# NoSQL AND TRADITIONAL DATABASE INTEGRATION: CASE STUDY PROJECT BDIC-DM

Ramiro Tadeu Wisnieski [1]

Data de entrega dos originais à redação em: 12/08/2016
e recebido para diagramação em: 20/04/2018

*This article describes the procedures involved in integrating a NoSQL database with a traditional one. These procedures were implemented in an academic project context of agile development, entitled Big Data, Internet of Things and Mobile Devices (BDIC-DM in Portuguese). This project was conceived in the first half of 2015 at Aeronautics Institute of Technology (ITA). As a requirement for the effectiveness of the mission, the implementation of an e-commerce system to manage transactions involving large volumes of data (Big Data), a number of different technologies were used. Among these tools the ones that stand out are those involving the generation, storage and consumption of large volumes of data. As a starting point for some features of Real Time Transactional Processing (OLTP - Online Transactional Processing) system, the traditional Database Management System (DBMS) MySQL was used along with the DBMS NoSQL Cassandra to store the portal purchase data. As for the batch data analysis (OLAP - Online Analytical Processing) Apache Hadoop Ecosystem was used. An infrastructure based on the Apache Sqoop tool allowed the export of data from traditional relational database to the HDFS(Hadoop File System).*

*Keywords: Big Data. Cassandra. Iintegration. MySQL. NoSQL.*

## I. Introduction

With the recent rise of Big Data worldwide, it became necessary to use specialized tools that facilitate the process of storage, consumption and analysis of huge amounts of data. The International Data Corporation (IDC) projects an average growth rate of annual Big Data usage of 53% in the volume of data between 2011 and 2016. According to Ashish Nadkarni, Research Director of storage systems at IDC, "Storage is one of the areas of greatest investment for big data and data analysis in this period and in the next decades". [1]Having the know-how of the technological apparatus allows several companies to extract insight from Big Data generating profit, maximizing results and reducing waste as well as finding opportunities that were not visible before the implementation of an OLAP system (Online Analytical Processing) or were not aware of the possibilities of managing large volumes of data transactions in real time via OLTP (Online Transaction Processing).

This work aims at showing a small portion of the numerous possible procedures within the world of Big Data, making integration between two different databases possible, respecting the characteristics of volume, speed, range, veracity and value inherent in Big Data [2].

## II. Context

A mission statement was proposed in order to guide the project development to be as analogous as possible to the real scenario of purchasing involving devices and its requirements.

### A. Traditional Database Concept

"A Database "is a collection of interrelated data, representing information about a specific domain" [1]. It represents part of the real world in an abstract way. When a database is handled by a software (Database Management System - DBMS), it is said that there is a database system. The standard relational database system makes use of relationships between different entities, attributes and records. If together with this relationship, the DBMS has its segmented data and associated by types and common fields, we say that this is a relational database with structured data. The database project takes place in two phases: Conceptual Modeling and Logical Design. The first is made user account regardless of the DBMS, there is only the definition of the data that exist in the database. The second depends on the type of DBMS, whether it is relational, object-oriented, hierarchical, among others. The relational DBMSs, in which data is organized in tables [3], are the most common ones.

The Structured Query Language (SQL), standardized by ANSI and ISO, is currently the most used for creation and manipulation of structured data in relational databases.

SQL is divided into four language subsets: DDL - Data Definition Language, DML - Data Manipulation Language, DCL - Data Control Language and TCL - Transaction Control Language.

### B. NoSQL Concept

NoSQL database is an alternative to relational database technology. The use of NoSQL database mitigates the scalability problem in traditional databases. Its origin occurred in a context of Big Data, in which large volumes of data require high processing power. The data models can be based on columns, documents, graphs, key-value among others.

### C. Apache Cassandra

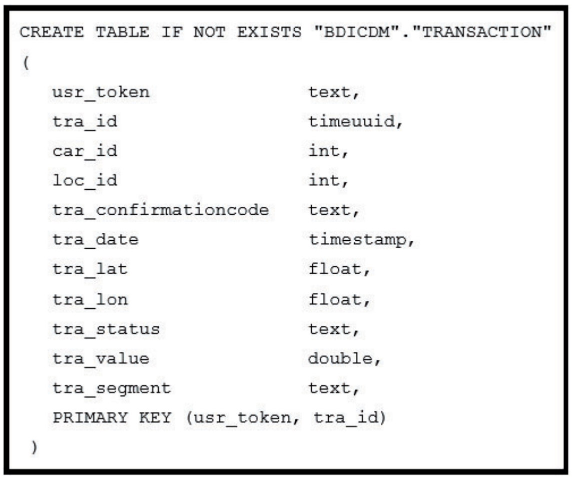
Big Apache Cassandra is a scalable open source NoSQL database that provides high availability and high performance. It was developed at Facebook and became an Apache project in 2010. In Cassandra, objects and data are manipulated via CQL (Cassandra Query Language), which is very similar to SQL language.

1 - Computing Division - Federal Institute of Education , Science and Technology São Paulo, IFSP - Itapetininga, Sao Paulo, Brazil. < ramirotadeu@gmail.com.br >.

## III. Development of the solution proposal

### A. *MySQL and Cassandra*

Initially the NoSQL Cassandra was the only database to be used in the BDIC-DM project. However, after a careful analysis done by the developers, the conclusion was that the use of a traditional relational DBMS was needed, since an e-commerce portal would be part of the project. The main purpose of the portal was to simulate the sale of some products, taking into consideration the payment by credit card. All tables except for the transaction table (figure 2), which was responsible for the financial transactions on the Project Data Model (figure 1), were allocated in the relational database [4] [5].



```
CREATE TABLE IF NOT EXISTS "BDICDM"."TRANSACTION"
(
   usr_token                  text,
   tra_id                     timeuuid,
   car_id                     int,
   loc_id                     int,
   tra_confirmationcode       text,
   tra_date                   timestamp,
   tra_lat                    float,
   tra_lon                    float,
   tra_status                 text,
   tra_value                  double,
   tra_segment                text,
   PRIMARY KEY (usr_token, tra_id)
 )
```

Figure 2 - Transaction Table

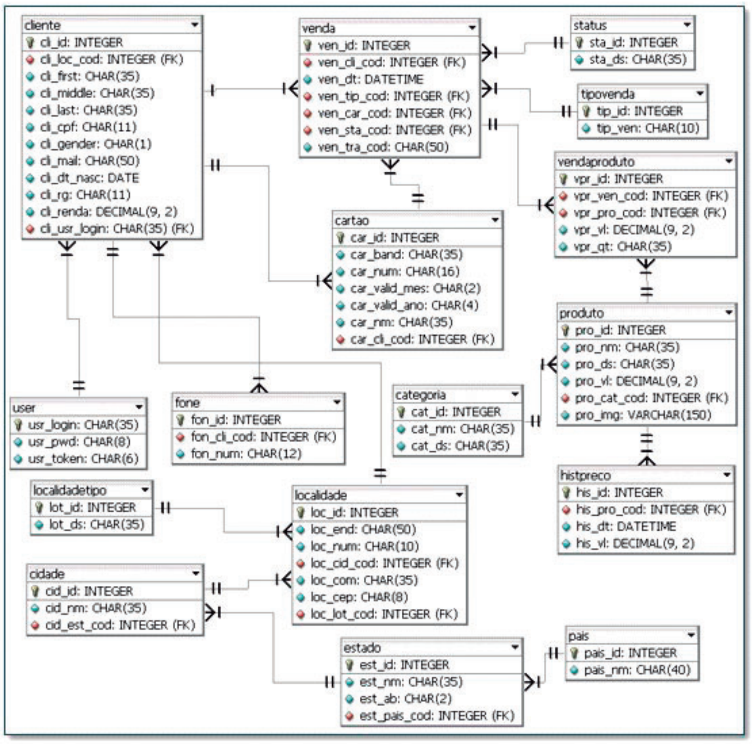


Figure 1 -

During phase 2 of the project, the financial transactions involving the purchases made by customers through the e-commerce portal, were stored in the NoSQL Cassandra DBMS. The application, which absorbed all the business rules, was in charge of the storage and retrieval of the data from the two project databases.

The main purpose of the architecture was to separate the financial transactions in the NoSQL database. Thus, it was possible to apply fraud detection algorithms with a very fast response time, increasing the search efficiency and significantly reducing the likelihood of the number of frauds.

### B. *Big Data and Hadoop*

The term Big Data usually refers to large datasets generated by companies and institutions, which often have some common features as the 4 V's: volume, variety, velocity and veracity. The source of this data is heterogeneous, for instance, sensors, supercomputers, security cameras, servers, social networks, among others. In Big Data, the analysis of these large data volumes allows that a new "insight" can be extracted. Information such as human behavioral patterns, climate change, security alerts or any type of strategic information that brings added value to the institution [6].

The increase of datasets in size and variety in the recent years associated with a global policy in searching for solutions with a good cost benefit took companies like Facebook and Yahoo to seek scalable solutions that could analyze terabytes or even petabytes of data.

This work uses Apache Hadoop as the OLAP analysis solution, thus isolating the processing of large volumes of transactions from the OLTP environment that uses Cassandra, optimized for high performance. The scalability offered by Hadoop combined with the ease of implementation of the jobs needed through the Apache Hive was crucial for this solution to be chosen [4].

## IV. Hadoop Ecosystem

### A. Hadoop

When datasets reach hundreds of gigabytes and the demand for data analysis of unstructured data such as logs, social networks and sensor data increases, there is the need for a horizontal scalable solution that provides the tools for the Data Scientist to extract information from these large volumes of data in a parallel manner [2].

Based on the Google MapReduce and Google File System technology, Apache Hadoop originated at Yahoo company. It is a framework written in Java, optimized for analysis of large volumes of structured and unstructured data using relatively inexpensive hardware. Hadoop is a powerful tool for sequence analysis, and it is not a replacement for relational databases, functioning as a complement that allows the management of different types of data in large volumes. In the paradigm used by Hadoop, the processing power is brought to the data, unlike traditional solutions that rely on large transfers so that data is moved, for instance, from a storage into the cluster processing nodes [7].

### B. HDFS (Hadoop Distributed File System)

One of the most important components of the solution is the distributed file system that provides high fault tolerance through data replication among the system nodes, enabling the growth of the capacity of the system, in a transparent manner, and high performance in the transmission of the data in the network. The HDFS works with large blocks of data (64MB default) due to the performance of Apache Hadoop to be superior when access to the data takes place sequentially, in other words, the number of "seek" operations are reduced. Very large files are ideal for Hadoop as it looks for the beginning of each block, keeping it sequentially from that point. An important feature of HDFS is the topology knowledge so the system assumes that the available bandwidth is greater when the nodes are located in the same rack, and the cost of access to such data increases with the distance among the racks since the network traffic must pass through several switches. The system uses its topology awareness in order to create a distribution policy that maximizes reliability and performance while accessing the data. [8].

In the case of BDIC-DM project, the HDFS provided an environment for storing and processing the assimilated data from the relational and NoSQL databases, providing significant gains in disk space and fault tolerance.

### C. MapReduce

Originated on the Google MapReduce programming model, MapReduce was created for operations with large amounts of data and became accessible to a larger number of users, hiding its complexity of developing parallel applications. The MapReduce is a core component of Apache Hadoop and can be used in a transparent manner through its eco-system tools like Apache Pig and Hive. Although the team had no specialists in implementing parallel algorithms or knowledge in low level implementation of MapReduce, it was possible to use this feature due to the choice of using Hadoop.

### D. Apache Hive

Although the MapReduce model facilitates the creation of applications that benefit from parallel computing environments, implementing this type of software is complex for several users. In 2007 Facebook identified the need to bring familiar concepts of databases  MapReduce, increasing the range of professionals who could benefit from the technology in the company. The Apache Hive offers a language similar to SQL called Hive Query Language (HQL) and abstracts the process of implementing the MapReduce. The Hive user writes a HQL query and submits it to the cluster through the command line or web interface. The application is responsible for creating and distributing all the necessary jobs and returns the results on the screen. The Apache Hive, however, is not a tool designed for real-time analysis, it is designed for high-volume batch analysis [9]. In BDIC-DM project Apache Hive was used in the analysis of financial transactions imported from Cassandra, avoiding the use of the main database for historical analyzes.

## V. Methodology

### A. Moving data into Hadoop

Moving large volumes of data into HDFS can be quite a challenge for some institutions due to limitations in their corporate networks and storage systems.

In this case study, there was a necessity to transfer data between a relational database (MySQL), a NoSQL database (Cassandra) and the HDFS so that it became possible to use the Hadoop Hive on the analysis of the dataset.

To solve the task transfer between MySQL and HDFS servers, Apache Sqoop was chosen as a tool for transferring data between relational databases and HDFS. Sqoop allows transfers to be executed in parallel and enables the execution of MapReduce jobs directly in the remote data. For example, this flexibility allows queries to be submitted to the relational database, filtering results at source, reducing network traffic and storage requirements in HDFS [10].

In the case of Cassandra, it was not identified at the time of testing a similar tool to Sqoop, so the native Cassandra command COPY TO CSV was used through a combination of Shell Script and CLASH (Cassandra Language Shell). The data was then transferred to HDFS with the use of Hadoop [11] put command.

In the first step of the data transfer to the HDFS, a temporary area was used to create the same tables from the database source.

### B. Denormalisation Dataset

Although the normalization of databases is a common and recommended practice in the case of relational databases this can be a problem for large data volumes. JOIN operations are very costly, so running datasets in the order of Terabytes and Petabytes causes a drop in the performance of queries on Apache Hive. In these cases the denormalization, which consists of basically gathering information scattered in multiple tables in a larger table, becomes a good practice in performance optimization [12]. One of the queries we tested on the project involved sales statistics by location. The modeling used in the MYSQL database was normalized as the data model (Figure 1). After the denormalization process of the four tables involving locality, a single table was generated.

The normalization was performed using a query on Apache Hive, which transferred the data stored in text file format of the staging area to a new production area. The file format for storing files in the production area was the ORC (Optimized Row Columnar), one of the latest advances of Apache Hive. The ORC format is most effective when it is used to storage and access to data, bringing several optimizations such as automatic indexing and the ability to identify the type of data stored. The format allows the use of data compression through ZLIB and SNAPPY libraries [12]. The migration of the text file format for ORC with SNAPPY provided a 70% reduction in the use of disk space (Figure 3).

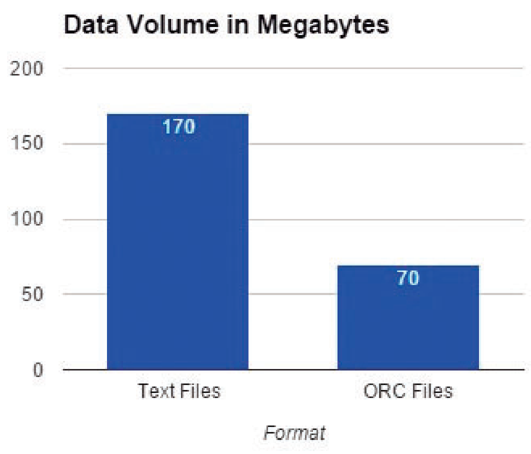
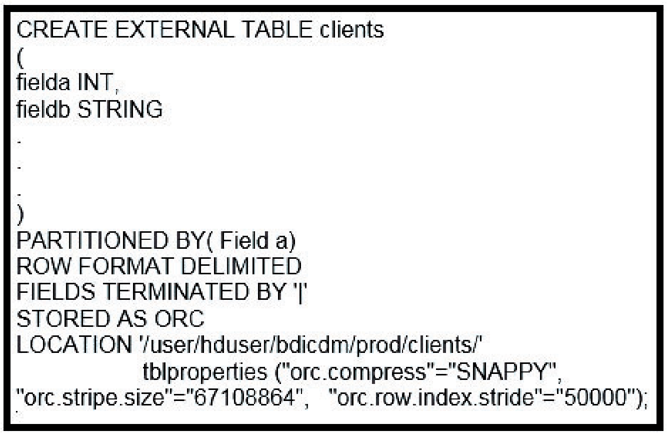


Figure 3 - Disk space occupied by Dataset

## C. Accessing Dataset

After the transfer of the Dataset to our production area , it was accessed through Apache Hive. In our research it was found that the use of external tables would be a better option. This technique indicates the path of the files in HDFS in the table creation DDL in the Hive. Thus, the system can access the data without the need for INSERT operations. The great advantage of external tables is the DDL independence in HIVE and data if a table is deleted by mistake for instance, the data remains intact. Figure 4 shows an example of the DDL used to create an external table in Hive:



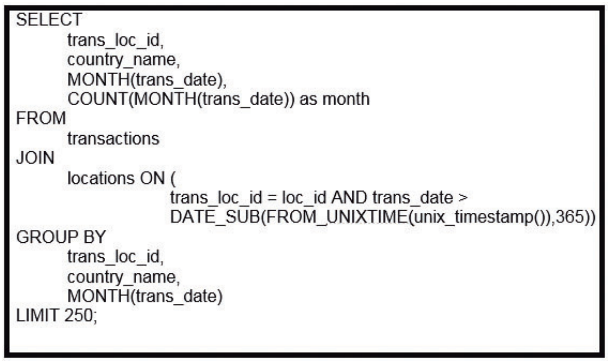
```
CREATE EXTERNAL TABLE clients
(
fielda INT,
fieldb STRING
.
.
.
)
PARTITIONED BY( Field a)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '|'
STORED AS ORC
LOCATION '/user/hduser/bdicdm/prod/clients/'
                tblproperties ("orc.compress"="SNAPPY",
"orc.stripe.size"="67108864",   "orc.row.index.stride"="50000");
```

Figure 4 - Example of a Hive DDL for an external table

The HQL query in figure 5 retrieves the amount of transactions divided by country and grouped by month for the past 365 days based on the system date. For the implementation of the project, nineteen queries were created with the aim of classifying transaction volumes, amount of sales in specific cities, holidays, and so forth.

## D. Performance Improvement

During the execution of the first jobs in BDIC-DM project, we noticed queries running for a very long time although the standard recommendations from the official documentation of the Apache Hive had been into account. While migration file format brought many



```
SELECT
        trans_loc_id,
        country_name,
        MONTH(trans_date),
        COUNT(MONTH(trans_date)) as month
FROM
        transactions
JOIN
        locations ON (
                        trans_loc_id = loc_id AND trans_date >
                        DATE_SUB(FROM_UNIXTIME(unix_timestamp()),365))
GROUP BY
        trans_loc_id,
        country_name,
        MONTH(trans_date)
LIMIT 250;
```

Figure 5 - Data being accessed through HQL query

benefits to the performance of Hadoop, some important best practices were identified that have significant impact on query performance.

One of the techniques implemented in our production server was the automatic table partitioning. This technique indicates to the HIVE one or more fields that the application should use to separate the data into HDFS directories. The transaction table contains the field data in "YYYY-MM-DD" format. The addition of the option "PARTITIONED BY (YEAR, MONTH, DAY)" to this DDL table, caused at the time of migration from the staging server HIVE, separated this table data into smaller files, now divided into subdirectories YEAR, MONTH and DAY (Figure 6). The great benefit of this technique for the large datasets is to run a particular query to a particular period or year, for example 2012, Hive will only access a narrow set of data [13].

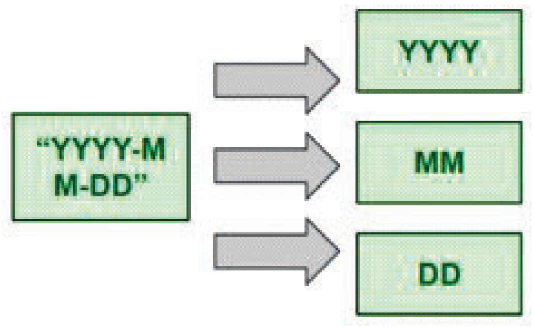


Figure 6 - Transaction table partitioning

## E. Testing Architecture

The HQL language is quite similar to traditional SQL, but it has some differences that caused some issues early in the project, since the team was using the language for the first time. As syntax errors and some unexpected results began to appear, validations of queries through unit tests became necessary before they were submitted to the production environment. Our team has implemented a test architecture based on HiveRunner framework, which is based on JUnit, well known in the java world [14].

Our model tests consisted of first use SQLite and .NET language to create an oracle then to validate a set of expected results through a set of synthetic data generated by the Generate Data tool [15] [16] [17]. From the data, expected test cases were implemented in HiveRunner then validating the queries used in the project.

The decision to use software testing techniques reduced the time that the team spent with refactoring consultations with different operation more than was expected, significantly increasing our productivity. Figure 7 contains a diagram of the test architecture described above.

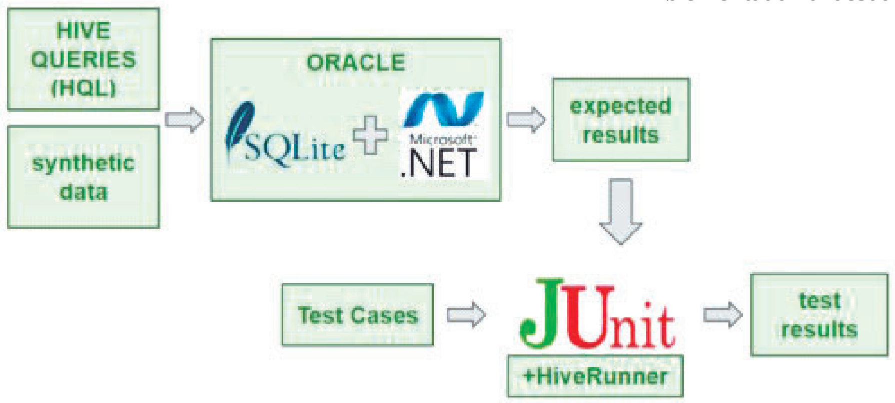


Figure 7 - Test architecture of BDIC-DM project

## VI. Results

During the BDIC-DM project, it was possible to apply the proposed architecture in figure 8 through a generated data assimilation infrastructure based on Apache Sqoop and features of Cassandra to export data in csv format. It was also possible to centralize varied data sources in a single environment that combines storage power (HDFS) and processing (MapReduce). By the OLTP and OLAP processing isolation, it was possible to identify significant performance gains in the system, since Cassandra was only used for online transactions.

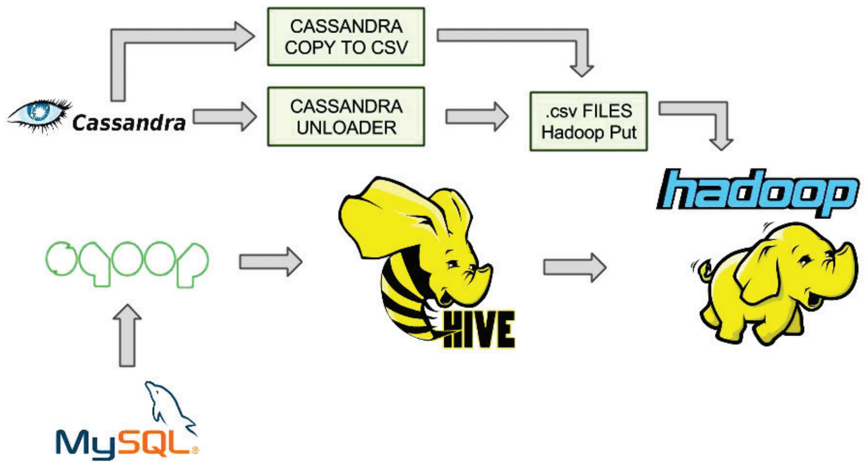


Figure 8 - Database Integration Architecture of BDIC-DM project

The use of HDFS provided a 70% reduction of disk space usage for storing data by using ORC format. Query performance gains were observed on Apache Hive after the dataset denormalization and employment of table partitioning. In a larger dataset the gains would probably be even more evident.

## VII. Conclusion

This article discusses the integration of technologies in the generation, storage and processing of large volumes of data, within a context of Interdisciplinary Problem Based Learning (IPBL), in the graduate program at the Brazilian Aeronautics Institute of Technology (Instituto Tecnológico de Aeronáutica - ITA).

The results were obtained thanks to the implementation of best development practice and to the commitment of the students that took part in the BDIC-DM project, who spent a great amount of technical effort to make the execution of this case study possible.

The proposed architecture at the beginning of the project proved to be capable of fulfilling its assigned mission.

The BDIC-DM project has provided a real experience of development and research of the topics discussed in this document, in addition to starting several new implementations and future studies, such as the incorporation of Apache Spark [18], enabling the execution of MapReduce jobs in memory.

It is possible to conclude that the integration of the Hadoop ecosystem technology, NoSQL and relational databases enables a wide range of implementations that bring important values such as processing and storage scalability, performance and flexibility.

## VIII. Acknowledgment

The author of this paper would like to thank the support and contribution of Brazilian Aeronautics Institute of Technology (ITA) for making the development of BDIC-DM project possible.

## REFERENCES

[1] GARTNER OFFICIAL WEBSITE. **Gartner Says 4.9 Billion Connected "Things" Will Be in Use in 2015**, November 11, 2014. Last access: 10/10/2015. Available at: < http://www.gartner.com/newsroom/id/2905717 >

[2] MARR, Bernard. **Big Data:** The 5 Vs Everyone Must Know. March 6, 2014. Last Access: 20/08/2015. Available at: < https://www.linkedin.com/pulse/20140306073407-64875646-big-data-the-5-vs-everyone-must-know >

[3] KORTH, H.F. e SILBERSCHATZ, A.; **Sistemas de Bancos de Dados**, Makron Books, 2a. edição revisada, 1994

[4] DATASTAX ACADEMY OFFICIAL WEBSITE. **NoSQL Databases Defined and Explained**. Last Access: 22/10/2015. Available at: < http://www.planetcassandra.org/what-is-nosql/ >.

[5] MySQL OFFICIAL WEBSITE. **MySQL Applier for Hadoop (happlier)**. Last access: 07/08/2014. Available at: < http://dev.mysql.com/tech-resources/articles/mysqlhadoop-applier.html >

[6] VENER, Jason. **Pro Hadoop**. New York. Apress. 2009

[7] BHOSALE, H. and GADEKAR, D.P. **A Review Paper on Big Data and Hadoop.** Last Access: 29/10/2015. Available at: < http://www.ijsrp.org/research-paper-1014/ijsrp-p34125.pdf >

[8] SHVACHKO, K. et all. **The Hadoop Distributed File System.** Last Access: 22/10/2015. Available at: < http://zoo.cs.yale.edu/classes/cs422/2014fa/readings/papers/shvachko10hdfs.pdf >

[9] THUSOO, Ashish. et all. – **A Petabyte Scale Data Warehouse Using Hadoop**. Last Access: 22/09/2015. Available at: < **http://infolab.stanford.edu/~ragho/hive-icde2010.pdf** >

[10] APACHE OFFICIAL WEBSITE. **Apache Sqoop.** Last Access: 25/10/2015. Available at: < http://sqoop.apache.org/ >

[11] APACHE OFFICIAL WEBSITE. **Cassandra CQL**. Last Access: 27/10/2015. Available at: < https://cassandra.apache.org/doc/cql/CQL.html >

[12] HUAI, Yin et all. **Major Technical Advancements in Apache Hive** - Access: 05/10/2015. Available at: < http://web.cse.ohio-state.edu/hpcs/WWW/HTML/publications/papers/TR-14-2.pdf >

[13] BARKHA Jain; KAKHAN Manish Km." **Query Optimization in Hive for Large Datasets" . Advances in Computer Science and Information Technology (ACSIT)**, India, 2015.

[14] GITHUB OFFICIAL WEBSITE. **HiveRunner:** klarna. Access: 05/10/2015. Available at: < https://github.com/klarna/HiveRunner >

[15] SQLITE OFFICIAL WEBSITE. Access: 06/10/2015. Available at: < https://www.sqlite.org/ >

[16] MICROSOFT OFFICIAL WEBSITE. Access: 07/10/2015. Available at: < http://www.microsoft.com/net >

[17] GENERATE DATA OFFICIAL WEBSITE. Access: 01/10/2015. Available at < http://www.generatedata.com >

[18] SPARK APACHE OFFICIAL WEBSITE. Access: 10/10/2015. Available at: < http://spark.apache.org/ >