SCHEMA AGNOSTIC INDEXING WITH LIVE INDEXES

Monica Bansal
M.Tech Information Technology, Dronacharya College of Engineering

Abstract- **Now-a-days, schema is the most popular standardized language to describe data. Developers are working with applications that create massive volumes of new, rapidly changing data types — structured, semi-structured, unstructured and polymorphic data. Long gone is the twelve-to-eighteen-month waterfall development cycle. Now small teams work in agile sprints, iterating quickly and pushing code every week or two, some even multiple times every day. Organizations are now turning to scale-out architectures using open source software, commodity servers and cloud computing instead of large monolithic servers and storage infrastructure. Relational databases were not designed to cope with the scale and agility challenges that face modern applications, nor were they built to take advantage of the commodity storage and processing power available today. This research paper proposes a model to allow insertion of the data without a predefined schema and also with schema agnostic indexing with the concept of live indexes. It makes it easy to make significant application changes in real-time, without worrying about service interruptions – which means development is faster, code integration is more reliable, and less database administrator time is needed.**

Index Terms – automatic indexing, live indexes, nosql, **schema agnostic**

I. INTRODUCTION

In the recent boom of the world of enterprise computing, many changes have been seen in platforms, languages, processes, and architectures. But throughout a major of time, in the software profession, relational databases have been the default choice for serious data storage, especially in the world of enterprise applications. NoSQL comes into picture when some data in hand crosses some threshold of Relational DB in terms of requirements like practically 100% availability & reliability, extremely distributed environments, undeterminable scalability requirements with practically no room for availability sacrifice, no fixed schema etc. In such a situation, traditional systems fail to provide expected outcomes. Based on priorities of mentioned parameters, appropriate NoSQL systems are chosen. Recently there have been many such new models booming to change the RDBMS monopoly, but most of these options come as a complete replacement of the current systems and that too mostly beneficial just for the said scenarios.

The technology used in this thesis combined with the proposed system aims to enable the developer to use the proposed system along with the RDBMS, both handling the parts of data that they are efficient and good at handling. Goals of the research are: no explicit indexing required in the proposed system, provide an alternative solution to Relational Database Management System, Currently RDBMS using projects should easily be able to adapt the proposed system

II. RELATED WORK

A. Schema-Agnostic Database

In a relational database, if you define schema in advance, then every time you throw data at the database, it must match that schema. A schema agnostic database system can take data, and no matter what the schema is, as long as it's well formed XML, it can parse it and store it in the structure that is supplied by the XML tree.

Schema agnostic databases are not bound by schemas — but are aware of the schemas – and specific schemas can be enforced at the database level if desired or necessary.

B. Schema-Agnostic Indexing

With a goal to eliminate the impedance mismatch between the database and the application programming models, we can exploit the simplicity of JSON and its lack of a schema specification. It makes no assumptions about the documents and allows documents within a database collection to vary in schema, in addition to the instance specific values. For example, in DocumentDB's database, engine operates directly at the level of JSON grammar, remaining agnostic to the concept of a document schema and blurring the boundary between the structure and instance values of documents.

C. Azure DocumentDB

It is a fully managed, highly scalable, NoSQL, multi-tenant, distributed database as-a-service for managing JSON documents at internet scale that offers rich query and transactional processing over schema-free data. As a JSON database, DocumentDB natively supports JSON documents enabling easy iteration of application schema, and support applications that need key-value, document or tabular data models. DocumentDB embraces the ubiquity of JSON and JavaScript, eliminating mismatch between application defined objects and database schema. Deep integration of JavaScript also allows developers to execute application logic efficiently and directly - within the database engine in a database transaction.

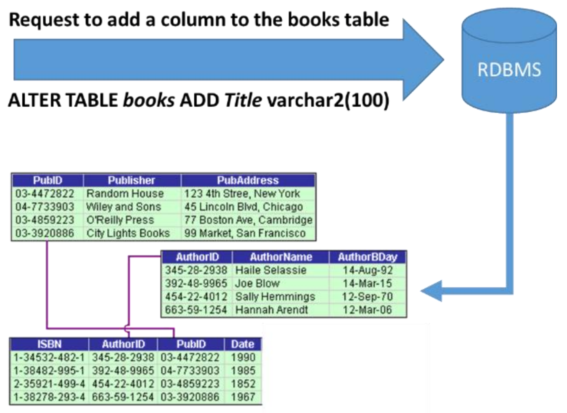
III. LIVE INDEXES

In this work, I propose to harness the capability of this indexing idea in working projects as a part of its source model rather than as database model. In this entire section we will discuss on how this will be designed and implemented.

The idea of this proposal is to facilitate those projects which currently have complete dependencies on their relational data base management systems to handle all their data related requests. For such projects, which fall under the further mentioned target datasets, I propose to create a separate source model, which will act as a subordinate data management system to handle the mentioned data.

A. Comparison

In the existing RDBMS model, when clients send request to the database system, the database system processes the request and sends the output. For the relational database, developers need to manage the database system because it requires change in schema whenever a alter column or create table request comes in. It becomes a tedious task to maintain the database schema as changes in schema should not affect the existing database model.



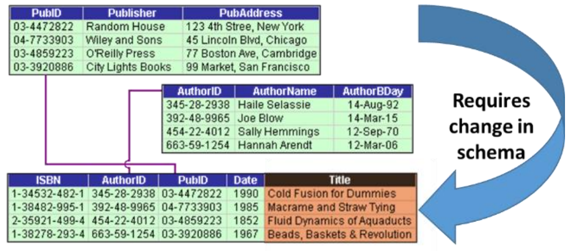
```
ALTER TABLE books ADD Title varchar2(100)
```

PubID	Publisher	PubAddress
03-4472822	Random House	123 4th Stree, New York
04-7733903	Wiley and Sons	45 Lincoln Blvd, Chicago
03-4859223	O'Reilly Press	77 Boston Ave, Cambridge
03-3920886	City Lights Books	99 Market, San Francisco

AuthorID	AuthorName	AuthorBDay
345-28-2938	Haile Selassie	14-Aug-92
392-48-9965	Joe Blow	14-Mar-15
454-22-4012	Sally Hemmings	12-Sep-70
663-59-1254	Hannah Arendt	12-Mar-06

ISBN	AuthorID	PubID	Date
1-34532-482-1	345-28-2938	03-4472822	1990
1-38482-995-1	392-48-9965	04-7733903	1985
2-35921-499-4	454-22-4012	03-4859223	1852
1-38278-293-4	663-59-1254	03-3920886	1967

Fig 1 - Alter Table Query



PubID	Publisher	PubAddress
03-4472822	Random House	123 4th Stree, New York
04-7733903	Wiley and Sons	45 Lincoln Blvd, Chicago
03-4859223	O'Reilly Press	77 Boston Ave, Cambridge
03-3920886	City Lights Books	99 Market, San Francisco

AuthorID	AuthorName	AuthorBDay
345-28-2938	Haile Selassie	14-Aug-92
392-48-9965	Joe Blow	14-Mar-15
454-22-4012	Sally Hemmings	12-Sep-70
663-59-1254	Hannah Arendt	12-Mar-06

ISBN	AuthorID	PubID	Date	Title
1-34532-482-1	345-28-2938	03-4472822	1990	Cold Fusion for Dummies
1-38482-995-1	392-48-9965	04-7733903	1985	Macrame and Straw Tying
2-35921-499-4	454-22-4012	03-4859223	1852	Fluid Dynamics of Aquaducts
1-38278-293-4	663-59-1254	03-3920886	1967	Beads, Baskets & Revolution

Fig 2 - Alter Table Result

B. Indexes

For the existing relational DBMS, indexes are used to improve the speed of the data retrieval operations on a database table. They are used to quickly locate the data without having to search every row in a database table every time a database table is accessed. For the existing model, we have to create indices every time when there is a need to access the data faster. Every database management system provides ways to create indices or delete them as per the requirements of the developers.

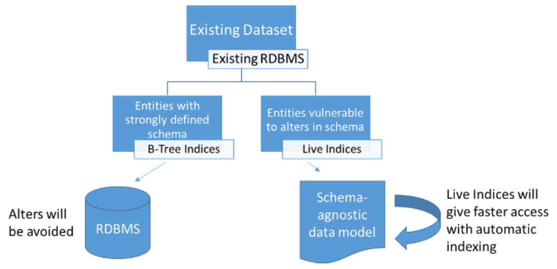

Existing Dataset
Existing RDBMS
Entities with strongly defined schema
B-Tree Indices
Entities vulnerable to alters in schema
Live Indices
Alters will be avoided
RDBMS
Schema-agnostic data model
Live Indices will give faster access with automatic indexing

Fig 3 - Index Access Comparison

C. Design

In the proposed work we will consider each document as a JSON object. The system will take such documents as input to create index. It will iteratively parse each key value pair and will form triads according to the hierarchy of the values. We will utilize the hash map data structure and use these triads as key of the map. The value will then contain a list of the identifier of the documents. Collating with the recommendations of DocumentDB, we can even create multiple hash-maps for different indexing purpose like in the following diagrams:

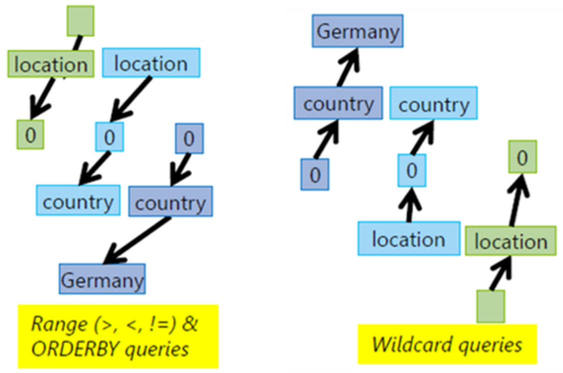
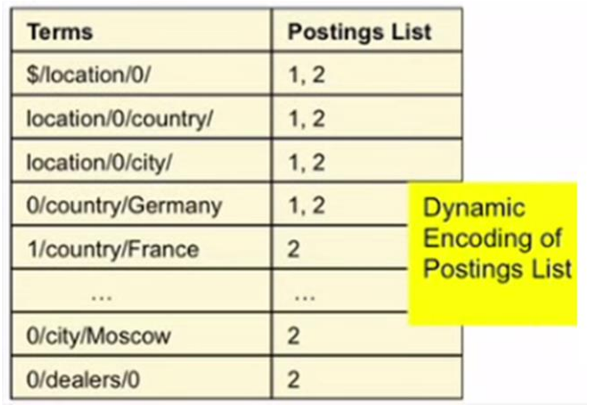


Fig 4 - Types of Path Listing



Terms	Postings List
$/location/0/	1, 2
location/0/country/	1, 2
location/0/city/	1, 2
0/country/Germany	1, 2
1/country/France	2
...	...
0/city/Moscow	2
0/dealers/0	2

Dynamic Encoding of Postings List

Fig 5 - Dynamic Encoding of Posting Lists

The proposed design will keep this hash map on the server static memory. This means that the map will be parsed and cached on the server. We can implement the Least Recently Used technique to segregate the entire listing into main memory and persisted as files. We will discuss the scope of this in possible future works.

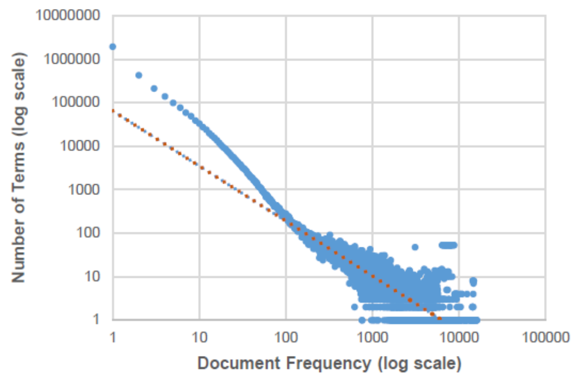


Fig 6 - Document Frequency vs Number of Terms

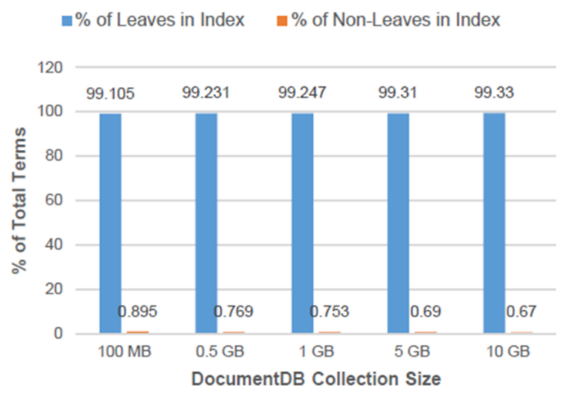
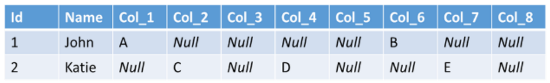


Fig 7 - Collection Size to Percentage of Total Terms

IV. TARGET DATASETS

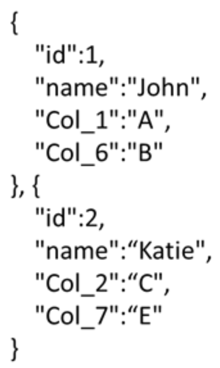
A. Sparse Data

Sparse Data are those kinds of datasets which have usually very large number of attributes/ columns but each individual entry of that entity, has value for a comparatively very less number of these attributes. Sparse data is a very big deal in the current generation of datasets. It is intuitively clear why RDBMS is not at all an ideal way to store and manage such kinds of databases. This is because the table for such entities will contain large amount of columns with value of less than 30% of these. The main issue with this is that it will become unreasonably very expensive to create any secondary index on such data. Also, the decision to select for which permutation should the developer create secondary indices becomes very crucial and will always end with high compromise in efficiency.



Id	Name	Col_1	Col_2	Col_3	Col_4	Col_5	Col_6	Col_7	Col_8
1	John	A	*Null*	*Null*	*Null*	*Null*	B	*Null*	*Null*
2	Katie	*Null*	C	*Null*	D	*Null*	*Null*	E	*Null*

Fig 8 - Sparse Dataset prototype



```
{
   "id":1,
   "name":"John",
   "Col_1":"A",
   "Col_6":"B"
}, {
   "id":2,
   "name":“Katie",
   "Col_2":“C",
   "Col_7":“E"
}
```

Fig 9 - Sparse Data as JSON

B. Flexible Schema

In many cases, there is no constant definition of schema. Data can keep on flowing without any particular schema which becomes extremely difficult to manage with an RDBMS system. Although, logically common entities will still have many common attribute keys, but there can be addition of new keys without predefinition. There should definitely be a better way to handle such a data where schema should not be a pre requisite to enter an entry. This JSON driven proposal will do precisely that thing. In this case, the entries will define the structure and logical schema of the data in hand. Any entry will be JSON converted and then added to the files managed by this system. This gives freedom to the users as well as developers to handle any number of attributes without any issues of altering and efficiency. One of the most beneficial features being, two similar entities with different attributes can still be managed/compared together.

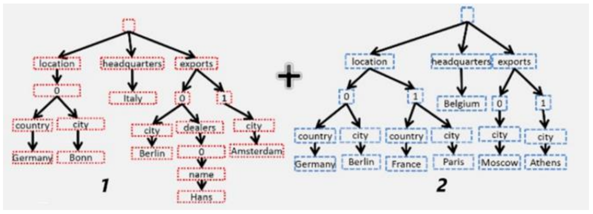


Fig 10 - Index formation from multiple trees

The above diagram shows example of 2 documents with similar entities, but different attributes. It can be observed that document 1 has an addition of "dealers" in exports. That too is not consistent in every index of "exports" array. Such data can be directly compared and also unified for creation of index as follows:

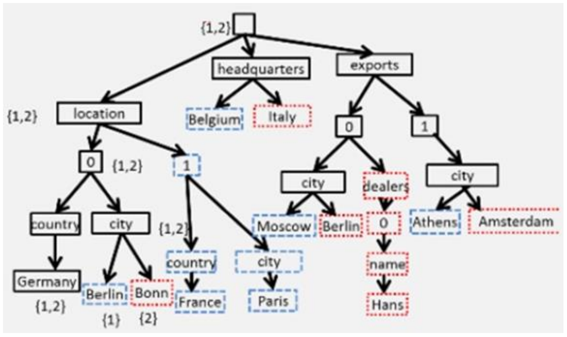


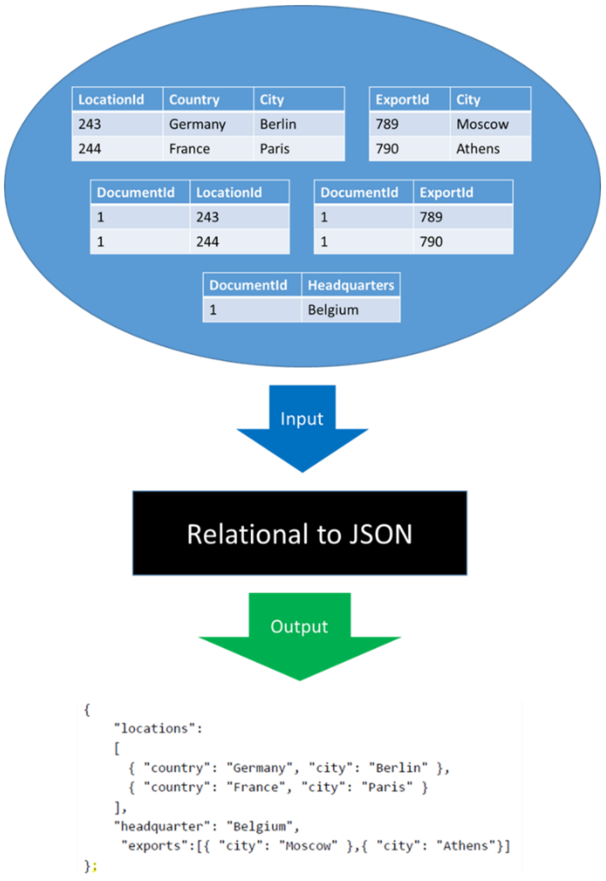
Fig 11 - Final Index

D. Historical Data

Many use cases include very large amount of historic data which are not very frequently accessed but still are stored in RDBMS with the same privileges as that of most frequently accessed current data. Many keep a practice of taking a dump of this data outside the database management system and then loading it back on system to access it, which gives us the mentioned situation again.

V. SYSTEM MODELS

In this design we have proposed that how relational data that is stored in form of tables can also be stored in JSON format. We have proposed a model that will take relational data in form of tables and will give JSON data as an output. This model doesn't require any schema data to store data in JSON format. But to store data in tables we are require to provide schema data to database management system. It can also be used for the existing system which do not want to change the existing model but have large data, so for those type of system this model can take the relational data and produces JSON string output and can stored that JSON strings in files and then those files can be used for retrieving data. We have also proposed a concept 'Live Index' that will help us to fetch the document fast without providing any explicit index. Thus we can conclude that this model states that for any data type, data set as a input we can transformed that data set into JSON format and then stored that JSON string in file. Now a days, we use relational data model so we have considered that data store, it does not make any difference for the relational data nor we have to specify any relation between the tables while transforming it into JSON string. For this model, we don't have to mention explicitly about any relation between the entities that we are storing in the files. It is also a very simple task to retrieve the data from the transformed files.



LocationId	Country	City
243	Germany	Berlin
244	France	Paris

ExportId	City
789	Moscow
790	Athens

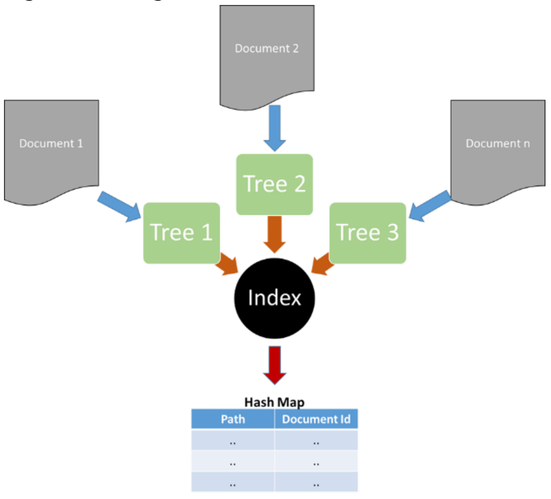
DocumentId	LocationId
1	243
1	244

DocumentId	ExportId
1	789
1	790

DocumentId	Headquarters
1	Belgium

```
{
    "locations":
    [
      { "country": "Germany", "city": "Berlin" },
      { "country": "France", "city": "Paris" }
    ],
    "headquarter": "Belgium",
     "exports":[{ "city": "Moscow" },{ "city": "Athens"}]
};
```

Fig 12 - Existing Relational to JSON model



Path	Document Id
..	..
..	..
..	..

Fig 13 - Documents to HashMap flow

After all the data of each required entity is converted and saved as a JSON format as explained above, we will now see what happens during the execution.
Now, when a new instance of the proposed system is created for an entity, firstly, all the JSON documents will be read, converted to trees as explained in the previous sections. These trees are the combined to create the index required for our model. This index is saved as a hash map in the instance.

VI. FUTURE SCOPE

Currently we are creating an instance of the proposed system which will contain all the indexes on the main memory. Future scope can implement the Least Recently Used technique to segregate the entire listing into main memory and persisted as files. This will reduce the load on the main memory & will avoid creation of indices every time an instance is created. This will dramatically increase the performance once succeeded.
The proposed system can also be adapted for Object Relation Mappings. This sounds contradictory as one of our points was to avoid Object Relational Mappings to in turn avoid impedance mismatch. But this adaptation will be completely superficial & internally will work in exactly the same way as described above. But this will make being able to adapt this system all the easier for such projects which use ORMs like Hibernate.
With continuous sustained input of data in our proposed document store, updating indexing becomes a tedious job for the system. The main factor being that updates are carried in different locations of our document store. This calls for a need of providing a solution to save the number of writes and unnecessary reads.
Document Store and how documents are arranged in it is not well defined yet. We can use a system file hierarchy for the same. This will involve grouping documents of same entities and placing them in a common folder. This can be even made better in future.

VII. CONCLUSION

This research paper described the design and model of Live Indexes for managing JSON documents at massive scale.
We first studied current systems in this field headlined by DocumentDB. Then we defined what we plan to achieve in this research which was mainly providing a flexible alternative to RDBMS with definite benefits. We defined that our target datasets will have the virtues of one or more from sparse, flexible and historic. We then described the models using black box diagrams which describe how we will have current systems migrated to our proposed

systems and what changes must be made to the current system to adapt out proposal.
We designed our module to be schema-agnostic by representing documents as trees. Support of automatic indexing of documents is provided by default. Finally, we mention the future scope of this proposal.

APPENDIX

JSON - JavaScript Object Notation
CRUD - Create, read, update and delete
ORM - Object Relational Mapping
MVC - Model, View, Controller

ACKNOWLEDGMENT

I would like to express my sincere gratitude to all those people who have given their heart willing support in making this completion a magnificent experience.
I am thankful to Ms. Megha Goel, H.O.D, Department of Technology, Dronacharya College of Engineering, Gurgaon for providing us good and healthy environment for the preparation of this dissertation.
I am also thankful to my dissertation guide Mr. Dharmendra Pal, Associate Prof., Department of Information Technology for his timely comments and suggestions. He advised on the details and gave invaluable discussions. Without the guidance of my supervisor, this dissertation may not have well materialized.
I am really grateful to my parents for their support, appreciation and encouragement. A very special acknowledgement to authors of various research papers and books which help me a lot.

REFERENCES

[1] Shukla, Dharma, et al. "Schema-agnostic indexing with Azure DocumentDB." Proceedings of the VLDB Endowment 8.12 (2015): 1668-1679.
[2] Copeland, Marshall, et al. "Overview of Microsoft Azure Services." Microsoft Azure. Apress, 2015. 27-69.
[3] Crockford, Douglas. "The application/json media type for javascript object notation (json)." (2006).
[4] Pokorny, Jaroslav. "NoSQL databases: a step to database scalability in web environment." International Journal of Web Information Systems 9.1 (2013): 69-82.
[5] Liu, Zhen Hua, Beda Hammerschmidt, and Doug McMahon. "JSON data management: supporting schema-less development in RDBMS." Proceedings of the 2014 ACM SIGMOD international conference on Management of data. ACM, 2014.
[6] Becker, Riccardo. Learning Azure DocumentDB. Packt Publ., 2015.
[7] Sriparasa, Sai Srinivas. JavaScript and JSON Essentials. Packt Publishing Ltd, 2013.
[8] Sadalage, Pramod J., and Martin Fowler. NoSQL distilled: a brief guide to the emerging world of polyglot persistence. Pearson Education, 2012.
[9] https://azure.microsoft.com/en-us/documentation/services/documentdb/
[10] https://en.wikipedia.org/wiki/Database_schema