MySQL, we can conduct them as easy as queries like selecting and sorting. Whereas, Mango, the query language interface for CouchDB in Fauxton, does not directly support such a function. To

achieve this in CouchDB, we need to use map and reduce function with JavaScript to calculate data. In the map function, we first create a view to filter the data we need for calculation. Then, we use a reduce function, either default one or customized one, to do the aggregate calculation based on the data provided by the map function. In simple words, a view can contain both the result set and the total number of results. The former is generated by the map function and the latter by the reduce function. The results of both are stored and indexed. The contents of the view can be retrieved in batches using a *limit* constraint but it will also include the total count, which does not have to be computed on each request (or even on the first request), like in MySQL.

Thus, we do require to take efforts to gain knowledge of Map-Reduce functionality and JavaScript to perform this operation in CouchDB. Since we have been working on relational databases from a long time, it does seem easier to perform aggregate functions in MySQL. But as mentioned earlier, in real world applications on the internet, data is usually very large and distributed. Thus, in such a scenario, MySQL will not be helpful, as it would take extremely long time to retrieve results, which will also require a lot of computational resources to just retrieve some basic information. Also, the use of Map-reduce functionality is an advantage when our data is distributed over a network, as this will give us faster results in that case.

# 7. Conclusion

Thus, in this project, we compare the use and applications of document store like CouchDB to relational database like MySQL in terms of performance and complexity of queries. This project builds a greater horizon of knowledge for us as students who usually study relational databases in academics. This project helped us to understand the need and use of NoSQL databases.

Thus, a NoSQL document store will be suitable when our schema is subjected to frequent changes or we frequently need data in a computed or aggregated form. Hence it is very important to choose a right database as per our application.

The applications most suitable for implementing document stores are:

- Content management systems
- eCommerce applications
- Blogging platforms
- User generated content like tweets, comments, reviews, rating etc.
- Data from IoT applications, like various sensor data from IoT devices

As in the future we are leading towards more of user generated content, these databases will be form a very important part of application development.

# 8. References


1 NoSQL, AWS, Available:https://aws.amazon.com/nosql/

2 Document, AWS, Available:https://aws.amazon.com/nosql/document/

3 CouchDB, Apache, Available:http://docs.couchdb.org/en/stable/intro/why.html

4 CouchDB, Consistency, Apache, Available: http://docs.couchdb.org/en/stable/intro/consistency.html

5 CouchDB vs MongoDB, Available:https://blog.panoply.io/couchdb-vs-mongodb

6 CouchBase vs CouchDB, Available:https://www.couchbase.com/couchbase-vs-couchdb

7 Installation on Windows, CouchDB, Available:http://docs.couchdb.org/en/2.2.0/install/windows.html

8 Fauxton Visual Guide, Apache, Available:http://couchdb.apache.org/fauxton-visual-guide/#using-fauxton

9 Find, Apache, Available:http://docs.couchdb.org/en/stable/api/database/find.html#selector-syntax

10 Views, Available:http://docs.couchdb.org/en/2.2.0/ddocs/views/intro.html

11 Zomato Restaurants data, kaggle, Available:https://www.kaggle.com/shrutimehta/zomato-restaurants-data

12 Bulk import CouchDB, Available:http://bitbucket.org/tdatta/tools/src/

13 CouchDB Import, GitHub, Available:https://github.com/glynnbird/couchimport

14 CouchDB Selector Syntax, Apache, Available:http://docs.couchdb.org/en/stable/api/database/find.html#selector-syntax