A Process for Inference of Columnar NoSQL Database Schemas

Angelo Augusto Frozza[1,2], Eduardo Dias Defreyn[1], Ronaldo dos Santos Mello[1]

[1]Universidade Federal de Santa Catarina (UFSC) – Florianópolis – SC, Brazil

[2]Instituto Federal Catarinense (IFC) – Camboriú, SC - Brazil

angelo.frozza@ifc.edu.br,eduardo_dududex@hotmail.com,r.mello@ufsc.br

Abstract. *Although NoSQL Databases do not require a schema a priori, to be aware of the database schema is essential for activities like data integration, data validation or data interoperability. This paper presents a process for inference of columnar NoSQL DB schemas. We validate the proposed process through a prototype tool that is able to extract schemas from the HBase columnar NoSQL database system. HBase was chosen as a case study because it is one of the most popular columnar NoSQL solutions. When compared to related work, we novel by proposing a simple solution for the inference of column data types for columnar NoSQL databases that store only byte arrays as column values, as well as a generated schema that follows the JSON Schema format.*

1. Introduction

In the current scenario of computer systems development, new applications have as a requirement the need for greater flexibility over the data representation, which may comprise complex nested structures and data structures *in memory*. In order to meet these needs, NoSQL databases (NoSQL DBs) have been proposed. They are classified into four data models: key-value, document, columnar and graph [Sadalage and Fowler 2013].

Unlike traditional relational DBs, NoSQL DBs are usually *schemaless*, *i.e.*, they do not require a previous schema or support a flexible schema, which facilitates data storage [Han et al. 2011, Sadalage and Fowler 2013]. Nevertheless, to be aware of the data schema is essential for processes such as data integration, data interoperability or data analysis. Although NoSQL DBs do not require an explicit schema, there is usually an implicit schema on each DB instance that is ruled by the application that accesses it [Ruiz et al. 2015]. However, to infer the data schema from the source code of the application is a complicated process. Another alternative is to analyse and get information from the DB catalog, which may also be a hard task.

In order to provide a less complex solution to this problem, this paper presents a process for columnar NoSQL DB schema inference, focusing on the HBase DB management system (DBMS), which can be adapted to other columnar NoSQL DBMSs. It also generates schemas in *JSON Schema* format. HBase particularly stores the data as byte arrays. It makes the column data types known only by the application that generated the data. So, the challenge is how to infer these data types. The process is materialized as a prototype tool called *HBase Schema Inference (HBaSI)*.

This paper is organized as follows. Section 2 introduces columnar NoSQL DBs and HBase. Section 3 details the proposed process and Section 4 shows the evaluation. Section 5 comments the related work and Section 6 presents the conclusion.

2. Columnar NoSQL Databases and HBase

A NoSQL DB can be defined as a distributed and scalable DB that normally does not have a fixed schema, avoids join operations, does not always have an SQL access interface and tends to be open source [Tudorica and Bucur 2011]. Columnar NoSQL DBs is one of the four main NoSQL DB data models. Figure 1 shows the structure of a columnar NoSQL DB. Each column is associated with a key-value pair. A set of columns defines a row, which is accessed by an identifier (*Row-Key*). Different rows can hold different columns, being suitable for the representation of heterogeneous data [Hewitt 2010]. Some columnar NoSQL DBMSs support the *supercolumn* concept, *i.e.*, columns composed of other columns. The most popular columnar DBMSs are HBase[1], Cassandra[2] and Hypertable[3].

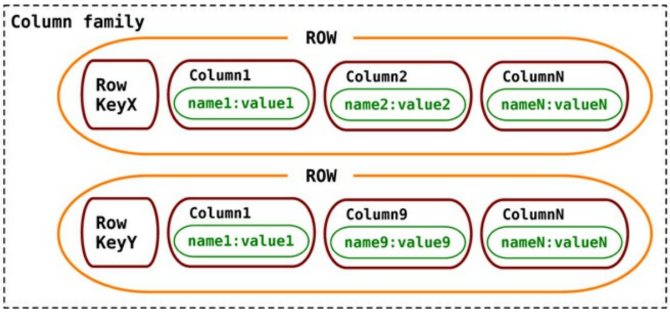

Figure 1. Example of a columnar NoSQL DB structure [Sadalage and Fowler 2013]

HBase is a columnar NoSQL DB, developed by *Apache Foundation*, that runs on the *Hadoop* distributed system[4]. It is a high performance and open-source DBMS with a flexible schema [Shriparv 2010]. Its data model can accommodate diverse semistructured data, which are converted into a *byte array* to facilitate data distribution. This physical representation makes hard to determine the data types of the columns without prior knowledge when accessing data.

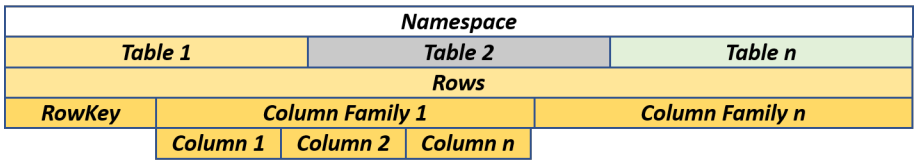

Figure 2. HBase hierarchical structure

The HBase data model, as well as other columnar NoSQL DBMSs, defines a hierarchical structure (see Figure 2)[5]. A DB instance is called *namespace*, which contains a set of *tables*. Each table, in turn, holds a set of *rows*, where each row has an identifier (*RowKey*) and at least one *column* that is always linked to a *column family*. A *column family* holds a set of required or optional *columns*, and different column family instances may hold a different number of columns (a so-called *column-oriented* data model).

[1] https://hbase.apache.org/
[2] https://cassandra.apache.org/
[3] https://hypertable.org/
[4] https://hbase.apache.org/book.html
[5] adapted from http://www.informit.com/articles/article.aspx?p=2253412

3. The Schema Inference Process

As stated before, we consider HBase as our case study as it is a widely used columnar NoSQL DBMS. Our schema inference process receives as input a *namespace* (see Section 2) and outputs a schema specification for the DB in *JSON Schema*, which is a common recommendation for data representation and exchange[6]. The basis of the schema inference process is the analysis of the DB hierarchical structure. A new *namespace* is created, containing a copy of the input without the data, *i.e.*, only the nested structure *table-column family-column* is maintained. The process goes through each table and, for each one, it analyses its columns, making the inference of the data type (see Figure 3). It is necessary to scroll all the column values to check for variations in the values of the data types stored into a column.

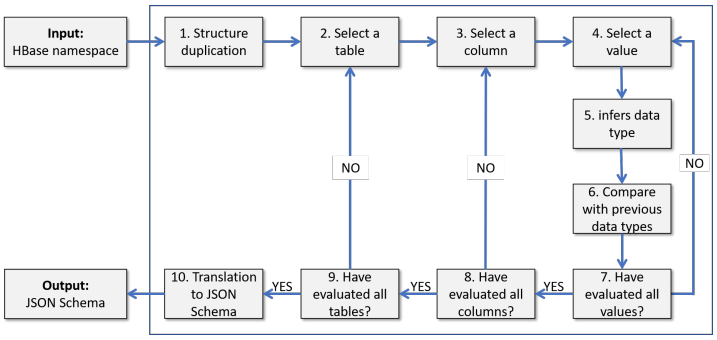

Figure 3. Organization of the JSON Schema document generated by the process

As mentioned earlier, HBase stores data in terms of byte arrays. This particularity represents the most difficult task of our process, *i.e.*, to infer a data type from a byte array, as the data type is only known by the application that uses the data [Shriparv 2010]. Thus, our strategy here is to develop a set of rules for the inference of most common data types from the binary content of the data item as follows:

- *byte*: a byte array of size one, accepting any amount;
- *boolean*: a byte array of size one, only with the values 0xFF or 0x00;
- *string*: a variable size byte array that follows the UTF8 binary standard;
- *short*: a byte array of size two, accepting any value;
- *char*: a byte array of size four, having its two least significant bytes represented in UNICODE;
- *float*: a byte array of size four, being limited by the values indicated in the IEEE 7541 standard (representation of binary numbers in floating-point);
- *integer*: a byte array of size four that can use up to 32 bits to represent a value;
- *double*: a byte array of size eight, being limited by the values indicated in the IEEE 7541 standard;
- *long*: a byte array of size eight that can use up to 64 bits to represent a value;
- *blob*: any entry that does not fit any of the previous rules.

After analyzing all the values in a column, the inferred data types are compared each other to define a consensual data type, *i.e.*, the more general data type that best represents all the values in the column. Once analyzed the DB structure and the columns of all tables, a *JSON Schema* document is created. Since most of the inferred data types

[6]https://json-schema.org

are not the same as the native data types of JSON Schema, the *definitions* section of the JSON Schema document is used to describe these data types, concerning extended data type definitions specified in a previous work of our research group [Frozza et al. 2018]. In turn, the *properties* section of the JSON Schema describes the DB hierarchical structure. Each column is a property of an object that represents a column family, each column family is a property of an object that represents a table, and so on.

4. Experimental Evaluation

A prototype tool called *HBaSI* was developed to validate our schema inference process over HBase. The Java language was used for the implementation, with the aid of the Apache Maven, HBase API and Gson libraries. As shown in Figure 4 (a), it is able to connect to HBase and list the available namespaces (left window), as well as the different schema versions generated for the selected *namespace* (right window). If the user decides to create or update a namespace schema, the tool generates a corresponding *JSON Schema* document that can be copied to the clipboard or exported as a JSON file (Figure 4 (b)).

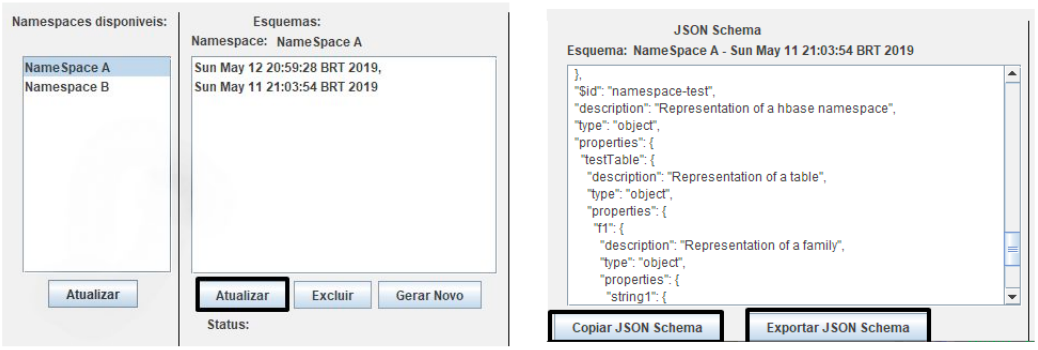

Figure 4. User interface of the prototype tool

For the experimental evaluation, we also implemented a data generator that creates random DB tables with different amounts of rows (ten, one hundred and one thousand). The data types supported by the data generator are the same mentioned in Section 3. In order to assess the quality of the schema inference, three possible results were considered: *a)* the data type does not match as expected (*incorrect*); *b)* the data type is as expected (*correct*); *c)* more than one possible (equivalent) data type is returned and the expected type is one of them (*partial*). The *partial* option deals with cases where more than one data type is inferred to the same column value, which usually occurs for numeric types.

The extracted schemas from the generated HBase DBs always got a 100% accuracy for the structural hierarchy, as expected, since this information can be obtained in a straightforward way from the DB namespace. This is not the case for column data type inference, as shown in Table 1. We see that *string*, *boolean*, *byte*, *short* and *blob* types were correctly identified. It happens because the set of rules that defines them (see Section 3) is restricted and does not contain many ranges in common with other data types.

On the other hand, *char*, *integer*, *float*, *long* and *double* data types, although it was possible to obtain the expected type, it was not possible to always define it accurately. It happens because of the limitations of the four bytes data types (*char*, *integer* and *float*)

Table 1. Result of the data type inference experiment

Data type	Incorrect	Correct	Partial
string	0%	100%	0%
boolean	0%	100%	0%
byte	0%	99%	1%
short	0%	99%	1%
blob	0%	100%	0%
char	0%	0%	100%
integer	0%	17%	83%
float	0%	0%	100%
long	0%	11%	89%
double	0%	0%	100%

which are not exclusive, as well as the eight bytes ones (*long* and *double*), which are also not. In fact, when we analyse data in binary format, the representation of one data type may be contained in the representation of another one. For example, a *char* content always has a binary encoding similar to an *integer* or *float* content. Additionally, some *integer* contents do not have a binary encoding that also represents a *char* or a *float* content. Therefore, in some cases, it is possible to correctly infer an *integer* data type. The same holds for some *long* contents. For these cases, our schema inference process considers the set of all possible data types.

We also analyse the spent time to generate the schemas. Although HBase can be used as a distributed DB, this experiment was limited to a monolithic DB instance, running on an Intel (R) Core (TM) i5-3337U CPU @ 1.80GHz machine, with 6 GB RAM. Table 2 shows the execution times for each input size. The last column shows how many Key-Value (K-V) pairs were analysed per second. It is possible to see that, with a small number of rows and K-V pairs, the time required to generate the schema (third column in Table 2) impacts the final performance. When the number of K-V pairs increases, a more linear relationship between processing time versus K-V pairs number is maintained.

Table 2. Result of schema inference processing time experiment

Rows	K-V pairs	HBaSI processing time (seconds)	K-V pairs per second
10	550	4	137
100	50500	203	248
1000	5005000	23037	229

As this is a preliminary assessment over a monolithic instance, it is not always possible to guarantee a good performance of our solution, as the processing time can be degraded in a distributed environment. However, this tendency for a linear complexity of our process w.r.t. the number of K-V pairs shows that it is a promising proposal.

5. Related Work

When migrating from a relational database to a NoSQL database, the biggest concern is how to represent the relationship data in the adopted NoSQL data model [Zhao et al. 2014]. This work, instead, aims to extract the most possible detailed schema from an HBase database, respecting its columnar data model.

Few approaches in the literature are related to our proposal. In [Ruiz et al. 2015] are applied MDE (Model-Driven Engineering) techniques to create NoSQL DB schemas that follow the aggregate-oriented data model, as is the case of the columnar NoSQL data model. However, it requires that the extracted data items have a unique identifier and a *type* property, and the considered data types must be the same as those available for JSON format. Another work proposes an approach for HBase data integration, in which schemas are extracted in the format of OWL (Ontology Web Language) ontologies [Kiran and Vijayakumar 2014]. Genetic algorithms are used to identify which record best represents a table's data set. Different from this work as well as the work of Ruiz, our schema inference proposal is less complex as it neither imposes constraints on the data items to be analysed nor requires the mapping to high level abstraction model as an ontology. Additionally, our extracted schema follows the *JSON Schema* recommendation.

6. Conclusion

This paper presents a process for inferring column-oriented NoSQL database schemas that deals with the problem of determining data types for columns that represented in a low level of abstraction as byte arrays. Our proposal differs from related work by introducing a simple inference process and outputting the extracted schema in JSON Schema format. We also developed a prototype tool that is able to generate DB schemas from the columnar NoSQL HBase DBMS. Preliminary experimental evaluation shows that our proposal is promising in terms of quality of the generated schemas as well as processing time. The tool source code is available at *UFSC Database Group repository*[7]. Future works include the improvement of the data type inference rules, the inference of optional and mandatory columns, performance analysis with large data volumes, and tool internationalization.

References

Frozza, A. A., Mello, R. d. S., and da Costa, F. d. S. (2018). An Approach for Schema Extraction of JSON and Extended JSON Document Collections. In *XIX IEEE Int. Conf. on Information Reuse and Integration*, pages 356–363.

Han, J., Haihong, E., Le, G., and Du, J. (2011). Survey on NoSQL database. In *VI International Conference on Pervasive Computing and Applications*, pages 363–366.

Hewitt, E. (2010). *Cassandra: The Definitive Guide*. O'Reilly Media.

Kiran, V. K. and Vijayakumar, R. (2014). Ontology-based data integration of NoSQL datastores. In *IX Int. Conf. on Industrial and Information Sysvems*, pages 1–6.

Ruiz, D. S., Morales, S. F., and Molina, J. G. (2015). Inferring Versioned Schemas from NoSQL Databases and its Applications. *LNCS*, 9381:467–480.

Sadalage, P. J. and Fowler, M. (2013). *NoSQL Distilled : A Brief Guide to the Emerging World of Polyglot Persistence*. Addison-Wesley.

Shriparv, S. (2010). *Learning HBase*. Packt Publishing.

Tudorica, B. G. and Bucur, C. A. (2011). A Comparison between Several NoSQL Databases with Comments and Notes. In *Proc. RoEduNet IEEE Intern. Conference*.

Zhao, G., Lin, Q., Li, L., and Li, Z. (2014). Schema conversion model of SQL database to NoSQL. In *Proc. 9th Intern. Conference 3PGCIC*, pages 355–362. IEEE.

[7]https://github.com/gDB-ufsc/HBaSI