NOSQL Application: Blog Posts in CouchDB

Anonymoous

Abstract— **The demand for efficient data storage is increasing at a rapid rate. More number of organizations are looking to implement databases that can store multiple different types of data in various forms and build horizontally-scalable databases to run on high-performance clusters.**

In this paper, we are going to discuss the various features of CouchDB including storage, indexing, query processing, transaction management, and security support. We further explain why CouchDB performs well and is the right choice of a database for an application to store and process blog posts.

I. OVERVIEW

NoSQL database provides the capability to store and retrieve data which may not be modeled as relations in a table as it is with relational databases(RDBMS). NoSQL databases are regularly used in real-time web applications and big data. The key motivation to use NoSQL databases in big data is because of the ease of horizontal scaling to clusters of machines. Another feature that makes NoSQL databases the preferred choice, is when applications require storage of large volumes of data without structure and want fast retrieval of this data. CouchDBs replication feature makes it an ideal choice for an application which must be replicated on multiple servers, possibly located with significant distance between them.

A typical application that one could use CouchDB for is a blog post application. A blog post application needs some key features including role-based access, browser-based GUI, document storage support and replication which is provided for by CouchDB. Users of the website must be able only to add blogs to the website, but should not be able to remove or delete blog posts. This can be implemented by providing different access permissions to different people; it can be easily performed using CouchDB. A browser-based GUI allows the administrators to view, update and delete blog posts using Mango queries. Mango is highly efficient regarding performance for retrieving large amounts of data. CouchDB has document storage support. The replication feature would help a blog post application regarding storing multiple instances of the same documents across the world either locally or remotely. For example., a user in Tokyo, would view the same instance of the blog posts as a user in the Rochester even though there are two replicated servers present in Japan and USA respectively.

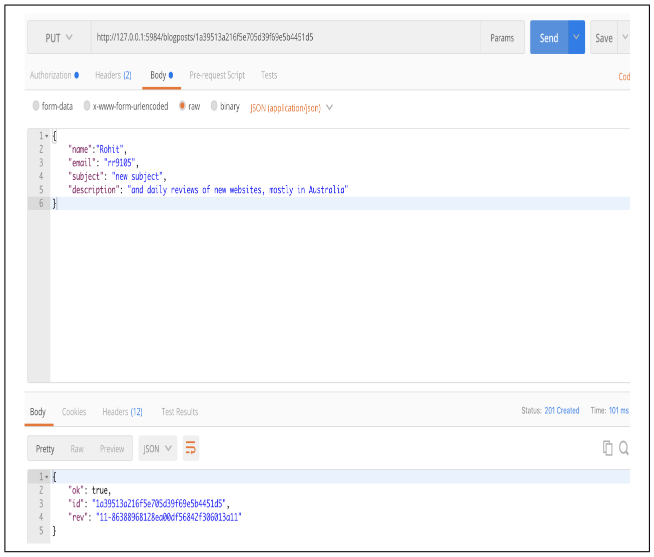
In the following section, we describe how our application performs when using CouchDB versus if we had used a relational database to accomplish the same tasks. The topics we discuss include, transaction management and security support, data storage and indexing, and query processing and optimization.

In regards to query processing and optimization, we will study the behavior in which CouchDB queries through the data. Since this is a NoSQL database, SQL-like queries are for sure not replicated in this database. Instead, there is a use of the MapReduce functional algorithm, in which a specific querying index is constructed, which is called views. We will continue with a more informative study on the specification of the querying process.

Following that of query processing and optimization, transaction management and security support features of CouchDB are discussed. This includes ACID properties, role-playing, hashing passwords, update validation, and replication that has been explained and elaborated upon.

II. DATA STORAGE AND INDEXING

Data is stored in couchDB as documents. It is similar to a record with basic data types being comprised of integers and strings akin to JSONs. Each data record is stored in a JSON format by means of a key, value pair. In the below, image a POST request is used to send the document to couchDB where the database is blogposts. It describes an example how data is stored in this NoSQL DB.



```
http://127.0.0.1:5984/blogposts/1a39513a216f5e705d39f69e5b4451d5
```

```
{
    "name":"Rohit",
    "email": "rr9105",
    "subject": "new subject",
    "description": "and daily reviews of new websites, mostly in Australia"
}
```

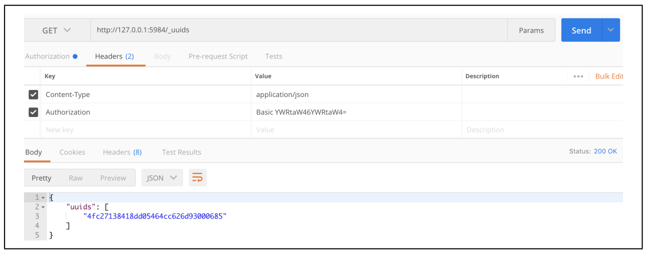
Status: 201 Created Time: 101 ms

```
{
    "ok": true,
    "id": "1a39513a216f5e705d39f69e5b4451d5",
    "rev": "11-86388968128ea00df56842f306013a11"
}
```

Fig. 1. Sending a post request of the blog post

The number of data items to insert depends upon the application. Our application is based on creating a blog-post website where other users with similar interests may comment on peoples experiences.

If a new document has to be inserted to the database, a universal unique identifier must be generated. The uuid is a 32 digit number generated by Math.random() which is RFC 4122 version 4 compliant.This is used to uniquely identify an object and is obtained as:



```
http://127.0.0.1:5984/_uuids
```

Key	Value	Description
☑ Content-Type	application/json	
☑ Authorization	Basic YWRtaW46YWRtaW4=	

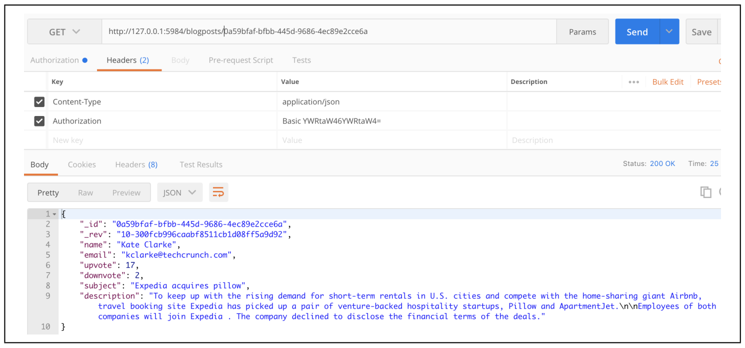
Status: 200 OK

```
{
    "uuids": [
        "4fc27138418dd05464cc626d93000685"
    ]
}
```

Fig. 2. Getting a UUID

A GET request can be issued with the uuid to obtain the URL which makes up the JSON document or data. This can be seen as:

Alongside, we can use an example of Mango being used to query the document, where *name* is equal to *Kate Clarke*:



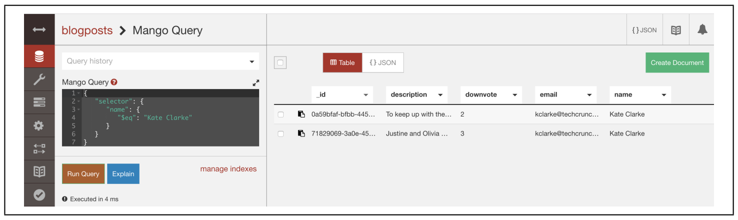
```
http://127.0.0.1:5984/blogposts/0a59bfaf-bfbb-445d-9686-4ec89e2cce6a
```

Key	Value	Description
☑ Content-Type	application/json	
☑ Authorization	Basic YWRtaW46YWRtaW4=	

Status: 200 OK Time: 25

```
{
    "_id": "0a59bfaf-bfbb-445d-9686-4ec89e2cce6a",
    "_rev": "10-300fcb996caabf8511cb1d08ff5a9d92",
    "name": "Kate Clarke",
    "email": "kclarke@techcrunch.com",
    "upvote": 17,
    "downvote": 2,
    "subject": "Expedia acquires pillow",
    "description": "To keep up with the rising demand for short-term rentals in U.S. cities and compete with the home-sharing giant Airbnb,
        travel booking site Expedia has picked up a pair of venture-backed hospitality startups, Pillow and ApartmentJet.\n\nEmployees of both
        companies will join Expedia . The company declined to disclose the financial terms of the deals."
}
```

Fig. 3. Retrieve a blog post with a specific UUID



```
{
    "selector": {
        "name": {
            "$eq": "Kate Clarke"
        }
    }
}
```

Executed in 4 ms

_id	description	downvote	email	name
0a59bfaf-bfbb-445...	To keep up with the...	2	kclarke@techcrunc...	Kate Clarke
71829069-3a0e-45...	Justine and Olivia ...	3	kclarke@techcrunc...	Kate Clarke

Fig. 4. Mango query to retrieve all blog posts by Kate Clarke

Unlike in RDBMS, querying in CouchDB is done through Mango queries. It is a JSON style querying used to retrieve documents. Creating views in CouchDB is much more simpler than in relational databases. To create a view CouchDB uses a function, known as a map function which takes a single argument and returns key/value pairs which makes up a view. RDBMS has predefined data types to store a number, date and a string value. Similar data which has instances related to each other can be grouped together in a single table. Whereas, the data obtained through NoSQL is unstructured and may contain other types of data such as images or audio.

III. QUERY PROCESSING AND OPTIMIZATION

With the rise of popularity of using NoSQL databases, originally query processing was not deemed by developers as high priority; as opposed to the performance of efficiency, large volume handling of data, and flexibility optimizations.

With the NoSQL database, querying possesses a few options in terms of techniques. These approaches are listed as the following:

– *Overall elimination of query processing*: A query free process would allow for a use case of where there is a multitude or infinite amount of varied data to be handled and would want quick scalability

– *MapReduce*: Algorithm where data processing is completed over multiple nodes through decomposition of data into tuples recursively. This algorithm is modeled on a distributed programming basis

- CouchDB has been observed as one of the first among the NoSQL databases to integrate a MapReduce Querying process
- This completely complies with the basis of the document storage model
- Also allows for the enablement of the true architecture consisted in CouchDB: A peer-based distributed database system

– *Querying through DBMS specifications*: Querying through the process of method chaining; usually integrated within MongoDB and allows for developer interaction

– *SQL equivalencies in NoSQL*:The adaptation or reworking of SQL into a non-relational database. Ex. Cassandras CQL (a columnar data model)

CouchDB and views:

The only possible way to query through a dataset in CouchDB is through the use of what is notated as views. The process of views does make use of the MapReduce algorithms known in the computing world. A view in CouchDB is built to contain three columns: (Key, Id, Value), where:

- ***Key*** = Optional argument which allows for any JSON value
- ***Id*** = Id of mapped document is integrated as generated view row
- ***Value*** = Optional argument which allows for any JSON value

A view must be generated first in order for CouchDB to begin the querying process. We may note that the use of reduce within the original MapReduce algorithm is considered as optional when querying with views in CouchDB. However, if the reduce is needed to be used, then the behavior would be of grouping results and returning combined values at the end of the query.

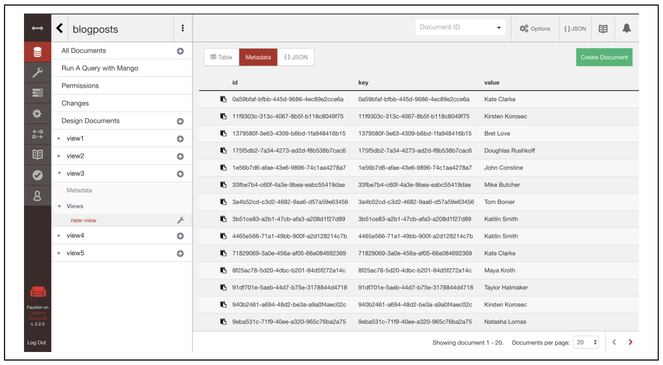
Let us now analyze a little further on the process of views as querying through data within CouchDB:

There are many different cases in which view querying has proven to be useful for the handling of data in CouchDB. One of these ways is by placing the data in a certain structure which resembles the order of which you intend to retrieve the data. With this, CouchDB would hold a view as a view function in the form of a string in the views field of a design document. From this, when the querying of the view is to be requested, then you would be faced with the view result. CouchDB is the main proponent in which the view is queried and run; this is done through the source code itself.

Since CouchDB uses the source code to query the view upon every single document within the database, we can observe that this is equivalent to that of performing a file scan throughout the entirety of the database. Through this, it might take quite a bit of time to perform the whole operation, given that there exists an ample amount of documents within the database already. However, in CouchDB, this file scan operation would run through all the documents at one time in order to avoid any extra expensive costs.

In regards to the storage of the results that we retrieve through view querying, everything is stored in what is known as a B-Tree structure. This structure is the same as the one in which documents are held within CouchDB. This B-Tree structure, that we can notate as the views B-Tree structure for this querying situation, is then stored as a separate file.

If we create an unclustered index on ¡id, name¿, the result of finding a name would be faster than doing a file scan because a B-Tree maintains the data in a sorted manner and thus allows for efficient searches in O(log n) time. This is possible because the data to be searched is compared against each internal node and our subtree for search recursively reduces in size.



id	key	value
0a59bfaf-bfbb-445d-9686-4ec89e2cce6a	0a59bfaf-bfbb-445d-9686-4ec89e2cce6a	Kate Clarke
11f9303c-313c-4067-8b5f-b118c8049f75	11f9303c-313c-4067-8b5f-b118c8049f75	Kirsten Korosec
1379580f-3e63-4309-b6bd-1fa948416b15	1379580f-3e63-4309-b6bd-1fa948416b15	Bret Love
175f5db2-7a34-4273-ad2d-f8b538b7cac6	175f5db2-7a34-4273-ad2d-f8b538b7cac6	Doughlas Rushkoff
1e56b7d6-afae-43e6-9896-74c1aa4278a7	1e56b7d6-afae-43e6-9896-74c1aa4278a7	John Constine
33fbe7b4-c60f-4a3e-9bea-eabc55419dae	33fbe7b4-c60f-4a3e-9bea-eabc55419dae	Mike Butcher
3a4b52cd-c3d2-4682-9aa6-d57a59e63456	3a4b52cd-c3d2-4682-9aa6-d57a59e63456	Tom Bonier
3b51ce83-a2b1-47cb-afa3-a208d1f27d89	3b51ce83-a2b1-47cb-afa3-a208d1f27d89	Katilin Smith
4465e566-71a1-49bb-900f-a2d128214c7b	4465e566-71a1-49bb-900f-a2d128214c7b	Katilin Smith
71829069-3a0e-456a-af05-66e084692369	71829069-3a0e-456a-af05-66e084692369	Kate Clarke
8f25ac78-5d20-4dbc-b201-84d5f272a14c	8f25ac78-5d20-4dbc-b201-84d5f272a14c	Maya Kroth
91df701e-5aeb-44d7-b75e-3178844d4718	91df701e-5aeb-44d7-b75e-3178844d4718	Taylor Hatmaker
940b2461-a694-48d2-be3a-a9a0f4aec02c	940b2461-a694-48d2-be3a-a9a0f4aec02c	Kirsten Korosec
9eba531c-71f9-40ee-a320-965c76ba2a75	9eba531c-71f9-40ee-a320-965c76ba2a75	Natasha Lomas

Showing document 1 - 20. Documents per page: 20

Fig. 5. Example of the Query Optimization through the B-Tree in CouchDB

Optimization of Views:

The optimization of the query processing within CouchDB lies in the innate structure and use of the B-Tree. This B-Tree is used by CouchDB in order to index through the documents and query views. Now let us look into more about the B-Tree structure to get a better understanding.

The B-Tree data structure is primarily objectified to handle large volumes of data. In terms of depth the B-Tree does hold an ample amount of depth, however it is designed to expand to a large-scale width. Typically, this tree will only reach a height that is under the height of 10, no matter how many entries lie within the structure.

In a CouchDB B-Tree, the data structure makes use of Multi-Version Concurrency Control (MVCC). The MVCC will allow for simultaneous read and write operations that run in parallel to each other. Through serialization, only one write may be able write at a time. Dissimilarly, the read operations perform no such behavior and are able to have multiple reads go at one time. Along with the concurrency control, the B-Tree structure specific to a CouchDB implements an append-only design. We will cover more of this in the further section of Transaction Management and Security Support, however it is important to note the process of concurrency control in regards to how it aids in optimization of query processing through the database.

It is also important to note that all data retrieved is stored only in the leaf nodes of the tree. With this, only keys are ordered upon within the tree, making the access of keys similar to the model of a cluster, where associated keys will be close in vicinity of each other. Therefore, the beginning and ending of reading keys will contain less of a gap.

Through the previous points made, we can concur that the B-Tree CouchDB structure allows for a cheap option of reading, mainly due to the sorted nature of the structure.

Overall Thoughts On Querying Views:

- Being able to find specifications of a process pertaining to the documents
- Efficiency of indexing
- Indexes additionally being able to represent any relationships across multiple documents
- Producing ordered data from documents and/or performing additional operations such as calculations on the data retrieved

IV. TRANSACTION MANAGEMENT AND SECURITY SUPPORT

In CouchDB, multiple users can read a file. It is serialized on features such as updates, delete, and adding a document. The data is stored in a compact manner whereas, for an RDBMS, it is stored in different tables. Whenever a document is updated, the data and its index is flushed onto the disk. The update database header is written in two chunks and then flushed synchronously onto the disk. The users do not encounter any locks as opposed to the cases of relational databases. In an RDBMS, ACID (Atomicity, Consistency, Isolation, Durability) properties can only be achieved by locking and versioning the data.

CouchDB is ideal for this application as multiple users are allowed to read the blogs. Whereas when a user wishes to update any information, no other user should be allowed to work on the same document.

In comparison to relational database management systems, CouchDB offers better security features by means of allowing access with different privileges to two kinds of users: members and admins. In RDBMS, while there are innate support and infrastructure existing to sustain concurrent access, there is also a downfall in this regard. This downfall consists of the fact that since in an

RDBMS system, data integrity is the key objective which causes for a locking system to be created. This locking system is where the data that is trying to be accessed will first be marked in order to ensure no other user can access this same data. As this does indeed preserve data integrity as the main goal, this is not a feasible mechanism to handle data in a large scale, where there would exist a higher multitude of users as this affects the performance level greatly. However, in contrast to an RDBMS system to CouchDB, this behavior is easily avoided. This is due to the mere fact that a NoSQL database system completely abstains from using the locking system at all. This allows for a higher and unhindered level of performance for the database. By this, a larger assemblage of users are welcomed and will have the proper support without the trade off of time or flexibility of performance.

With simultaneous data access in CouchDB, members can read, add and edit documents if they are provided with access to do so. Administrators can add, edit and read all the documents as well the security document. The security document stores the member's id of all the members with access to update and delete documents.This way the administrator can set the privileges of other users. To avoid a rogue member or security breach, a username and password can be created for the administrators. This ensures only the administrator is allowed to do certain tasks like updating, deletion, or addition of data. The password created can be hashed for an extra layer of protection as hashed passwords cannot be read. It also supports cookie authentication to ensure a secure connection. This way the user is limited to only the databases he has access to. Cookie authentication can be done easily with a regular HTML post form. It also has an update validation feature that ensures that every time a document is updated, the updated document and user credentials is being given to the validation formula.

CouchDB is configured to listen 127.0.0.1 at port 5984 by default and the remote CouchDB listens to 0.0.0.0 by default. This NoSQL database has a replication system. The database is stored on the server as well as a remote server. CouchDB makes a distinction among the two so if there is a change on one of the servers, it makes the change on the other server as well. As the database changes the sequence number of the document each time it is updated, CouchDB checks for changes made by checking the sequence number. This replication feature is useful as there is a backup of all the databases, in case of a system failure while updating or other reasons. Another way to take advantage of this feature is to create snapshots of the data. The test cases are run on the snapshots and not the original data. RDBMS does not have such security features to offer. It is better to use CouchDB over an RDMS in scenarios where security and transactional features are a priority.

For this application, only the admin has access to all the documents. The users are given role-based security access to the database. The author of the blog is only allowed to update their blogs or comments. Other members/user can read the documents posted by their fellow users. A feature of this application is that a blog post can be either private or public. If the post is private, then no one but the author can read the post. Whereas, if the post is public, then all the users can read the blog.

V. NoSQL Application

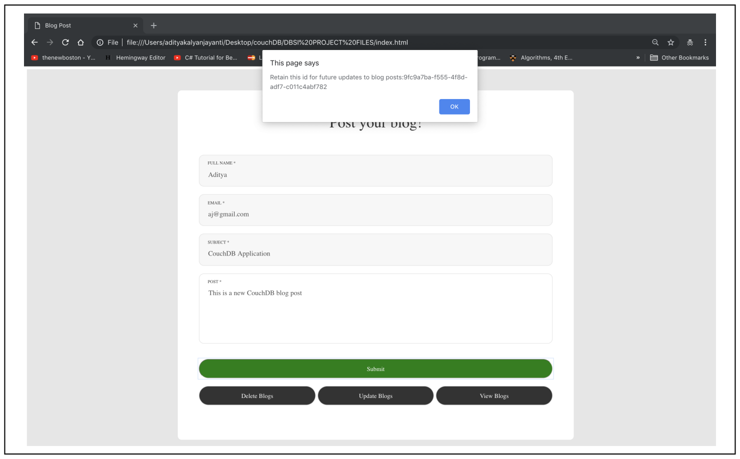
The NoSQL database application we have built is used to create and blog posts. Following the installation of CouchDB, we used Fauxton to create an account with admin privileges, which would enable us to create and delete databases. Fauxton is an easy to use browser-based GUI that allows us to create, read, update and delete databases and documents. A document is the building block of CouchDB, it can be visualized as a record in a relational database. Every document is uniquely identified by a document id. Subsequently, we used Postman, a REST API client, to send requests and receive responses. Postman described the structure of the responses that one could expect when sending requests to CouchDB via the exposed APIs.

Further, we built a simple HTML+CSS webpage. The HTML webpage provided a form with input fields including, author name, author email, subject of the blog and the real blog post itself. The CSS was used to improve the aesthetics of the webpage and to make it more user-friendly.

Once entering to this first HTML/CSS webpage, the user is now faced with multiple options in regards to the blog he or she is to submit or already have submitted. These options being pertaining to submitting, viewing, updating, or deleting a blog. Evidently, these options are in direct correlation with the basic CRUD operations that serve as the functionality to prove that a model is complete.

For the example of one instance, when the user is ready to submit, the user must click on the submit button. When the user hovers over the button, it changes color (green in this case of submission of blog) indicating that the form can be successfully submitted. Upon clicking the button, the javascript function is invoked using an onclick listener. The javascript function uses XML-HTTPREQUEST to send a PUT request to create this new document in CouchDB. The javascript function directly queries the CouchDB database to insert the data into the database with a uniquely generated a UUID or a universally unique identification number. Based on the response, we alert the user of the success/failure status. This makes use of one of the key features of CouchDB, i.e., being able to directly query from JavaScript rather than having a middleware such as Java or Node JS to query the database.

Elaborating further on the example as given, upon the success of submission, the user will now be faced physically with the respective UUID pertaining to the specific blog being submitted. As the UUID is offered as a unique id pertaining to the blog, or the document in a general sense, it will also be preserved for the user in our application. The user can then make use of this id as is or if they wish to manipulate their respective blog post with further operations, they may do so. For example, after obtaining this UUID, a user is able to delete the blog post, only once entering the valid UUID for that blog. This UUID will match as the unique id in the backend database, and the document will be deleted accordingly. This remains as a huge benefit, as this allows for security of the documents within the database. As in, it would be highly controversial if a user is able to manipulate any other blog within the backend which is not theirs. Therefore, the structure of UUID assignments is imperative.



This page says

Retain this id for future updates to blog posts:9fc9a7ba-f555-4f8d-adf7-c011c4abf782

Post your blog!

FULL NAME *
Aditya

EMAIL *
aj@gmail.com

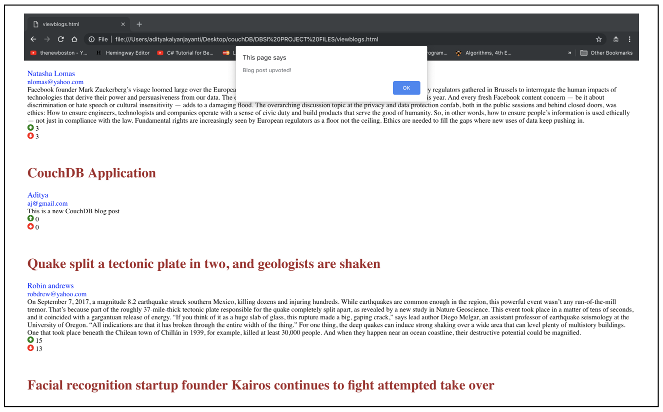
SUBJECT *
CouchDB Application

POST *
This is a new CouchDB blog post

Fig. 6. Image of the user being presented with the UUID upon presenting

We further run queries on this data by means of creating views. Views are considered building blocks of a CouchDB applications, because they help you query the database using a map function. Another key feature of CouchDB we explored is using MapReduce by means of creating views, but the reduce function is optional in CouchDB. The blog website is designed such that, the user can view other blog posts and would need to sign in to comment on other bloggers. Each blog is associated with upvotes/downvotes and it can be ranked based on the number of votes. However, only administrators can delete posts. This provides for a role-based access for the CouchDB instance by making sure that no user has access to any blog that is outside of their respective UUID scope. We give access to any person who may have the correct username, password as specified in our application as admin, admin. This ensures that no external user besides the administrator may manipulate any of the documents or blogs lying in the backend. The use of such an administrator ties into the implementation of a JSON token which uses the associated username, password as the role-based access mechanism.

Examining a bit more on the Upvote/Downvote system created for this application, we can observe more in depth the advances of the implementation we aimed for within this final stage of the application. For this, each blog is retained in a JSON data format in the configuration as lists. This meaning that once the user tries to query through the database to retrieve all blogs, the database will then return a JSON list containing all the blogs. Through this, the list of JSON formatted blogs are then parsed through to display the JSON in a readable format. Each time the user obtains and is able to view the blog, they are also presented with an option to Upvote or Downvote on the blog, based on their preference towards the blog post. If the user clicks the Upvote button, the UUID and Upvote value associated with the respective blog are retrieved accordingly. Based on the information attained, a new entry is made within the database which preserves the UUID but will change the Upvote value to an updated amount. Similarly, the Downvote procedure works in the exact same fashion.



This page says

Blog post upvoted!

Natasha Lomas
nlomas@yahoo.com
Facebook founder Mark Zuckerberg's visage loomed large over the Europea y regulators gathered in Brussels to interrogate the human impacts of technologies that derive their power and persuasiveness from our data. The is year. And every fresh Facebook content concern — be it about discrimination or hate speech or cultural insensitivity — adds to a damaging flood. The overarching discussion topic at the privacy and data protection confab, both in the public sessions and behind closed doors, was ethics: How to ensure engineers, technologists and companies operate with a sense of civic duty and build products that serve the good of humanity. So, in other words, how to ensure people's information is used ethically — not just in compliance with the law. Fundamental rights are increasingly seen by European regulators as a floor not the ceiling. Ethics are needed to fill the gaps where new uses of data keep pushing in.
⬆ 3
⬇ 3

CouchDB Application

Aditya
aj@gmail.com
This is a new CouchDB blog post
⬆ 0
⬇ 0

Quake split a tectonic plate in two, and geologists are shaken

Robin andrews
robdrew@yahoo.com
On September 7, 2017, a magnitude 8.2 earthquake struck southern Mexico, killing dozens and injuring hundreds. While earthquakes are common enough in the region, this powerful event wasn't any run-of-the-mill tremor. That's because part of the roughly 37-mile-thick tectonic plate responsible for the quake completely split apart, as revealed by a new study in Nature Geoscience. This event took place in a matter of tens of seconds, and it coincided with a gargantuan release of energy. "If you think of it as a huge slab of glass, this rupture made a big, gaping crack," says lead author Diego Melgar, an assistant professor of earthquake seismology at the University of Oregon. "All indications are that it has broken through the entire width of the thing." For one thing, the deep quakes can induce strong shaking over a wide area that can level plenty of multistory buildings. One that took place beneath the Chilean town of Chillán in 1939, for example, killed at least 30,000 people. And when they happen near an ocean coastline, their destructive potential could be magnified.
⬆ 15
⬇ 13

Facial recognition startup founder Kairos continues to fight attempted take over

Fig. 7. Example image of Upvoting on a Blog post

A NoSQL database such as CouchDB is an ideal choice for the blog post application for a multitude of reasons, viz., blog posts can be of many types, textual, photos or videos. A relational database would not allow a description or the actual blog post to hold a photo or a video. Another reason to use a CouchDB is that it can be directly queried from a front-end tool such as JavaScript, through CouchDBs exposed APIs, making it a low latency query since there are lesser number of hops to access the data. Relational databases, on the other hand, cannot be directly accessed from a front end tool such as javascript. CouchDB provides a native easy to use web-based GUI which makes accessing the database and information easy. You need not have any other tools apart from CouchDB and a browser to work with the data unlike the case with relational databases. Along with this, CouchDB is one of the rarer databases systems in which mobile support is also available. Therefore, using the browser whether it is on your computer or mobile phone will both be acceptable to the use of a CouchDB application, such as blog posts. As any blogger would want to be able to post, read, or do any other type of viewing with their own or other peer's blogs all while on the go, this proves that CouchDB is a unique fit as the versatility offered is extremely useful. Other database systems, such as MongoDB, do not propose such compliance's as mobile support. And lastly, CouchDB provides replication, meaning, multiple instances of the same database would exist over different servers which is useful if a server were to fail. Keeping all these benefits in mind, CouchDB was an ideal choice for this implementation of the blog post application.

VI. Final Remarks

In all, we are able to gain a better insight as to what CouchDB is and what it has to offer as a standalone NoSQL database. Along with this, we can clearly observe that for our specific application of handling blog posts, a relational database such as SQL would not be of a good fit. This is due to the requirements the application demands. Meaning, that as a blog post is of a document style in and of itself, it would make the most sense and it would be of the most well-informed decision to choose a database in which the document of a blog post could most likely be replicated in the backend of the database as well. This meaning that we would want to store our data of the application at the backend in the most similar fashion as it is represented through the frontend.

With this in mind, CouchDB offers a perfect glove fit to our model since the core of such a NoSQL database is to store not records, but documents. As we do choose CouchDB as our database mainly for the reasoning of the document designed data structure, we are also offered various benefits through using this non relational database. These advantages including added security components, a distributed and peer to peer based cluster set up, and fundamental functions for persistent storage of documents. Therefore, CouchDB is our optimal find for us to model and aid us through the application.

APPENDIX

Abstract and Overview: *Aditya, Anika, Rohit*
Data Storage and Indexing: *Aditya*
Query Processing and Optimization: *Anika*
Transaction management: *Anika*
NoSQL Database application and implementation: *Rohit and Aditya*
Final Remarks: *Anika*

REFERENCES

[1] http://guide.couchdb.org/draft/views.html
[2] http://guide.couchdb.org/draft/btree.html
[3] javatpoint.https://www.javatpoint.com/features-of-couchdb
[4] http://docs.couchdb. org/en/2.2.0/intro/overview.html
[5] https://en.wikipedia.org/wiki/NoSQL
[6] https://blog.panoply.io/sql-or-nosql-that-is-the-question
[7] http://docs.couchdb.org/en/stable/cve/index.html
[8] https://www.ietf.org/rfc/rfc4122.txt
[9] http://marcgrabanski.com/ exiciting-features-in-couchdb/
[10] https://www.oreilly.com/library/view/writing-and-querying/9781449303693/ ch05s02.html
[11] https: //www.exoscale.com/syslog/nosql-query/
[12] https://www.tutorialspoint.com/couchdb/couchdb/introduction.htm