SQL Beyond Structure: Text, Documents and Key-Value Pairs

Daniel Tahara
Yale University and Hadapt
daniel.tahara@yale.edu

Daniel Abadi
Yale University and Hadapt
dna@cs.yale.edu

ABSTRACT

Despite, or perhaps because of, the bevy of data formats used in modern applications, the development community has yet to settle on a standard query interface for analyzing that data in an efficient manner. As a result, they are forced to rely on complicated scripting and ETL in order to analyze their data, significantly increasing their overall 'time to insight.'

Meanwhile, SQL remains entrenched among the skillsets of analysts and database managers, with conventional wisdom saying that its semantics are incompatibile with the new, relaxed data formats. However, with our 'Flexible Schema and Multi-structured Tables' approach, we show that it is possible to unify structured, semi-structured, and fully unstructured data as part of a single analytics system. Our approach defines an extended relational abstraction (*Multi-structured Tables*) that maps arbitrarily structured data into a schema-ed, relational view (*Flexible Schema*). With these two components, we can then provide a storage backend in order to supply a performant, fully SQL-compliant analytics backend.

1. INTRODUCTION

The traditional solution to the problem of analyzing multi-structured data has been either a) to store the data using different backends depending on their level structure (or lack thereof) and analyze them with complicated scripts or b) to use ETL tools to store the data in a single backend (and therefore offer a single interface) but give up SQL compatibility. MongoDB, for example, offers a connector that allows application developers to synchronize or migrate between storage backends [2]. Despite the increased attention toward compability with existing analytics standards, however, the impetus for managing the data remains on the developer.

There has been a recent trend toward offering a single software solution to querying multi-structured data. Among these systems, IBM DB2 recently added a backend storage component for managing JSON data alongside traditional relational data, but it does not allow that data to be queried using SQL (it uses the MongoDB query language) [1]. Even when the system does attempt to preserve some sort of relational abstraction, the abstraction is often incomplete and requires that the system's query interface breaks from standard SQL. In particular, with its version 9.3 release, Postgres added support for a native datatype that allows database managers to create and store attributes of JSON text. The release also adds corresponding operators for dereferencing JSON keys and otherwise manipulating the objects [3]. Although Postgres's efforts have created one of the tightest SQL/NoSQL integration of systems we have seen[1], it suffers from the two problems listed above—it extends SQL syntax and does not treat JSON keys as first-class relational attributes. Although neither are inherently problematic, they do limit the ability to leverage third party analytics tools and cloud the relational abstraction presented by an RDBMS.

The system we have developed in our joint work at Yale and Hadapt therefore differs from previous work in that it provides a single system for storing multi-structured data and querying it using a standard SQL interface, without the need for complex scripting or ETL. What follows is a motivating example to clarify our target use case (Section 2), and an overview of our system design (Section 3).

2. MOTIVATING EXAMPLE

Consider a naïve solution to offering a SQL interface to various forms of multi-structured data, which simply merges structured data with semi-structured and unstructured data by storing each in their own respective text columns. Figure 1 represents the result for a sample clickstream. Although the structured components may be easily queried using standard SQL constructs querying the attributes containing JSON or unstructured text require complicated regular expressions[2], a resulting query over that data, a sample of which is shown below, is not only incredibly difficult to understand but also might not even produce a correct result when faced with non-regular data formats such as XML.

```
SELECT user,
  regex_value(json,
    '"productName":\s*"([^"]*?)"', 1, 'i')
FROM actions
WHERE regex_match(json, '"action":\s*"view"', 'i')
```

[1]Teradata will be releasing a similar feature later this year. See: http://www.dbms2.com/2013/10/24/json-in-teradata.
[2]Note: regular expressions are technically not part of SQL standard, but most popular RDBMSs support them.

date	user	browser	json	product_comment
2/12/2013	Sam	Firefox	{"eventID": 1, "action":"view", "productName": "door", "price": 12.50,"color":"green", "tags": ["home", "green"]}	I really need this door.
2/13/2013	Jim	Chrome	{"eventID": 2, "action":"view", "productName": "window", "color":"orange, "price": 52.50,"tags": ["home", "orange"]}	The prices for windows appear to be getting higher.
2/14/2013	Bob	IE	{"eventID": 3, "action": "purchase","price": 57.12}	

Figure 1: Merged clickstream

date	user	browser	product name	price	tags	color	action	product comment
2/12/2013	Sam	Firefox	door	12.50	home green	green	view	I really need this door.
2/13/2013	Jim	Chrome	window	52.50	home orange	orange	view	The prices for windows appear to be going higher.
2/14/2013	Bob	IE		57.12			purchase	

Figure 2: Virtual view of data from Figure 1

```
  AND product_comment LIKE '%higher%';
```

An ideal abstraction (both fully SQL-compatible and entirely transparent to the user), on the other hand, would represent each of the attributes appearing in the semi-structured data as its own attribute in the table. This representation, referred to as the universal relational model [4, 5], describes each record as as an n-tuple of attributes, with the tuple format (and correspondingly, the columns in the virtual, tabular view) described by the set of attributes existing across all items in the entire dataset. When a datum does not contain a certain attribute in its original representation, it simply represents it with a null in the corresponding column. The resulting view would look something like Figure 2, and the query presented above would be reduced to:

```
SELECT user, productName
FROM actions
WHERE action = 'view'
  AND product_comment LIKE '%higher%';
```

With a cleaner relational abstraction, the query predicates and projected columns are immediately apparent. Furthermore, integration with external analysis tools is immediately simpler because those tools can remain agnostic to the underlying format of the semistructured and unstructured components of the data.

3. SYSTEM OVERVIEW

Maintaining a universal relational representation is fairly straightforward when the structure of the application data is fixed because the columns can be defined prior to data storage. However, flexible, self-describing data formats such as JSON and XML, present a number of challenges when attempting to coerce data into a relational form—attributes can appear and disappear, and attributes originally corresponding to values of a single type might change type. A fully faithful physical representation of a universal relation is therefore infeasible[3].

Our solution to the problem combines a traditional RDBMS, a user-supplied parsing function, and an external text index. A load, which is straightforward for any structured components, applies the parsing function to the semi-structured object (which can be anything from JSON to simple key-value data to HTTP query strings). The system then takes the internal representation of the data (at this point a set of *attribute, value* pairs) and stores the values in the text index segmented by attribute. The object itself is stored as an RDBMS binary or text column using a custom, serialized format, which allows our system to store objects that contain previously non-existent attributes. Similarly, any unstructured (i.e. text) data is indexed in the text index and stored in an independent RDBMS column.

Although the resulting physical representation of the data is not unlike that of Figure 1, our system presents the ideal, user-facing abstraction of the data shown in Figure 2 through a process of query transformation. Projections over virtual columns (i.e. those that remain serialized in the physical storage layer) are performed by transforming the projection in the original query to a projection over the special serialized column and then extracting the desired column from the serialized datum after the RDBMS returns a result. Predicates over virtual columns are delegated to the text index, which returns a set of record identifiers corresponding to rows that satisfy the given predicates. The result set is then applied as a filter over the remaining query, which is delegated to the RDBMS.

[3]Most RDBMSs limit the number of attributes for a given relation because of the cost to maintaining wide tables.

4. FURTHER WORK

One drawback to keeping non-relational data in a serialized format is that the attributes are opaque to the RDBMS optimizer, and, if the deserialization happens in a layer apart from the RDBMS, then users cannot utilize B-tree indices on those attributes. A potential solution to this problem might involve periodic analysis of the dataset and user access patterns in order to selectively create physical columns for virtual attributes previously stored in the serialized format.

5. CONCLUSION

We have presented an overview of a major problem we have seen among developers of web applications, the simultaenous need to represent their application data with arbitrary levels of structure and to analyze it using a well-known, efficient interface. We offer our solution that combines a dynamic virtual view of the data stored in multi-structured RDBMS tables and an external text index with a query transformation process to provide developers with a performant SQL interface to their data.

6. REFERENCES

[1] Db2 json capabilities: http://www.ibm.com/developerworks/data/library/techarticle/dm-1306nosqlforjson1/index.html.

[2] Introducing mongo connector: http://blog.mongodb.org/post/29127828146/introducing-mongo-connector.

[3] What's new in postgresql 9.3: https://wiki.postgresql.org/wiki/what's_new_in_postgresql_9.3.

[4] R. Fagin, A. O. Mendelzon, and J. D. Ullman. A simplied universal relation assumption and its properties. *ACM Trans. Database Syst.*, 7(3):343–360, Sept. 1982.

[5] D. Maier, J. D. Ullman, and M. Y. Vardi. On the foundations of the universal relation model. *ACM Trans. Database Syst.*, 9(2):283–308, June 1984.