Storage Engine that Handles Structured, Semi-structured and Un-structured Data

Prof. Shiv Kumar Goyal[1] Arvind Mihirlal Adak[2]
[2]Deputy Head of Department
[1,2]Department of MCA
[1,2]VESIT, University of Mumbai, Mumbai – 400074, India

Abstract— Database management system now-a-days are specialized in handling specific type of data i.e. most of them are specialized in storage of structured data. Very few software are available to handle semi-structured and unstructured data. Though not frequently required but the later mentioned two types of data are evenly useful in many applications where the structure of data is not fixed or simply vague. This proposed system addresses the same issue and put forward an concept and working mechanism of a system that handles all the above mentioned data structures. The idea is to make a generic storage engine, which takes inputs/command/data from other query processing engines. And stores it based on the nature and structure of data. Which is independent of the file query execution engine.

Key words: Structured, Semi-structured, Un-structured

I. INTRODUCTION

Database Management systems Now-a-days are much more than just relational database storage system. They not just manage data of the applications they are being used with, but also contribute a lot to performance of the application. However the current trend of cloud is taking the data storage requirements to a new horizon. Where a Database management should not only be storing and handling structured data (like in the case of the relational database management system). But should also address some real life data storage scenarios like semi-structured data and un-structured data.

Data that fall into the category structure data, are well organized. And quite literally the could be organized into table like structures (like in the case of relational database management system). Whereas semi-structured data are of the category in which data is organized i.e. data related to one particular entity may have more information fields than other similar entity. In other words the Meta data of each entity of data differs. And hence it cannot be completely organized using a static structure. However there exist some common fields of data among all the entity which forms the base for classification of this type of data. On the other hand un-structured data are completely unorganized. There are no fields based on which they can be organized (e.g.: Text file having some random test which may be useful data, but are raw and have no metadata fields).

This paper will puts forward a mechanism for each of the above mentioned data structure. That can be incorporated into one storage engine. The query languages that will be used for querying these varying type of data are out of the scope of this paper. This paper focuses on how these data will be stored and processed by the storage engine.

The rest of the paper is organized as follows. In section 2 we describe how the proposed system will work generically based on the data structures. In section 3, 4 and 5 we describe about how the structured, semi-structured and un-structured data will be handled by the proposed system respectively. In section 6 we discuss conclusions and future work.

II. AN OVERVIEW OF THE WORKING OF THE SYSTEM

This proposed system is based on the observation and study of the nature and structure of data broadly classified into the three types mentioned above. However the system handles each type of data separately. As of now the system does not make a judgement on what sort of data is provided it. And it is the responsibility of the upper layer software modules to specify the structure of data.

In this project, like most other DBMS systems, the storage engine will be the logical unit (Software component) which will handle and manage files. So basically everything will be stored in files. However the way they are stored into one or many files, differ based of their structure. Furthermore, how indexes will be applied to the data or whether or not indexes will be applied at all. Whether the data will be stored in binary format or in the text format in the files. Are the prime concern of this proposed system.

So, from the above description it is clear that the data structure is a logical abstraction (way to categorize data). And that the storage engine is the actual software entity that will implement this using various file system.

Now the flow of the system is as shown in the figure 1. Storage engine will take the parsed and validated query from the upper layer software module and then based on the structure of the data invokes the appropriate set of functions to carry out the task. Basically this task can be any operation that can be performed on the data. Various file structures will be used to store these data (Detailed explanation further in this paper). Performing data validation and applying constrains, if any, is the task that is supposed to be taken care of by the upper layers of the overall software. However integrity of the file system and the data in the file system will be maintained by the storage engine.

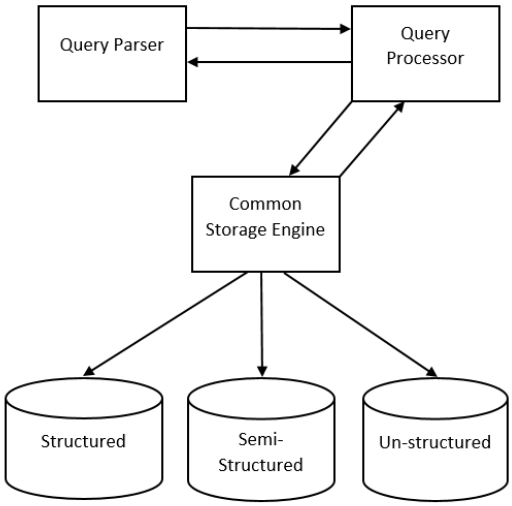

Fig. 1: Storage engine structure

Now here arise a question: Whether there can exist any relation or linking between two or more data structure in the database. The answer would be, as of now one single database cannot have heterogeneous data structures. Because the correlation among the three different structures of data is still a topic of discussion and research by many organizations and lacks standardization.

III. HANDLING OF STRUCTURED DATA

Almost all DBMS software available in the market can handle structured data. This proposed system can does the job in similar way as other in this case. Since structured data are the most easiest to categorize and have an ample number of Meta data fields. They are basically stored as binary data in files. This not only makes it easy for the system to process it. But also makes it easy to be indexed.

Operations like retrieval, manipulation and deletion of data is faster than the other types. And also reorganizing and restructuring of the files in which data is stored is quite easy. Section 3.1 and 3.2 gives some insight of how structured data is stored and indexed into binary files.

A. Storage of Structured data into binary files

The mechanism is quite simple. Each entity is treated as a row of a table. Each row is stored in pages and in turn these pages are stored into files. Pages are typically of 8kb in size. And each page may contain multiple rows if the size of the rows are smaller than 8kb. However if the size of the row is larger than 8kb the row can span into multiple pages. Figure 2, Illustrates multiple rows are stored in one page and that their starting byte address is stored in the offset table which is at tailing side of the page. Each page is organized in B+ tree page directory in the file as shown in figure 3.

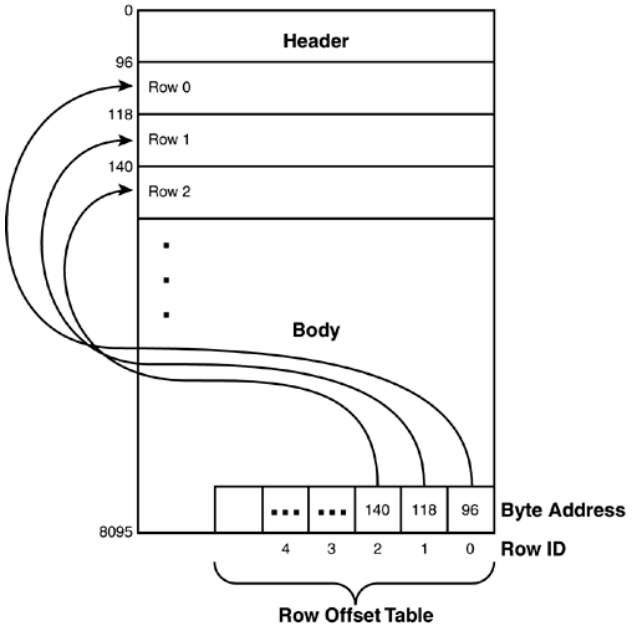

Fig. 2: Page structure

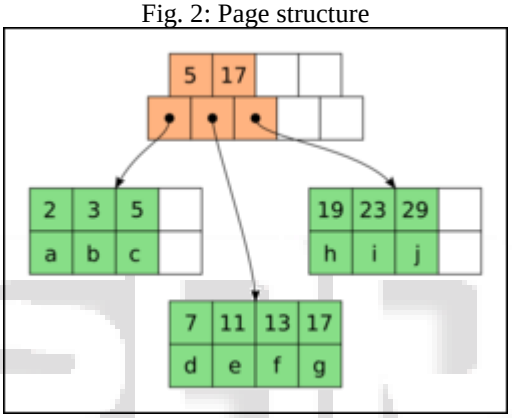

Fig. 3: Organization of pages in files.

Each page is stored as well as indexed using B+ tree structure in the file. The ordering of the pages in the file may vary based on the type of index applied to particular file.

B. Indexing of Structured data in files

Indexes form the base for the faster access of data in a storage system. And binary files are easy to be indexed. The proposed system uses B+ tree structure to store as well as index the data. In this structure all leaf nodes contains actual data. And all other nodes between the leaf node and the root node contains pointers to the appropriate pages to be traversed after them. As shown in figure 3, the page illustrated in orange colour represents a root node that contains pointers to the pages shown in green colour. The pages shown in green colour contains the data.

If the file is structured using clustered index all the pages are organised following a particular order. And if an un-clustered indexing mechanism is used, the pages are stored in random order however appropriate pointers are maintained for each page in the index. These indexes are also classified as primary and secondary indexes. A file can have more than one index structure applied to it. But the golden rule that holds true only one clustered index (i.e. only one column of the table can have a clustered index) can be applied.

IV. HANDLING OF SEMI-STRUCTURED DATA

As mentioned earlier in this paper, Semi-structured data is a form of structured data that does not conform to the formal structure of data models associated with relational databases or other forms of data tables, but nonetheless contains tags or other markers to separate semantic elements and enforce hierarchies of records and fields within the data. Therefore, it is also known as self-describing structure.

The two preferred ways to store these type of data are XML (Extensible Mark-up Language) and JSON (JavaScript Object Notation). XML being the most widely used standard because of its long existence. JSON is quite new however it has got wide industry acceptance. In our approach we will be using JSON structure to store semi-structured data into files. As many languages now-a-days have built-in libraries to parse JSON and because in JSON gives more variation and control over the data representation.

```
{
  "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
  "age": 25,
  "height_cm": 167.6,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    }
  ],
  "children": [],
  "spouse": null
}
```

Fig. 4: JSON structure example

A. *Storage of Semi-Structured data into files*

Data is stored into files with ".json" extension which happens to be the standard file format for storing JSON data. Logically data is stored in text format. It is the structure of data that makes the difference. Quite literally JSON is so flexible that light weight that it is fast and rapidly replacing XML.

Each entity in the database is stored as a single JSON object in the file. Since the JSON is self-descriptive, i.e. every individual field has its own descriptive label. Hence the dependency over a Meta data is removed. No description file is needed for this type of structure. There may or may not exist similarities between data related to different entities.

B. *Indexing of Semi-Structured data*

Indexing of semi-structured data is way different from structured data. Since the file is not in binary format the actual pointer to the first byte of data cannot be extracted. Hence the idea is to store only a fixed and limited number of entities in one file and breaking the entire dataset into multiple files. And indexing particular fields into a separate binary file which will be used to traverse. Once a match is found the relevant file which contains the record is hit.

Since every file contains only a fixed number of records it is not time consuming to read each record one by one. And hence it is quite efficient. The only drawback being that the number of files increases as and when data increases.

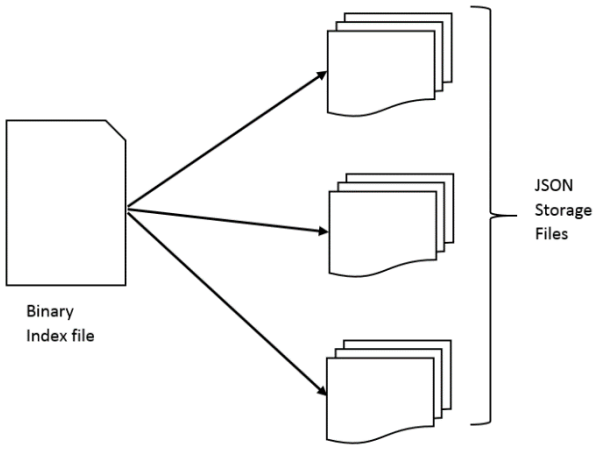

Fig. 5: JSON file storage structure

V. HANDLING OF UN-STRUCTURED DATA

Un-structured data falls into a category of information which either does not have a pre-defined data model or is not organized in a pre-defined manner. Un-structured data is typically text based data, but may contain data such as dates, numbers and facts as well.

The only preferred way to store this type of data is to store it in chunks into files with standard character encoding. It is almost impossible to index this type of data based fiends and labels because they don't have any. However the technique proposed below uses the traditional method but with a twist to achieve the two basic functionality needed for a database management system.

A. *Storage of Un-Structured data into files*

Data is stored into files with standard text based encoding each chunk of data is stored in a separate file. With a name that corresponds to an entry in the indexing file (explained below). Each file related to one particular data group are grouped together into one single folder.

One single file can be max of 4 GB in size to provide support for some older file systems. However that limit is too high for a single text chunk of data, but if at all the data size increases that limit. File is split into multiple files.

B. *Indexing of Un-Structured data*

Indexing of un-structured data is really a challenge when it comes to application like a database management system. However the proposed technique uses the keyword based indexing technique which happens to be efficient as well as time tested. This not only solves the problem, but also helps to categorise the files for web based search and indexing.

The technique is quite simple. It is the same technique used by all major search engine i.e. back of the book indexing technique. Where each file is scanned for keywords when it is added to the repository. These

keywords can be any words except for the most commonly used words in English or any other literature. These words are indexed in a structured manner into index files. When a retrieval search is made on the data. The storage engine searches for the keywords specified in the query in the index file and all files that qualifies the search is returned as the result of the retrieval result.

The indexing is done in two levels. The master file forms the level 1 and contains all the initial of the keywords and the links to corresponding secondary index files. Secondary index files which returns links for the file which contain the keywords.

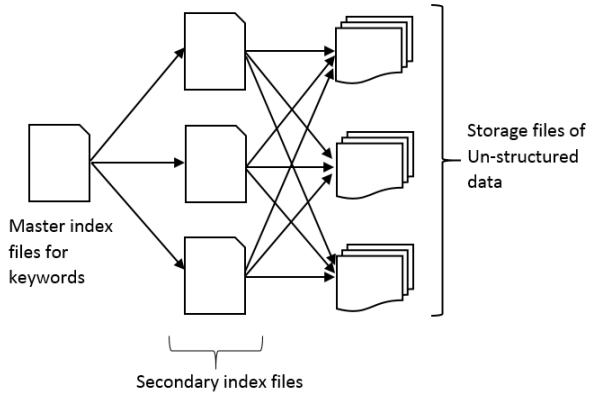

Fig. 6: Un-structured file storage system

VI. CONCLUSIONS AND FUTURE WORK

The idea proposed in this paper handles and manages data that can handle almost any sort of data irrespective of the structure. With a concept that is best suited for the cloud computing and cloud development environment. Thought each type of data is handled separately the application acts as a single point of interaction for all that is stored in the database.

Later development in this project would include a standard and common language structure for querying the database. And developing an inter-relation between the heterogeneous types of data. Adding distributed capabilities that allows scalability so as to make it handle robust databases. Developing standard interface for HTTPS to make it more compliant with web.

REFERENCES

[1] Raghu Ramakrishnan, Johannes Gehrke, *Database Management System* 3rd ed, *International Edition*, TATA McGrawHill 2003.
[2] Abraham Silberschatz, Henry F. Korth, S. Sudarshan, *Database System Concepts*, 6th ed, TATA McGrawHill 2001.
[3] Wikipedia documentation on semi-structured data [Online]. Available: *http://en.wikipedia.org/wiki/Semi-structured_data*
[4] Oracle documentation on JSON standards [Online]. Available: *https://docs.oracle.com/database/121/ADXDB/json.htm#ADXDB6247*
[5] International instruments, common data storage and File I/O approaches [Online]. Available: *http://www.ni.com/white-paper/9630/en/*

