Schema Application of the Read Modelling using Hadoop

Sladana Jankovic[1], Snezana Mladenovic[2], Stefan Zdravkovic[3], Ana Uzelac[4]
University of Belgrade, Vojvode Stepe 305
Belgrade
11000, Serbia
{s.jankovic@sf.bg.ac.rs} {snezanam@sf.bg.ac.rs} {s.zdravkovic@sf.bg.ac.rs} {ana.uzelac@sf.bg.ac.rs}

ABSTRACT: *The schema in the Relational Database Management Systems is used because of the unstructured data and the overhead of Extract, Transform and Load. Schema is important in the data analysis particularly in the tools such as Hadoop. Schema enables the loading of raw data and processing and ultimately help to interpret the data. Data sharing is important in the public data store. We in this paper applied the schema on read modelling and it will help to transfer big data and use source data. We conducted a case study in road traffic with the Hadoop Distributed File System, Apache data ware house and the query language.*

Keywords: Big Data Analytics, Data Sharing, HDFS, HiveQL

Received: 12 August 2020, Revised 18 November 2020, Accepted 29 November 2020

DOI: 10.6025/jdp/2021/11/1/1-6

1. Introduction

By analyzing signals obtained from machines and sensors, server logs and other new data sources, transport organizations can predict future events and become more proactive. Sensors and other intelligent devices can capture traffic data creating a large, ongoing flow of data. Such data require Big Data management systems for processing and reporting. As a result of the complexity, diversity and stochastic nature of transportation problems, the analytical toolbox required by transportation analyst must be broad. Big Data differs from other technologies is in terms of sophistication analysis it applies. Big Data analytics implies the process of discovering and extracting potentially useful information hidden in huge amounts of data (e.g. discovers unknown patterns and correlations). Big Data analytics uses advanced analytic techniques, such as machine learning, predictive analytics, data mining, text analytics, natural language processing and statistical analysis, against very large, diverse data sets. Data sets can be consisted of different data types such as structured/unstructured, streaming/batch, with different sizes (from terabytes to zettabytes). Also, while traditional analysis is often designed on the conditions that allow valid statistical inference about the characteristics of a population based on measurements on a small sample, Big Data-style analysis is built on the possibility to learn about systems by observing them in their entirety.

The intelligent transport systems have shown a rapid development over the last 15 years. This development has been accompanied with the need to test the systems that exist in the real world and collect data about their influence. Many previous and on-going transportation projects are not focused on the concept of data sharing and reusage data after the project finishes. The authors in [1] say: "More support services which sharing the data is needed to promote good research results." In this paper we propose schema on read modeling approach for Big Data analytics in traffic, which may be a modeling approach in the development of the information infrastructure for data sharing in the traffic domain. Schema on read is the revolutionary concept that we do not have to know what we're going to do with our data before we store it. Data of many types, sizes, shapes and structures can all be thrown into one of the Big Data storage systems. When we access the data, when we query it, then we determine the structure we want to use.

In the second section of this paper an overview of data modeling approaches is given. The third section of the paper is devoted to the description of our Big Data analytics modeling approach. We have implemented our approach in a case study in the traffic domain. In the fourth section of this paper our case study is presented. Finally, the conclusions we reached during the implementation of this modeling approach are given.

2. Data Modeling Approaches Overview

Relational Database Management Systems (RDBMSs) are widely used for to maintain data received in daily operations. Considering the data modeling of operational databases there are two main models: the Relational and the Entity- Relationship (ER) model. Systems using operational databases are designed to handle a high number of transactions that usually perform changes to the operational data [2]. These systems are called Online Transaction Processing (OLTP) systems.

The evolution of relational databases to decision support databases, referred as Data Warehouses (DWs), occurred with the need to store both operational and historical data, and to analyse that data in complex dashboards and reports. DWs are mainly used for OLAP (Online Analytical Processing) operations. Data modeling in DW consists of defining fact tables with several dimension tables, suggesting star or snowflake schema data models [3]. The most common data model used in DW is the OLAP cube, which offers a set of operations to analyze the cube model. Since data is conceptualized as a cube with hierarchical dimensions, its operations have familiar names when manipulating a cube, such as slice, dice, drill and pivot.

The volume of data has been exponentially increasing over the last years, namely due to the simultaneous growth of the number of sources (e.g. users, systems or sensors) that are continuously producing data. Therefore, there is a need to devise new data models and technologies that can handle such Big Data. NoSQL (Not Only SQL) is one of the most popular approaches to deal with this problem. NoSQL databases can be classified in four categories: Key-value stores, (2) Document-oriented databases, (3) Wide-column stores, and (4) Graph databases [4].

A Key-value store represents data as a collection of keyvalue pairs. Every key consists of a unique alphanumeric identifier that works like an index, and is used to access a corresponding value. Values can be simple text strings or more complex structures like arrays. The Key-value model can be extended to an ordered model whose keys are stored in lexicographical order. The fact of being a simple data model makes Key-value stores ideally suited to retrieve information in a very fast, available and scalable way.

Document-oriented databases were originally created to store traditional documents, like a text file or Microsoft Word document. However, their concept of document goes beyond that, and a document can be any kind of domain object. Documents contain encoded data in a standard format like XML, YAML, JSON or BSON (Binary JSON). Documents contain semi-structured data represented as name-value pairs, which can vary according to the row and can nest other documents. Unlike key-value stores, these systems support secondary indexes and allow fully searching either by keys or by values. Document databases are well suited for storing and managing huge collections of textual documents (e.g. text files or email messages), as well as semi-structured. MongoDB and CouchDB are two most popular Documentoriented database systems.

Wide-column stores (column-oriented databases) represent and manage data as sections of columns. Each section is composed of key-value pairs, where the keys are rows and the values are sets of columns, known as column families. Each row is identified by a primary key and can have column families different from the other rows. Each column of column family consists in a name-value pair. Column families can even be grouped in super column families. Wide-column stores are suited for scenarios like: (1) Distributed data storage; (2) Large-scale and batch-oriented data processing using the famous MapReduce method for tasks like sorting, parsing, querying or conversion and; (3) Exploratory and predictive analytics. Cassandra and Hadoop HBase are two popular frameworks of such data management systems [5].

Graph databases represent data as a network of nodes (representing the domain entities) that are connected by edges (representing the relationships among them) and are characterized by properties expressed as key-value pairs. Graph databases are quite useful when the focus is on exploring the relationships between data, such as traversing social networks, detecting patterns or infer recommendations. Neo4j and Allegro Graph are two examples of such systems.

Operational, decision support and Big Data approach to data management, from data modeling perspective, DBMSs perspective and data analytics perspective, were observed in Table 1.

Approach / Perspective	Operational	Decision Support	Big Data
Data Modeling Perspective	ER, Relational Models	Star Schema, OLAP Cube Models	Key-Value, Document, WideColumn, Graph
Database Management Systems Perspective	RDBMS	DW	Big Data- Based Systems
Data Analytics Perspective	OLTP	OLAP	Batchoriented processing, Streamprocessing, OLTP, Interactive ad-hoc queries

Table 1. Approaches and Perspectives of the Survey

3. Schema on Read Modeling Approach

As we can see in Table I, Big Data analytics can be categorized as follows: (1) Batch-oriented processing; (2) Stream processing; (3) OLTP and; (4) Interactive ad-hoc queries and analysis. Batch-oriented processing is a paradigm where a large volume of data is firstly stored and then analyzed, opposed to the streaming processing where data is continuously arriving in a stream, at a real-time, and is analyzed as soon as possible in order to derive approximate results [6]. *We propose schema on read modeling approach* for Big Data analytics based on batch-oriented processing.

Schema on write has been the standard for many years in relational databases. Before any data is written in the database, the structure of that data is strictly defined while metadata is stored and tracked. The schema – the columns, rows, tables and relationships are all defined first for the specific purpose that database will serve. Then the data is filled into its pre-defined positions. The data must all be cleansed, transformed and made to fit in that structure before it can be stored.

The emergence of Big Data technologies poses an alternative – a *schema on read* approach. Schema on read is simple up front: you just write the information to the data store. Unlike schema on write, which requires you to expend time and effort before loading the data, schema on read involves very little delay and you generally store the data at a raw level. In other words, you store what you get from the source systems, as it comes in from those systems, and define the schema at the time of data use (Figure 1).

Schema on read means you can first write your data and figure out how you want to organize it later. So why do it that way? As we can see in Table 2, the key drivers are flexibility and reuse of raw/atomic data. Exactly these characteristics make schema on read approach appropriate in scenarios of sharing data of public interest. In the next section we present the implementation of this approach to share information of public interest in the field of transport, between the various stakeholders.

4. Case Study

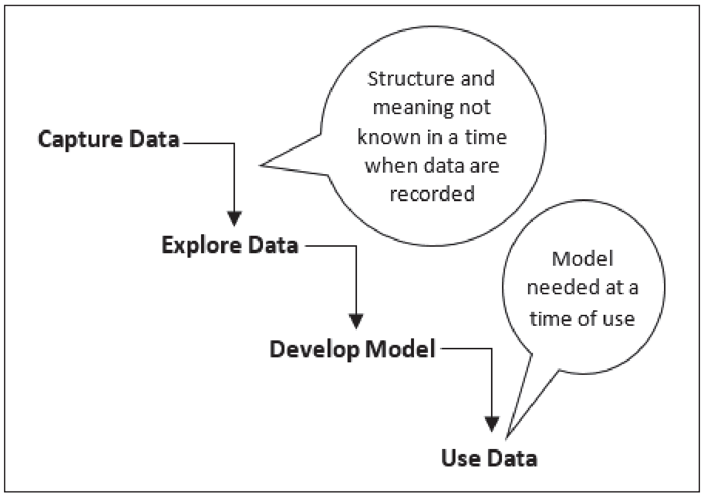
Capture Data
Explore Data
Develop Model
Use Data
Structure and meaning not known in a time when data are recorded
Model needed at a time of use

Figure 1. Schema on read modeling approach

Advantages	• Gives you massive flexibility over how the data can be consumed. • Your raw/atomic data can be stored for reference and consumption years into the future. • The approach promotes experimentation, since the cost of getting it "wrong" is so low. • Helps to speed the time between data generation and availability. • Gives you flexibility to store unstructured, semistructured, and/or loosely or unorganized data.
Disadvantages	• Since the data is not subjected to ETL (Extract, Transform, Load) and data cleansing processes, nor does it pass through any validation, that data may be riddled with missing or invalid data, duplicates, etc... • The SQL queries tend to be very complex. They take time to write, and time to execute. • Can be "expensive" in terms of computing resources. • The data is not self-documenting (i.e., you cannot look at a schema to figure out what the data are).

Table 2. Schema on Read Modeling Approach

Our implementation of schema on read modeling approach was realized through a case study of the Big Data analysis of traffic data. We analyzed traffic data from ten locations on the state roads and streets in the town of Novi Sad, Serbia, which the automatic traffic counters generated during the 2015. The Apache Hadoop platform was chosen to store and process the data. Our application of schema on read approach is based on *Hadoop Distributed File System (HDFS)*, *Apache Hive*™ data warehouse software, and *Hive Query Language* (HiveQL or just HQL).

HDFS represents a distributed Java-based file system designed to store very large files with streaming data access patterns that run on clusters of commodity hardware. "Very large" in this context means files sizing hundreds of megabytes, gigabytes, or terabytes. HDFS is based on the idea that the most efficient data processing pattern is a write-once, read-many-times pattern. Hadoop moves computing processes to the data on HDFS and not the other way around. Applications that require low-latency access to data, in the tens of milliseconds range, will not work well with HDFS. HDFS is optimized for delivering a high throughput of data, and this may be at the expense of latency. Some common storage formats for Hadoop include: plain text storage (e.g. CSV, TSV, TXT files), sequence files, Avro files, columnar file formats (e.g. Parquet, RC Files, ORC Files).

The Apache Hive™ data warehouse software facilitates reading, writing, and managing large data sets residing in distributed storage using SQL-like query language [7]. Hive provides an SQL dialect, called HiveQL for querying data stored in a Hadoop cluster. Hive translates most queries to MapReduce jobs, thereby exploiting the scalability of Hadoop, while presenting a familiar SQL abstraction. Hive is not designed for online transaction processing. It is commonly used for traditional data warehousing tasks where relatively static data is analyzed, fast response times are not required, and when the data is not changing rapidly. When you write data to a traditional database, the database has total control over the storage. The database is the "gatekeeper." An important implication of this control is that the database can enforce the schema as data is written. On the contrary, Hive does not have has any control over the integrity of the files used for storage and whether or not their contents are consistent with the table schema. So what if the schema does not match the file contents? Hive does the best it can to read the data. You will get lots of null values if there are not enough fields in each record to match the schema. If some fields are numbers and Hive encounters nonnumeric strings, it will return nulls for those fields. Therefore, Hive can only enforce queries on read. This is called schema on read.

Our solution was implemented through the following phases:

1. To count the traffic at the specified locations, the automatic traffic counters of type QLTC-10C were used. Each counter, during the course of the day, "writes" the data into one text file, so that during one year each counter generates 365/366 files. For each vehicle registered by a counter, one record is created in the text file. Record in the text file contains the following information: index, date and time, direction, lane, vehicle class, vehicle category, vehicle speed and vehicle length. Size of a text file is determined by the volume of traffic in one day at the observed counting place. In our case study, each of the ten counters in 2015 generated 365 files and each text file kept between 4000 and 14000 records.

2. Using the *Apache Ambari* user interface, on the Hortonworks Sandbox, text files generated by ten automatic traffic counters were uploaded into the HDFS.

3. Based on the structure of uploaded text files the data model for input data was created. According to developed data model, with the help of HiveQL query language, a *Traffic Counting* Hive database was created.

4. Using HiveQL *LOAD DATA INPATH* queries the Hive database tables were "filled" with the data from the text files stored on the HDFS (Figure 2).

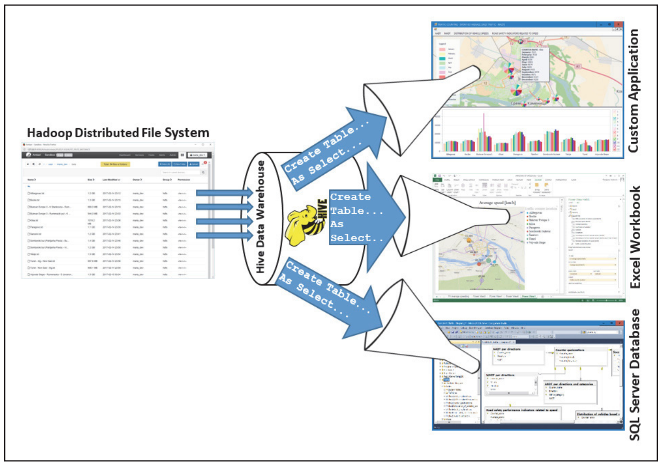
Hadoop Distributed File System
Hive Data Warehouse
Create Table... As Select...
Create Table... As Select...
Create Table... As Select...
Custom Application
Excel Workbook
SQL Server Database

Figure 2. Schema on read modeling approach implementation

5. We have recognized two possible types of the results of data analysis: indicators of traffic volume and structure and indicators of traffic safety in terms of vehicles speed. HiveQL has the well known powerful technique named *Create Table As Select* (CTAS). This type of HiveQL queries enable us to quickly derive Hive tables from other tables in order to build powerful schemas for Big Data analysis (Fig. 2). We carried out numerous CTAS and others HiveQL queries on the adoop *Traffic Counting* database resulting in traffic volume indicators and traffic safety indicators.

6. We have created three different "users" of the results of conducted Big Data analysis of traffic data: Windows geoapplication (developed in Microsoft Visual Studio 2015 and Visual Basic programming language), Microsoft Excel 2013 workbook and SQL Server 2012 database (Figure 2). For the geo-application schema on read approach enabled calculating of traffic volume indicators, as predefined attributes of the *OpenStreet Maps*. For the Excel Workbook this approach enabled the calculation of traffic safety indicators, as attributes of the *Bing Maps*. For the SQL Server database, this approach enabled the calculation of all traffic indicators according to its relational data model. Access to the *Traffic Counting* Hadoop database for all three users was enabled with the help of *Hortonworks Hive ODBC Driver*.

5. Conclusion

Schema on read approach can be seen as schema on demand. One area where we see the advantages far outweighing the drawbacks of schema on read strategies is in environments where multiple LOBs (Line of Business) try to hit the same source systems for their own copy of the data. The schema on read approach involves having a data "landing zone" where the raw or atomic data is written out. After getting the data once, all the LOB systems make their schema on read requests against the landing zone. This prevents the source systems from having to deal with all the LOB requests and provides a one-to-many approach of serving up data.

Similar studies, like Hadoop for exploratory analytics and Hadoop as a platform for transforming data or ETL, confirm that there is always a time cost to impose a schema on data. Hadoop provides an important advantage for exploratory BI in a single step from data load to query, which is not available in conventional RDBMS. The data-load-to-query in one step involves: 1. copy data into HDFS with ETL tool; 2. declare the query schema in Hive or Impala, which doesn't require data copying or re-loading, due to the schema-onread advantage of Hadoop compared with schema-onwrite constraint in RDBMS; 3. explore with SQL queries and launching BI tools for exploratory analytics. In schema on write strategies time cost is paid in the data loading stage. In schema on read strategies, that time cost is paid when we query the data.

Acknowledgement

This work has been partially supported by the Ministry of Education, Science and Technological Development of the Republic of Serbia under No. 036012. The data generated by automatic traffic counters provided by the company MHM - Project from Novi Sad.

References

[1] Gellerman, H., Svanberg, E., Barnard, Y. (2016). Data sharing of transport research data, *Transportation Research Procedia*, vol. 14, p. 2227 – 2236.

[2] Sarka, D., Radivojevic, M., Durkin, W. (2017). *SQL Server 2016 Developer's Guide*, Birmingham, Packt Publishing.

[3] Kimball, R., Ross, M. (2013). *The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling 3rd Edition*, Indianapolis, John Wiley & Sons.

[4] Ribeiro, A., Silva, A., da Silva, A.R. (2015). Data Modeling and Data Analytics: A Survey from a Big Data Perspective, *Journal of Software Engineering and Applications*, vol. 8, p. 617-634.

[5] Moniruzzaman, A. B. M., Hossain, S. A. (2013). NoSQL Database: New Era of Databases for Big data Analytics-Classification, Characteristics and Comparison, *International Journal of Database Theory and Application*, 6 (4) 1-14.

[6] Hu, H., Wen, Y., Chua, T. S., Li, X. (2014). Toward Scalable Systems for Big Data Analytics: A Technology Tutorial, IEEE Access, vol. 2, p. 652-687.

[7] Capriolo, E., Wampler, D., Rutherglen, J. (2012). *Programming Hive*, Sebastopol, O'Reilly Media.