Extracting rules to help in the conversion of a relational database to NoSQL (column-oriented) database technology

by

Rafik OUANOUKI

THESIS PRESENTED TO ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
IN PARTIAL FULFILLMENT FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY.
Ph.D.

MONTREAL, FEBRUARY 01, 2020

ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

ACKNOWLEDGMENTS

Praise be to God, who answers my prayers during times of doubt and who gave me the strength to keep pushing myself and never give up.

I dedicate this thesis to my mother, currently in a late phase of dementia, who has supported me since the beginning. There is nothing I can do as a son to repay her sacrifices, support and love. Mom, I know the sickness took your mind away but I would not have done it without you. I am simply the reflection of your endless love and education.

I am very grateful to my thesis director Dr. Alain April, who has been an amazing mentor, who has helped me overcome both academic and personal obstacles. My life has taken a massive turn since meeting Alain in the Quality Assurance course back in 2005. Thank you, Alain, for believing in me and for all your support.

I am also greatly thankful to Dr. Alain Abran and Dr. Jean-Marc Desharnais for all the time and support given to help me with my thesis.

I am deeply grateful to my wife Imane Harrache for her support and patient and continuous motivations in time of doubt.

Much thanks to Damien Sygusch, an old-time student roommate and long-time friend who gave a lot of his time helping me brainstorm, debate and browse new technologies. Your support has never failed me.

Last but not least, thanks to the members of my board of examiners for their time and effort reviewing my thesis and providing their feedback.

Extraction des règles d'aide à la conversion de base de données relationnelles vers une technologie de base de données NoSQL (orientées colonnes)

Rafik OUANOUKI

RESUMÉ

Cette thèse vise à aider les ingénieurs logiciels lors d'une première conversion de bases de données relationnelles vers une technologie de base de données émergente reconnue pour être mieux adaptée à l'infonuagique, c'est-à-dire la technologie NoSQL. Pour ce faire, une première activité de recherche vise à comprendre et à clarifier les notions de caractéristiques essentielles et optionnelles de la définition du terme infonuagique. Cette discussion est abordée sous l'angle de cette recherche (c'est-à-dire lors d'une conversion de base de données relationnelle vers le NoSQL). Par la suite, la contribution principale de cette recherche explore la possibilité d'extraire des règles de conversion de base de données relationnelle vers une base de données non-relationnelle (NoSQL), plus précisément la technologie HBase.

Deux expérimentations ont été effectuées afin d'identifier une liste de règles de conversion :

1. La première expérimentation consistait à demander à des participants de convertir un schéma relationnel particulier sans l'utilisation de règles de conversions. Plusieurs questionnaires ont étés complétés par les participants durant l'expérimentation. L'objectif principal de cette première expérimentation était d'évaluer s'il y avait un besoin d'avoir accès à des règles de conversions afin d'aider les ingénieurs logiciels lors de la conversion de schémas relationnels vers des schémas non-relationnels ;
2. La deuxième expérimentation consistait à composer des groupes parmi les participants. Chaque groupe a eu comme tâche d'effectuer la conversion d'une relation simple, c.à.d. une à une, vers un des schémas possibles en HBase. L'objectif était de tester les résultats d'une relation SQL vers tous les schémas possibles dans HBase, une base de données non-relationnelle, afin d'extraire les règles de conversions. Cette expérimentation a permis de démontrer pourquoi un schéma particulier est meilleur qu'un autre et a aussi permis l'extraction de règles de conversion basés sur des faits (ex. mesure de performance d'une requête sur chacun des schémas résultants).

Les contributions principales de cette thèse sont:

1. Une liste de règles de conversion d'un schéma relationnel vers un schéma non relationnel:
    a. Des règles portant sur la proximité des données ;
    b. Des règles portant sur les familles de colonnes ;
    c. Des règles portant sur la quantité de données ;
    d. Des règles portant sur les modèles d'accès.
2. Une clarification de la définition du terme infonuagique proposée à l'aide de notions de caractéristiques obligatoires et d'éléments facultatifs exclus de la définition.

Mots-clés : règles de conversion de bases de données relationnelle, NoSQL, définition infonuagique, mégadonnées.

Extracting rules to help in the conversion of a relational database to NoSQL (Column-oriented) database technology

Rafik OUANOUKI

ABSTRACT

This research aims to support software engineers in their first attempt at a conversion of relational databases to a database technology recognized as better suited to cloud computing (NoSQL), which in this particular case study is HBase. To do this, an initial research activity aims to understand and clarify the notions of essential and optional characteristics in the cloud computing definition. This discussion is considered by taking the perspective of this research (i.e. a database conversion from RDB to NoSQL). Subsequently, the main contribution of this research explores the possibility of extracting database conversion rules.

Two experiments were carried out to identify an initial list of conversion rules:

1. The first experiment consisted of asking participants (engineers from *École de technologie supérieure*) to convert a particular schema, without the use of a guide or conversion rules. Several questionnaires were completed by the participants during the experiment. The main goal of this experiment was to evaluate the need for conversion rules to help software engineers in the conversion of relational schemas to non-relational schemas.
2. The second experiment consisted of dividing the participants into sub-groups tasked to convert a single RDB relation (i.e. one-to-one) to a possible schema in HBase. The goal was to test the results of a basic relationship compared with all the schematic conversion possibilities in HBase, a particular non-relational database, and then extract the conversion rules. This experiment would show why one particular schema is better than another and would allow for the identification of conversion rules based on facts (e.g., performance measurement of a query on each of the resulting schemas).

The main contributions of this thesis are:

1. A list of rules for converting relational database schemas to the schema of a non-relational database:
    a. Rules on data proximity;
    b. Rules on column families;
    c. Rules on data quantity;
    d. Rules on access patterns.
2. Clarification to the term cloud computing is also proposed by introducing the notion of mandatory and optional characteristics excluded from the most popular definition.

Keywords: relational database conversion rules, NoSQL, cloud computing definition, database conversion, Big Data.

TABLE OF CONTENTS

LIST OF TABLES

Page

LIST OF FIGURES

LIST OF ABREVIATIONS

ACID	Atomicity, Consistency, Isolation, Durability
API	Application programming interface
CAAS	Communication as a Service
CER	Comité d'éthique de la recherche
CPU	Central Processing Unit
CRUD	create, read, update, and delete
DAAS	Database as a Service
DAL	Data Access Layer
DAPS	Distributed Application Platform
DB	Database
DBMS	Database Management System
DDI	Denormalization, Duplication, and Intelligent keys
DSL	Domain Specific Language
EAAS	Everything as a Service
ERP	Enterprise Resource Planning
ETL	Extract, Transform and Load
ÉTS	École de Technologie Supérieure
FK	Foreign Key
GFS	Google File System
GIMPS	Great Internet Mersenne Prime Search
HAAS	Hardware as a Service
Hadoop	High Availability Distributed Object-Oriented Platform
HBASE	Hadoop Database
HDFS	Hadoop Distributed File System
HTABLE	HBase Table
IAAS	Infrastructure as a Service
ISO	International Organization for Standardization
IT	Information Technology
JTC1	Joint technical committee 1
MAPREDUCE	Hadoop programming model

MSSQL	Microsoft SQL
NAS	Network-Attached Storage
NIST	National Institute of Standards and Technology
NoSQL	Not Only SQL
PAAS	Platform As A Service
PDA	Personal Digital Assistant
PK	Primary Key
QL	Query Language
RDBMS	Relational Database Management system
REST	Representational State Transfer
SAAS	Software as a Service
SC38	Sub Committee 38
SDO	Standards Development Organization
SQL	Structured Query Language
UML	Unified Modeling Language
UPS	Uninterruptible power supply
URL	Uniform Resource Locator
VOIP	Voice over IP
WG	Working Group

INTRODUCTION

Investment in information technology (IT) is considered a driver for firm performance and is covered in many firms' strategy (Teekasap, 2016). Nowadays, it is common for companies to invest large portions of their budgets on IT infrastructure, software and other IT services. According to Susilawati (Susilawati and Surendro, 2017), "in order to improve their performance, organizations are investing an ever-increasing amount of money in IT."

Cloud computing technology presents companies with an opportunity to limit the large upfront investments in software and main IT infrastructure. With cloud computing, companies are promised to be able to evolve their core business with a limited and progressive pay-per-use for IT infrastructure and software.

According to Pescholl (Pescholl, 2018), even if there are an insufficient quantitative studies, a useful argument can be made about the suitability and economic viability of using cloud computing. Companies today, whether small, medium or large, are generally looking for rapidly exploitable solutions and will therefore benefit from the reduced cost of IT infrastructure by using the 'pay-per-use' cloud approach. Cloud computing can reduce investing large amounts of the capital budget upfront to acquire equipment that are not always used at full capacity.

However, cloud computing continues its evolution (Dunno, 2019), a continuous evolution that is reflected in its ambiguous definition. This creates a lot of questioning and confusion about what it really is and how it can be used in the current IT market. As with every hyped technology, vendors hurry to promote cloud-based products without clearly explaining to their customers what this technology is all about (Plummer, 2009).

It is Google, the current world leader in cloud-based software services, with its search engine totalling more than 2.4 trillions searches a year as of October 2019 (Internet Live Stats, 2019), which revealed the use of a new, powerful and inexpensive technology enabling cloud computing to be effectively implemented on a very large scale with robust production characteristics. In 2003, Google presented its proprietary file system named GFS (Google File System) (Ghemawat, Gobioff and Leung, 2003), its distributed database named BIGTABLE (Fay et al., 2006) as well as the use of a new programming model and language for access to distributed data named MapReduce (J. Dean & Ghemawat, 2004). These technological innovations allowed Google to manipulate petabytes of data in both batch and real-time processing by operating reliable cloud services on a large number of inexpensive computers clustered together. This cloud computing architecture promotes parallel processing and has a mechanism to manage equipment failures without impacting the end-users.

The success of Google's cloud computing services and technologies has sparked interest from several companies and researchers as well as from the open source community, leading them to develop a set of similar technologies.

Open source software projects that imitate Google's technologies include, but are not limited to:

- Apache with HBase, a BigTable like database that provides random, real time read/write access to Big Data;
- Zvent with Hypertable, designed to manage and process large sets of data on a large cluster of commodity servers;
- Apache Accumulo, built on top of Hadoop, ZooKeeper, and Thrift, has cell-level access labels and a server-side programming mechanism;
- Facebook with Cassandra technology;
- LinkedIn with Vold'mot technology;
- Kai, a free software implementation of Dynamo from Amazon.

These technologies have one common goal: to be able to process a massive amount of data (i.e. several terabits or even petabytes of data) in a reliable and efficient way to meet the needs of cloud computing applications. During this same period, large software companies such as

IBM, Oracle, Microsoft and Amazon (DeCandia et al., 2007) also started developing their proprietary cloud computing technologies to compete with Google.

Google also published (Fay et al., 2006) a number of case studies presenting many services using their new cloud computing technology: Google Analytics, Google Finance, Orkut, Personalized Search, Writely and Google Earth (Ghemawat et al., 2003). However, Google specified that these new technologies, in particular the BIGTABLE database, do not support the relational model and therefore cannot be exploited with Structured Query Language (SQL), a language that allows the manipulation of relational databases (Fay et al., 2006). On the other hand, these new database technologies require software engineers to design databases differently in columnar data structures and use the MapReduce programming language to rapidly access large amounts of data (J. Dean & Ghemawat, 2004).

Using or moving towards this new database technology is a real challenge for software engineers for several reasons:

- Difficulty using and design non-relational databases;
- Difficulty re-engineering existing applications that work with relational DBMSs for cloud computing technologies;
- Difficulty installing an environment conducive to the use of these new technologies;
- Absence of courses explaining this new technology in universities.

Two research problems have emerged from this situation. The first is the difficulty of having a cloud computing definition that withstands the test of time and is based on real use cases rather than an enterprise’s need. The second research opportunity addresses the difficulty in understanding how to convert an existing information
system from current database technology to take advantage of Google's columnar database technologies (Fay et al., 2006).

The following section presents an overview of the cloud computing concept and its definition.

CHAPTER 1

OVERVIEW OF THE CLOUD COMPUTING CONCEPT AND DEFINITION

This chapter presents a review of the literature with regards to the cloud computing concept and its definition. First, many cloud computing definitions are presented, highlighting the NIST definition as a broadly adopted definition (Bohn, Messina, Liu, Tong and Mao, 2011). Then the cloud computing usage model and types are introduced followed by a comparison of some competing technologies that are similar to cloud computing.

1.1 Cloud Computing Definition

Cloud computing as concept have gained a lot of attention in both industry and academia (Donno, 2019). It has become more confusing due to the fact that it is a "perfect marketing buzzword" (Wayner, 2008), or as Weiss labeled it—"a buzzword almost designed to be vague" (Weiss, 2007). Cloud computing is undoubtedly a neologism that must be added to the knowledge of every person working in the field of information technology—a neologism that is also very popular for its various and ambiguous definitions (Rimal & Choi, 2009). From a reading of the cloud computing literature, various companies or IT industry professionals define cloud computing in ways that reflect their own views, understanding and business goals.

As Donno (Donno, 2019) mentions, a clear and neat definition of the cloud computing paradigms is hard to find in the literature. This makes it difficult for researchers new to this area to get a concrete picture of the paradigm.

According to Liming (Liming, 2008), cloud computing is a delivery of a resource and a usage pattern, in other words, it is getting a resource (hardware or software) through a network. The network in this particular case is called "Cloud". The hardware resource in the network seems extensible to infinity and can be used anytime and anywhere.

Another proposed definition of cloud computing comes from Foster (Foster, Yong, Raicu and Lu, 2008): “Cloud computing is a large-scale distributed technology that is driven mainly by economies of scale, where a group of services such as storage, platforms, and computing power are dynamically scalable and delivered on demand to external customers through the internet”.

Table 1.1 A Sampling of Cloud Computing Definitions

Authors/Organization	Definition
Gartner (Gartner, 2019)	Gartner defines cloud computing as a style of computing in which scalable and elastic IT-enabled capabilities are delivered as a service using Internet technologies.
National Institute of Standards and Technology (NIST) (Mell & Grance, 2011)	Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model is composed of five essential characteristics, three service models, and four deployment models.
ISO/IEC JTC 1 (ISO/IEC, 2009)	Cloud Computing provides IT infrastructure and environment to develop/host/run services and applications, on demand, with pay-as-you-go pricing, as a service. It also provides resources and services to store data and run applications, on any device, anytime, anywhere, as a service.

The basic concepts expressed by the term are quite simple. The word computing refers to any activity that involves computer processing or storage (Shackelford et al., 2006). A computer manipulates and stores data, which commonly resides on the hard drive of a computer, or in other hardware such as NAS (network-attached storage), commodity hardware (affordable and readily available hardware), the mainframe, etc. The hard drive can store anything, including structured data, unstructured data, software, databases, etc. In cloud computing terminology, these capabilities are designed as services, and these services are offered in a cloud, from and to any place where an Internet network is available. In other words, its location does not matter.

Where the confusion begins, and why so many different definitions were generated, is if those definitions try to include: 1) different perspectives (i.e. Infrastructure versus Software Engineering); 2) too many technical details; 3) a specific technology point of view. For example, some definitions include concepts like billing features, type of access, security issues, ownership of data, and even quality features associated with the technology. Since these concepts vary depending on the technology and can evolve, the definition of cloud computing can become broader and fuzzier over time.

One of the broadly adopted definitions (Bohn et al., 2011) for cloud computing has been proposed by the National Institute of Standards and Technology (NIST): "Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model promotes availability, and is composed of **five essential characteristics**, **three service models**, and **four deployment models**" (Mell & Grance, 2011).

The NIST definition (Mell & Grance, 2011) implies that the terminology of the essential characteristics, service models, and deployment models must also be precisely defined. Table 1.2 shows how these three concepts are defined in the NIST proposal.

Table 1.2 NIST Definition of Cloud Computing (Mell & Grance, 2011)

Essential characteristics	• On-demand self service • Multi network access • Resource pooling • Rapid elasticity • Measured service
Service models	• Cloud Software as a service (SaaS) • Cloud Platform as a Service (PaaS) • Cloud Infrastructure as a Service (IaaS)
Deployment models	• Private Cloud • Community Cloud • Public Cloud • Hybrid Cloud

The NIST definition shown in Table 1.2 was initially intended to serve as a means for broad comparisons of cloud services and deployment strategies, and to provide a baseline for discussion—from what cloud computing is, to how best to use it. The NIST definition raises a key issue. Cloud computing is complex, especially considering the nature of its components. Defining it requires that every essential characteristic, service model, and deployment model be well defined and that these elements do not change over time.

1.2 Cloud Computing Usage Model: Computer Utilities

Utility computing is not a new paradigm according to Foster (Foster et al., 2008); rather it is a supply model for which a computer resource such as software or storage is packaged in a measurable service similar to a utility such as electricity, water or natural gas. In other words, utility computing is the means of billing services on demand offered by cloud computing.

This usage model has the advantage of having no, or very low, initial cost. A cloud user can therefore lease services through cloud computing as shown in Figure 1.1 and only pay-per-use with the computer utility. It can thus respond to a peak of IT needs without investing in the purchase of new hardware.

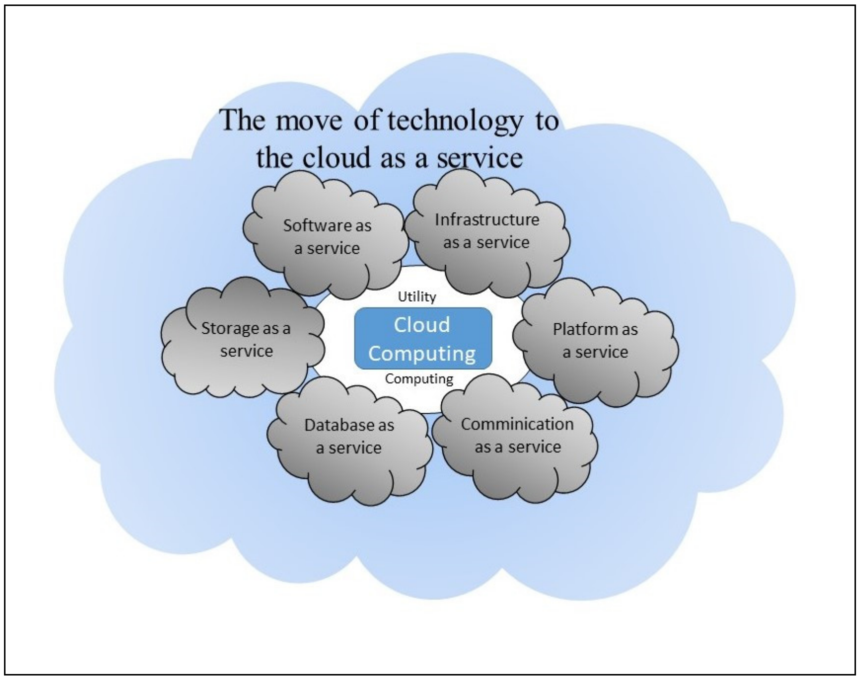


Figure 1.1 The Move of Technology to the Cloud as a Service

1.3 Cloud Computing Types

Rimal (Rimal et al., 2009) describes three models for the use of cloud computing:

1. Public cloud computing, where companies are served by cloud computing providers directly from the Internet, are also called Web services. This model allows companies to access a dynamic supply without having to invest in expensive hardware or to maintain hardware and software.
2. Private cloud computing allows companies to manage their own cloud computing. In other words, there are no more suppliers: companies must implement their own services based on cloud computing locally (on their premises with their own infrastructure). This model allows companies to manage their own data and processes without any restrictions on network speed, security or legal requirements of suppliers.
3. Hybrid cloud computing consists of a combination of the first two models, namely: the use of public and private cloud computing at the same time, as described in Figure 1.2.

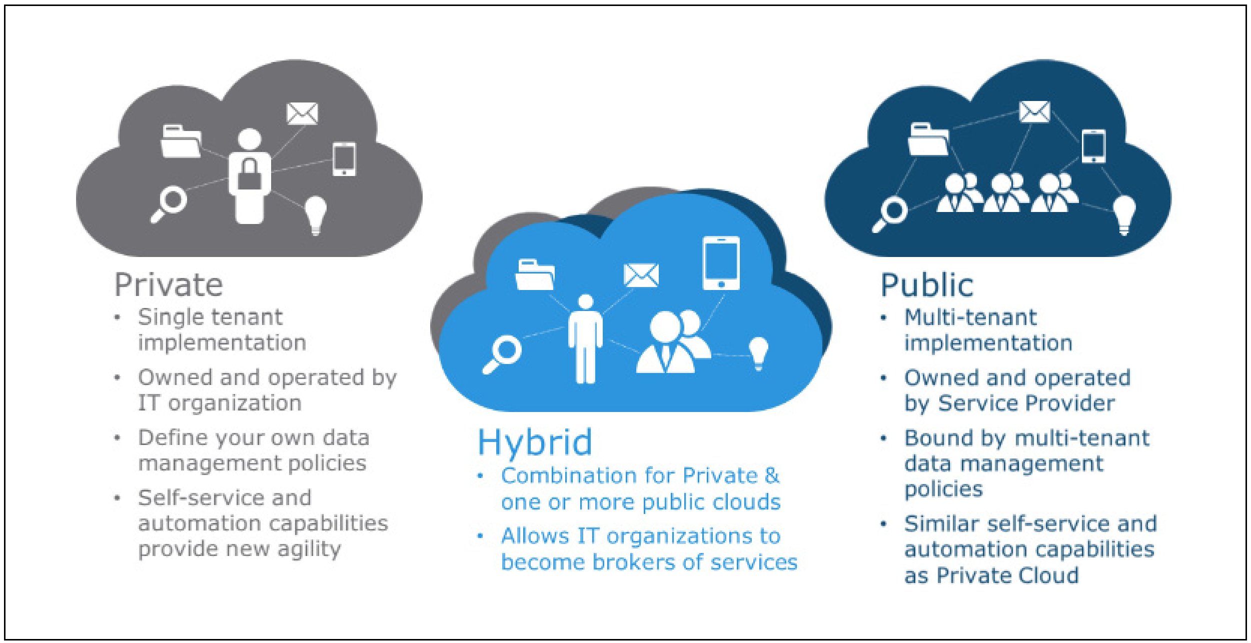

Private
Single tenant implementation
Owned and operated by IT organization
Define your own data management policies
Self-service and automation capabilities provide new agility
Hybrid
Combination for Private & one or more public clouds
Allows IT organizations to become brokers of services
Public
Multi-tenant implementation
Owned and operated by Service Provider
Bound by multi-tenant data management policies
Similar self-service and automation capabilities as Private Cloud

Figure 1.2 Types of Cloud Computing (Anderson, 2016)

1.4 Cloud Computing and Other Similar Concepts

Cloud computing is often compared with other similar concepts or technologies such as Grid Computing (Fiore et al., 2011). However, cloud computing can be implemented using different architectures, one of which is the computer grid. There are differences between cloud computing and the computer grid. The following section summarizes these differences.

1.4.1 Cloud Computing and Grid Computing

The computer grid is a grouping of powerful computers that are weakly dependent (computers can act independently of each other) and are brought together in order to offer a solution, most often, to a scientific problem of scale with many calculations. The entire grid is made available to a user for a predetermined period of time, which is different from cloud computing that shares its resources with multiple users at the same time. In addition, computers that form the computer grid are geographically dispersed most of the time in contrast to cloud computing which groups its data center in one place (Ghemawat et al., 2003). Cloud computing by Google

AppEngine (Barrett, 2009) is perhaps the exception in attempting to divide their computers into two geographically separate data centers.

In addition to the difference in the location of the infrastructure, there are other elements that differentiate cloud computing from the computer grid. These differences are summarized in Table 1.3 (Foster et al., 2008).

Table 1.3 Comparison between Cloud and Grid Computing (Foster et al., 2008)

Elements of comparison	Cloud Computing	Grid Computing
Business Model	Utility computing	Project-oriented model: The computer grid community gathers its clusters of computers to have a greater capacity and computational power. The computer grid generally does not have a billing model; in return each user contributes with a predetermined number of computers and is given access to all or part of the resources of the computer grid.
Architecture	Cluster of convenience computers located in one physical location	Powerful and costly computers, also called super computers, shared across the network for multiple users scattered around the world. Each organization is called a virtual organization because of its available power of calculation in remote places.
Protocol	Depends on the architecture with which cloud computing is implemented. Each implementation of the concept of cloud computing has its own communication protocol	To allow sharing of resources across the network, the grid has several protocols. Ian (Foster et al., 2008) states: "The computer grid defined and provided a set of standard protocols, middleware, tools and services based on these protocols. Interoperability and security are the main concerns of the IT grid infrastructure because of the resources coming from different domains, and has both global and local use resource policies, configuration of hardware, software and platform. These may vary in availability and capacity."

In summary, Table 1.3 highlights the differences between cloud computing and the grid computer. Cloud computing provides services over the Internet using a distributed architecture composed of convenience computers in order to be highly scalable and it supports data loads and calculations from a heavy load of potential users. The computer grid, on the other hand, aims to bring together computers from several institutions to solve large-scale scientific problems by giving access to the computing power of these combined supercomputers for one user at a time.

1.4.2 SaaS and Cloud Computing

SaaS is a component of cloud computing; it is a software distribution model through which applications are accessible via the Internet and on demand as described in Figure 1.3. Youseff (Youseff, Butrico and Da Silva, 2008) describes a layered representation of the services offered by cloud computing and places SaaS at the top. This component, or service layer, is the cloud computing service that this research is focused upon.

SaaS is primarily an online application that uses a database. This database can either be a relational database managed by an RDBMS and the SQL language for data access, or a distributed, non-relational database, which is the subject of this research.

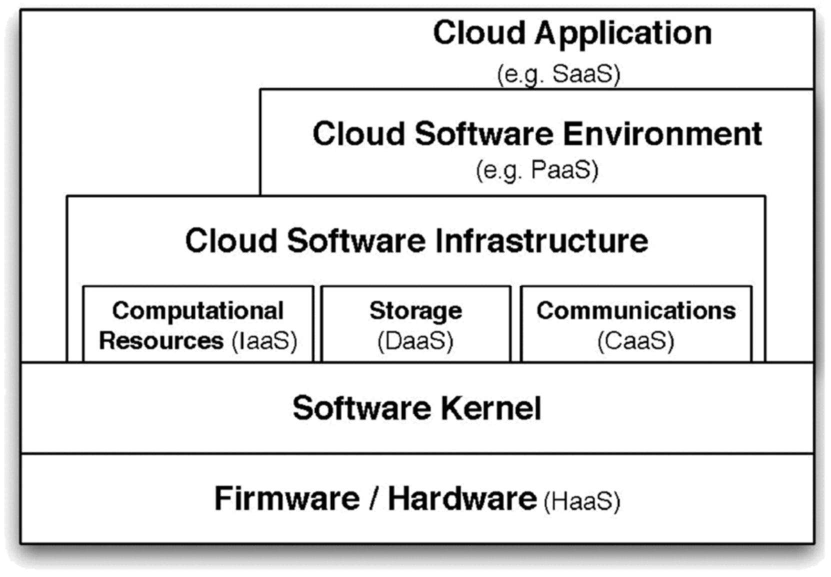

Cloud Application
(e.g. SaaS)
Cloud Software Environment
(e.g. PaaS)
Cloud Software Infrastructure
Computational Resources (IaaS)
Storage (DaaS)
Communications (CaaS)
Software Kernel
Firmware / Hardware (HaaS)

Figure 1.3 The Layout of Cloud Computing Components (Youseff et al., 2008)

1.5 Conclusion

This first chapter has presented an overview of the cloud computing concepts and definitions. Multiple definitions were presented including the broadly adopted NIST definition that defines the overall cloud computing concept as well as the essential characteristics of cloud computing and the various forms it can take in terms of both delivery and deployment models.

The next chapter presents a literature review of the Big Data technologies that are becoming a standard for many organizations, as cloud computing offers an easy way to process very large quantities of data. The technical descriptions of Big Data database technologies are also needed to help the reader understand the motivation for the main research contribution of this thesis, which is finding a list of rules for converting relational database schemas to the schema of a non-relational database and to clarify the cloud computing.

CHAPTER 2

INTRODUCTION TO BIG DATA TECHNOLOGIES THAT ARE PROMOTED BY CLOUD COMPUTING TECHNOLOGIES

This chapter introduces the distributed database technologies that are powering the concept of Big Data. First, we present the well-known relational database model and its limitation when used to implement a large-scale cloud-based solution. Then we introduce the NoSQL model used by Big Data applications. An overview of a specific NoSQL technology used by the Hadoop open source project follows, which introduces the Hadoop distributed database named HBase that is used in the research case studies.

2.1 Relational vs. Non-Relational Database

2.1.1 The Relational Database Model

Relational databases are databases structured according to the principles of relational algebra. They are essentially composed of tables; tables are composed of rows and columns. The relational term refers to the relationships and constraints defined between the tables. Accessing a relational database is performed using the SQL language and consists of queries that access data from one or more tables. A join is a query that accesses one or more tables. Normalization is the data structure used to ensure data consistency. Relational databases are implemented using RDBMS (Relational database management systems) that are still significantly more common than NoSQL databases (Shay, 2018). They are the most deployed in enterprises (TechTarget, 2017) with databases such as Oracle, SQL Server, Postgre SQL, MySQL and many others.

Relational databases have been and are still very popular due to their ease of use. However, even if their use is simple (declarative and non-procedural), the internal evaluation of queries remains very complex because SQL queries specify the *what* and not the *how* (Godin, 2006).

For example, the internal operation of the RDBMS for a basic SQL query of type SELECT can have an execution path that contains hundreds or thousands of queries invisible to the user.

2.1.2 Relational Database Limitations

Relational databases offer many benefits including robustness, simplicity, flexibility, scalability, compatibility, and more. But it is not necessarily the best compared with a solution that focuses only on few benefits such as fault tolerance and scalability. The latter has become a primary need, especially after the emergence of cloud computing which dramatically increases the number of users who deposit, transit and move data permanently on a SaaS application. Data from a cloud-based application can double their data loads and calculations in a matter of days, as was the case for YouTube (J. Cryans, A. April and A. Abran, 2008), and this is difficult to manage with a relational database that is hosted on a single server.

Relational databases evolve very well as long as the database remains in a single server. When it reaches limits in computational power and space, distribution on several servers becomes inevitable and this is where this technology shows its limits. Distributing a relational database over several hundred or thousands of servers is not an easy task. The distribution adds a lot of complexity to the data model because of the relationships between the tables and the characteristics that allowed the databases to be robust, simple to use, etc., dramatically reducing its ability to manage a large amount of data and calculations across a vast pool of interconnected servers. There are a number of challenges that a relational database will face when attempting to scale:

- If the service grows in popularity, too many reads will hit the database and cached memory will have to be added to the common queries. Reads will no longer have the ACID (Atomicity, Consistency, Isolation and Durability) proprieties;
- If the service continues to gain in popularity, and too many writes are hitting the database, a vertical scale (a server upgrade) will be required, which means that the cost will rise because new hardware will have to be purchased;

- New features mean higher query complexity, which leads to many more joins, and data denormalization has to be performed to reduce them;
- Increasing usage can swamp the server, and it will begin to operate too slowly. A solution might be to stop performing any server-side computations;
- If some queries are still being processed too slowly, one solution would be to determine the most complex ones and try to stop joining in these cases;
- If writes become slower, one solution might be to drop secondary indices and triggers. A next step could be to remove indices altogether.

At this point, scaling horizontally (adding more servers) is needed with attempts to build some sort of partitioning on the largest tables, or looking into some of the commercial solutions that provide multiple master capabilities. Ultimately, the conversion from a single or sharded relational database to a shared, remotely hosted relational database using a NoSQL schema may be considered. Many examples of this progression are provided in the literature: for instance, the YouTube example. YouTube first used a relational database (i.e. MySQL) with a master-slave replication, but eventually arrived at a point where the writes were using all the capacity of the slaves. Like many other organizations facing this situation, they tried partitioning their tables into shards so that the sets of machines hosting the various databases were optimized for their tasks (J.-D. Cryans, A. April and A. Abran, 2008). Ultimately they had to convert from relational database technology to NoSQL database technology. The next section provides an overview of the NoSQL distributed database model.

2.1.3 Distributed Database Model

Relational databases and distributed databases are fundamentally different because distributed databases used for Big Data situations often do not use relational models and therefore do not use the foundations of relational algebra and the SQL language to access data in the Cloud. This is not to say that relational databases are not used in cloud computing applications. It is simply a matter of using the right technology depending on the need and both relational and non-relational databases address different needs. Relational databases are used in cloud

computing for parts of applications with a limited amount of data in a hybrid mode whereas recent distributed databases developed to address Big Data problems in the cloud are becoming very popular for fast growing SaaS applications and for applications with very large volumes of data and computations.
The choice of one of the distributed databases is typically motivated by the following needs:

- Data is large (several gigabits or terabits);
- SaaS type application with the objective of being highly extensible;
- Expand infrastructure quickly and inexpensively.

Two different database options were introduced when implementing the concept of cloud computing or, more precisely, SaaS which is a component of cloud computing or a leading layer according to Youseff (Youseff et al., 2008). Relational databases can therefore implement cloud computing technology, but with growing difficulty when a SaaS-type application processes large amounts of data and calculations.

2.2 Hadoop Project

Hadoop is an open source software project consisting of a distributed infrastructure for batch processing of large amounts of data (several gigabits, terabits or even petabytes). Even if Hadoop can be used on a single computer, its true strength lies in its ability to process distributed data on a large number of computers.

This free software project came into being following the failure of Apache's Nutch project, which aimed to create a search engine capable of indexing and processing all of the Web data in an effective way (White, 2009b).

The Hadoop project consists of several interrelated sub-projects whose objective is to offer a free software solution allowing for an effective implementation of the concept of cloud computing. These sub-projects are presented in Figure 2.1.

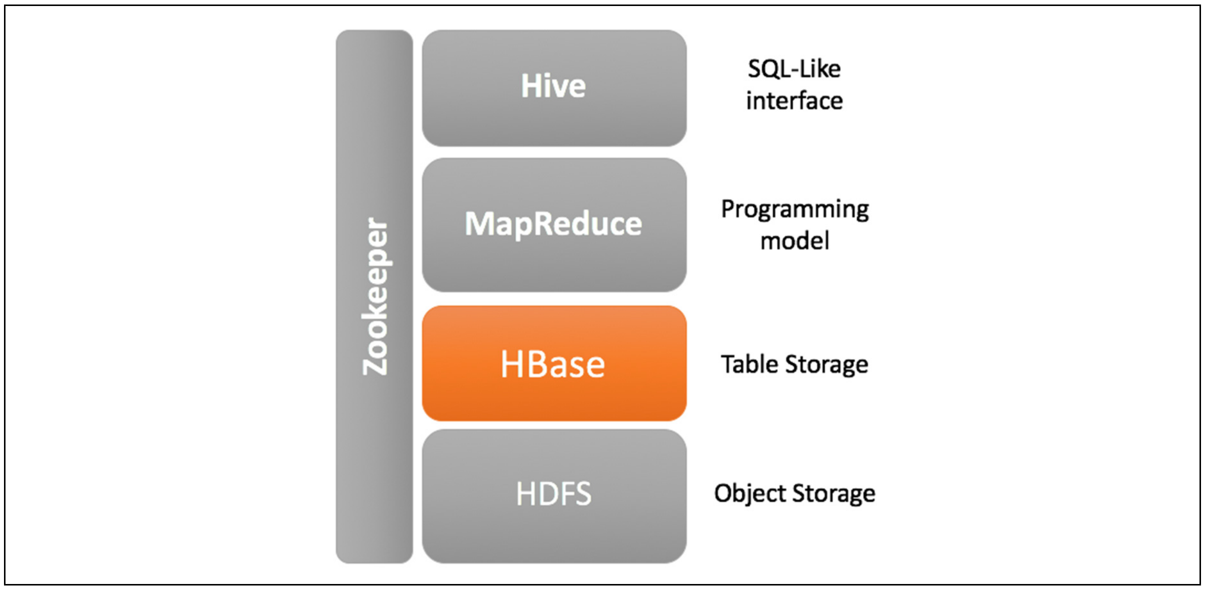


Figure 2.1 The Components of the Hadoop Project

At the bottom of Figure 2.1, the Hadoop HDFS (Hadoop Distributed File System), which is a file system in a distributed architecture that processes unstructured, index-free data.

HBase, the Hadoop database, relies on HDFS and thus benefits from all the properties of the HDFS in addition to being structured with an Index.

Hive is a data warehouse built on top of HDFS, with simple tools to give structure to the data in the HDFS. Hive has a language called QL (Query Language) that resembles standard SQL, allowing programmers familiar with the SQL language to use the Hadoop infrastructure.

MapReduce, whose name has not changed since its appearance in Google publications (J. Dean & Ghemawat, 2004), is a programming model that allows for the processing of distributed data. MapReduce is used in several Hadoop products including HDFS, HBase, and Hive. It provides different functionalities depending on the product used.

In summary, Hadoop offers a shared storage environment, as well as analysis of the elements stored through a programming model named MapReduce.

2.2.1 The Initial Need that Led to the Creation of Hadoop

Disk storage capacity has been in constant evolution. If the storage standard was megabit in the 1990s, gigabits and terabits are the standards at the time of this research.

The speed of access, or the speed at which the data can be read from the disks, has not changed in a manner comparable to that of the evolution in storage capacity. Tom White (White, 2009b) states that a typical hard drive of the 1990s could store 1370 megabits of data with a transfer rate of 4.4 megabits per second, which resulted in reading the entire disk in less than 5 minutes. Ten years later, a hard disk of a terabit with a transfer speed of 100 megabits per second, takes more than two and a half hours to read data from the entire disc.

The solution of reading several disks at the same time seemed to solve this problem; in fact, reading large amounts of data on multiple disks in parallel is actually faster than reading this same data from a single computer. On the other hand, parallelism generates new problems, such as fault tolerance and the combination of data read in parallel. Hadoop excels in solving these problems by offering a distributed environment for data storage, enabling faster reads and writes. Hadoop uses the MapReduce programming model that provides an abstraction of parallel manipulations to programmers.

2.2.2 Does Hadoop Offer an Alternative to the Use of RDBMS?

Hadoop and MapReduce are not an alternative to RDBMS; Hadoop software engineers consider it a complement to RDBMS. When it is a small amount of data (typically less than one gigabit), RDBMSs are more recommended than Hadoop, but for a massive amount of data (several gigabits or terabits), MapReduce by Hadoop gives excellent results (White, 2009b). Table 2.1 below shows the differences between RDBMS and the MapReduce of Hadoop.

Table 2.1 Differences between RDBMS and Hadoop MapReduce

	RDBMS	**HADOOP MapReduce**
Quantity of data	Gigabits	Gigabits, Petabytes and Terabits
Access	Interactive and batch	Batch
Update	Read/Write multiple times	One writing, several readings
Structure	Static schema	Dynamic schema
Integrity	High	Low
Course	Non-linear	Linear

Nowadays, more and more RDBMSs such as Aster Data and Greenplum offer features that resemble those of MapReduce for processing large amounts of data (White, 2009b). Distributed non-relational databases like HBase or Hive offer options found in RDBMSs such as a high level of query language that is more familiar to programmers with RDBMS knowledge.

2.2.3 Hadoop and Volunteer Computing

The Hadoop project is often compared with volunteer computing (White, 2009b) because MapReduce processes the data in a distributed way. Voluntary computing is a concept of CPU time shares: volunteers give access to their CPU for research organizations such as SETI @ home (an organization whose goal is to seek all forms of intelligence external to planet Earth) or GIMPS (Great Internet Mersenne Prime Search) to perform intensive calculations.

Voluntary computing divides a problem into several small tasks; they are then sent in the form of computations to computers that are located all over the world. These computers share their CPU times voluntarily. The calculations can take hours or even days to complete and the results are finally returned to the organization that requires resolution of the problem. Once this resolution is completed, computers that share their CPU times voluntarily receive other small tasks to solve.

Voluntary computing can be very similar to the calculations made by MapReduce of Hadoop; however, there are also great differences between them. Table 2.2 summarizes these differences.

Table 2.2 Differences Between Hadoop and Voluntary Computing

	Hadoop	**Volunteer computing**
Physical architecture	Cluster of convenience computers in the same data center	Computers around the world (personal computers)
Type of calculations	On demand, as required	Intensive calculations (requiring a lot of time)
Problem solving	Several minutes or days on dedicated and secure computers	Several days or years on non-dedicated and non-secure computers
Network speed	Powerful network	Variable (depends on each network, each volunteer)

In this sub-section, the Hadoop project has been presented along with its components and what motivates the use of this technology, as well as its position in relation to existing technologies such as voluntary computing and RDBMS. The next sub-section presents the detailed characteristics of HBase that need to be fully understood before a software engineer decides to undertake a database conversion effort.

2.3 HBase Characteristics

2.3.1 Associative Table (a MAP)

HBase is a database that contains sorted association tables. An association table is an abstract data type consisting of a list of unique keys and a list of values; each key is associated with one or more values. The operation of finding the value(s) associated with a key is called indexing, which is the most important operation in an association table. Table 2.3 shows an association table made up of three keys (names) associated with three values (telephone number):

Table 2.3 Contact table

Name	Telephone #
Fabien	5146656676
Vincent	5140097887
François	4162228989

2.3.2 Persistent

HBase allows for data persistence. A database is said to be persistent when it retains previous versions of its data as they are changed (Driscoll, Sarnak, Sleator and Tarjan, 1986). The operations underlying the data stored in HBase do not actually modify this data, rather they create other versions of that data (Jimbojw, 2008). HBase, by default, keeps three versions of the same data when it is modified, but this figure can be changed in the HBase configuration.

2.3.3 Distributed

HBase is designed on the Hadoop file system, the HDFS. Each piece of data is divided into a 64-megabit block and is stored in several computers in the HDFS computer cluster. Each data is also replicated several times in different computers in order to resist a possible hardware failure. The key sets and values stored in HBase are ordered alphabetically. Using the example in Table 2.3, the data in HBase would be ordered as displayed in Table 2.4.

Table 2.4 Contact table (ordered)

Name	Number
Fabien	5146656676
François	4162228989
Vincent	5140097887

Thus, when one accesses the data, similar elements like Fabien and François are closer. This sorting is particularly important because it allows a user to choose the keys in a way to obtain a good locality (the data sought are next to each other) to access the data.

In (Fay et al., 2006), Google mentions that for storing web pages, pages with the same domain names must be grouped together to achieve increased performance of data reads. The choice of the reverse URL key of the websites 'com.google.maps/index.htm' instead of 'maps.google.com/index.html' allows for having the data of websites of the same domain stored close to each other. This is a way to ensure that fewer computers are used to perform the subsequent analyzes. For example, 'mail.google.com' will be closer to 'www.google.com' than to 'mail.yahoo.com' and will be more likely to be on the same computer than if the data was stored a few thousand records later. It should be noted here that the sorting is done on the keys and not on the values associated with the keys. HBase is a multidimensional database; each line key points to multiple association tables with multiple keys. Table 2.5 illustrates an example of such a table.

Table 2.5 Multidimensional Table

Line key: (#product)	Column: ‘Feature’		Column: ‘Price’	
123001	‘feature: weight’	230kg	‘price: buying’	1231$
	‘feature: volume’	12pi	‘price: wholesale’	930$
123002	…	…	…	…

The characteristic features and prices are called column families. Weight, volume, buying and wholesale are called qualifiers. Thus, each key that is the product number in Table 2.5 has several column families. These are created at the beginning of table creation and are hardly ever changed. Each column families can have several qualifiers or none at all. Qualifiers can be added after the tables have been created.

Note: To lighten the text, we refer to qualifiers to designate a column under a column family. A qualifier is associated with a column family but is not required. A column family can exist without a qualifier.

Each column in a family column can keep a predefined number of versions of its data. In Table 2.6, T1 represents the price of a product at a time T1 and T2 represents the same price modified at time T2. HBase allows access to the data versions by specifying the time stamp associated with each cell (Line key, column family: qualifier).

Table 2.6 Multidimensional Table with Time Dimension

Line key: (#product)	Column : 'Feature'		Column: 'Price'	
123001	'feature : weight'	230kg	'price: sale' **T2**	1231$
			'price: sale' **T1**	1291$
	'feature: column'	12pi	'price: wholesale'	930$
123002	…	…	…	…

2.3.4 Sparse

HBase is a sparse database because you can have a different number of columns in the family columns for each line (J. Cryans et al., 2008).

2.3.5 Column Oriented

For reading performance requirements, the data stored in HBase is column-oriented instead of line-oriented. A table created in a database is actually stored in memory as a sequence of bytes. The storage of column-oriented Table 2.4 would look like this in memory:
Fabien, François, Vincent; 5146656676, 4162228989, 5140097887
instead of line-oriented storage:
Fabien, 5146656676; François, 4162228989; Vincent, 5140097887

2.3.6 High Availability and High Performance

HBase shares the same properties as the Hadoop file system as it was created over it. HBase is highly available thanks to high-performance data block replication in HDFS and the implementation of MapReduce in HBase.

2.4 Why HBase?

The use of HBase is mainly due to the desire to use the Hadoop HDFS properties for semi-structured data (J. Cryans et al., 2008). The Hadoop file system, HDFS, was created to process unstructured data with sequential readings (one data in succession). According to White (White, 2009b), the creation of HBase was motivated by the need to use the Hadoop architecture, namely the cluster of convenience computers, the power of parallel computing of MapReduce for applications which often make random reads and writes, and finally the reliability and availability of data generated by data replication.

It is the limitations of relational databases to process large amounts of data that actually motivated the development of HBase. Indeed, according to Cryans (J. Cryans et al., 2008), for a relational database to handle a large volume of data, companies often resort to replication, as was the case for YouTube (J. Cryans et al., 2008). The latter used MySQL's master-slave replication to the point where the writes on the disks were fully utilizing the capacity of the computers. Other companies have also used similar solutions, partitioning their RDB database tables so that subsets of computers become hosts to different databases optimized for specific tasks. After several iterations, this solution quickly reached its limit and the relational model became totally denormalized, which makes keeping them very difficult.

2.5 HBase Architecture

As HBase is built on HDFS, it also shares its architecture. HBase data is distributed across a cluster of convenience computers. The tables are automatically partitioned horizontally into regions. Each region includes a subset of rows in a table. A region is, by design, the HBase distribution unit across the cluster. Depending on the size of a table, it may be composed of several regions; these regions are distributed across several hundred convenience computers with a replication of regions.

Figure 2.2 illustrates the HBase architecture, composed of a model similar to that of HDFS with a master-slave architecture (Hadoop Master and several Hadoop Datanodes) residing in a Linux operating system. HBase is written and used in Java language, a Hadoop Job Tracker and a Hadoop Task Tracker to manage the Map and Reduce functions. The machines 1 to n represent the Datanodes of HDFS. Each Datanode has an HBase Region Server that will contain the regions of the partitioned tables. Region servers can contain zero or more regions serving clients and also handle partitioning of regions so that the HBase Master is informed of a new region to be managed.

The HBase Master coordinates the Datanode Region Server by assigning regions to the right servers and restoring servers after a breakdown.

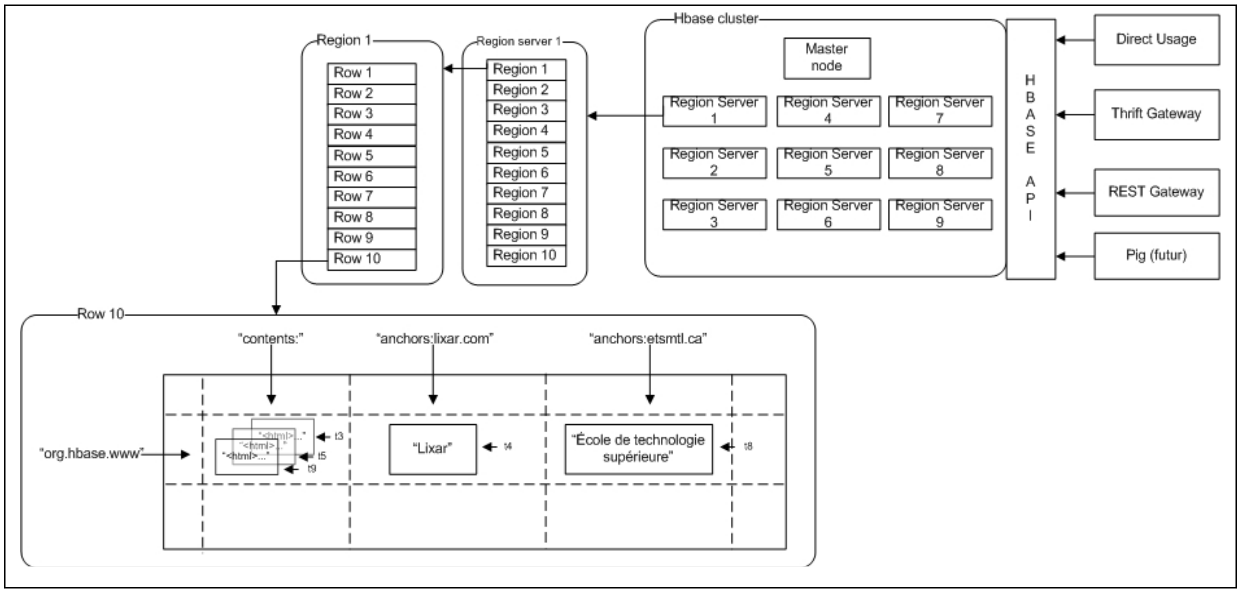


Figure 2.2 HBase Infrastructure (J. Cryans et al., 2008)

2.6 HBase Accessibility

HBase is a database written in Java and is therefore accessible from API (Application Programming Interface). It offers the standard CRUD operations that are: create, read, update and delete and tools for managing the cluster.

HBase is also accessible by using the Representational State Transfer (REST) gateway and offers operations such as Get, Put, Post, or Delete. Finally, HBase is accessible with the Thrift framework, which allows procedural calls with different languages such as C++, Ruby, PHP as well as several other languages.

HBase is a non-relational database that gathers several characteristics from several existing concepts and forms a technology capable of addressing quantities of data of the order of terabits. HBase is new because it does not use the SQL language and the design of the tables is different from that of relational databases.

2.7 Conclusion

In this chapter we presented Hadoop technologies, the Hadoop file system, the MapReduce programming model and the HBase database. This chapter helped in understanding 1) the concept of cloud computing, and 2) the technologies of Hadoop considered as technologies that effectively implement the concept of cloud computing. It demonstrates that these database technologies are quite different than what software engineers use daily, mainly relational database management system.

For the reader, the important elements to retain and that will be presented next are mainly the HBase distributed database characteristics and architecture, especially the fact that it requires the understanding of a novel concept, the columnar database model. As well, the data in a columnar database is not always accessed using the SQL language that is so common and widely used by software engineers today. This Big Data database technology requires a new way of designing databases and creates new challenges for software engineers that would like to convert their current systems to this technology.

The next chapter presents a literature review of database conversions for relational databases to non-relational database technologies.

CHAPTER 3

LITERATURE REVIEW OF DATABASE CONVERSION BETWEEN RELATIONAL AND NON-RELATIONAL DATABASES

3.1 Introduction

The previous chapter introduced NoSQL database technology, which is used when very large quantities of data need to be accessed. In order for software engineers to consider converting their current system to the Cloud using Big Data infrastructure, there will be a need to convert their existing relational database to this type of new technology. Database conversion is the activity of identifying how a database can be translated into another database type/paradigm or technology. Database conversion is not a new research topic. It started interesting researchers and the industry in the 1960’s when hierarchical and network databases fell out of favour and were massively replaced with relational databases (Joseph Fong & Bloor, 1994) (J. Fong & Huang, 1997) (Hudicka, 1998). For this research, which concerns converting relational databases to columnar databases, the focus is on initiatives that convert relational databases to NoSQL database technologies. This chapter summarizes the state of converting relational databases to columnar databases. The first section provides an overview of the conversion steps required.

3.2 Overview of Typical RDB to NoSQL Conversion Steps

This research is interested in how a relational database can be converted to a columnar database such as HBase. To understand which steps are required for such a conversion, a step back is needed to review what makes a columnar, cloud and NoSQL database, such as HBase, different from the well-known relational databases. Key HBase characteristics have been introduced by White as follows (White, 2009a):

- There are no real indexes: because the rows are stored sequentially, as are the columns within each row, there is no index bloating (dead tuple as a results of delete queries in relational database) and the insert performance is independent of table size;

- There is automatic partitioning: as the tables grow, they are automatically split into regions and distributed across all the available nodes;
- There is linear and automatic scaling with new nodes: when you add a node, point it towards the existing cluster and run the region server. The regions automatically rebalance and the load is spread evenly across servers;
- It can operate on commodity hardware: computer clusters of a private Cloud can be built at a cost as low as $1,000 to $5,000 per node. Alternatively, when using RDBMS technologies, they have been shown to have a higher consumption of input/output hardware, which is the most costly type of hardware in a cluster;
- It offers fault tolerance by default: using a large number of nodes with a distributed file system means that each node becomes relatively less critical, and there is no need to worry about individual node downtime since there is automatic replication.

Since a HBase database is completely different from a relational database, a conversion process in this case will require that all components (i.e. schema, data, application programs and queries) of the source database application be converted into their equivalents in the target database environment (Maatuk, Ali and Rossiter, 2008).

To represent an overview of the database conversion process, Figure 3.1 highlights four steps required in such a conversion. It starts with a source database assessment and analysis, which consists in deciding what part of the database needs to be converted (when in the presence of a hybrid database) or if the entire database will need to be converted. Once the source database assessment is done, it is followed by the creation of a target schema using the assessment and analysis findings. The second step is key and requires identifying a list of the most used/important queries and especially the ones that consume the most resources when they execute. This important step aims at discovering and ensuring that the most efficient target database schema possible, in HBase, will be designed. Identifying access patterns of the source database is a very important step since it defines how the data is accessed and how to replicate the data access logic in a different database technology, in this case, the HBase target database.

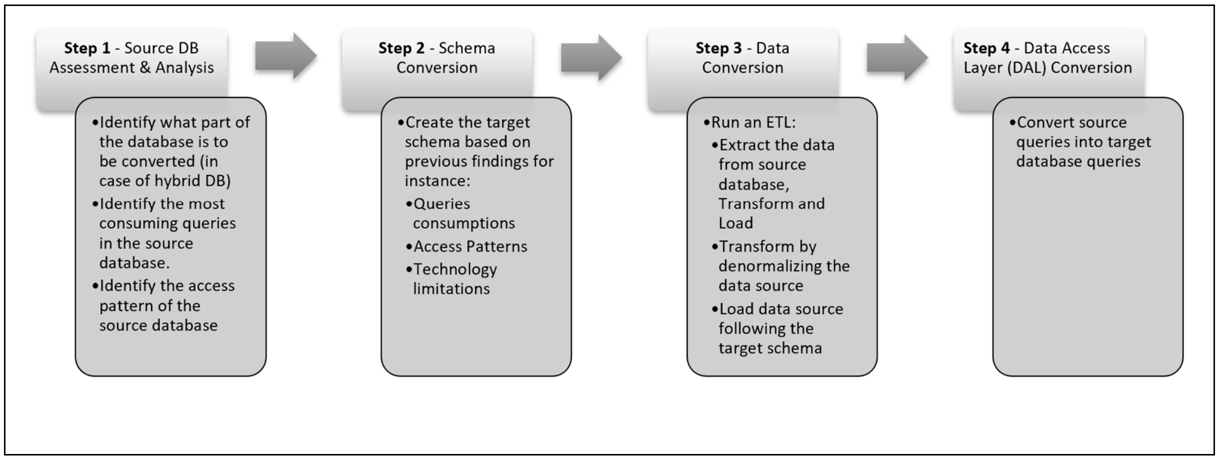

Step 1 - Source DB Assessment & Analysis
•Identify what part of the database is to be converted (in case of hybrid DB)
•Identify the most consuming queries in the source database.
•Identify the access pattern of the source database
Step 2 - Schema Conversion
•Create the target schema based on previous findings for instance:
•Queries consumptions
•Access Patterns
•Technology limitations
Step 3 - Data Conversion
•Run an ETL:
•Extract the data from source database, Transform and Load
•Transform by denormalizing the data source
•Load data source following the target schema
Step 4 - Data Access Layer (DAL) Conversion
•Convert source queries into target database queries

Figure 3.1 Database Conversion Process

Existing publications on this topic discuss the many ways to achieve a schema conversion. They will be discussed in the next section. The third step, the data conversion, is often named the extract, transforms and loads (ETL) step. Once a target schema is available, the data of the source database needs to be extracted, denormalized if needed, converted into the new schema layout, then loaded into the new target schema. Once the new database schema and data is determined, the final database conversion step will consist of converting the existing queries of the source database into queries that can be used with the target database technology.

Now that an overview of the conversion process has been described, the following section presents the database conversion literature review. For this research, which concerns converting a relational database to a columnar database, the review intends to present a synthesis of the database conversion publications that describe how to convert relational databases to NoSQL database technologies.

3.3 Database Conversion Literature Review

There are only few research papers published that present case studies of conversion from relational databases to NoSQL databases. Most proposals found share the same goal: how to efficiently convert an existing relational database to a NoSQL database.

Given that RDB technology is rooted in mathematical theory, it should be easy to convert any specific relational database implementation, such as a MySQL database for example, to any other relational database such as Oracle or MSSQL (Li, Xu, Zhao and Deng, 2011). NoSQL databases on the other hand, like HBase, do not use relational algebra and have a completely different schema design (Lars, 2013b). Another author highlights that NoSQL databases are typically designed considering a specific use case: queries and access patterns rather than using a general relation and normalization process.

There is very little information available in the literature on how to conduct schema translation. In this research, a schema translation uses an existing relational data model as an input and a non-relational (e.g., NoSQL) data model as an output. NoSQL databases are categorized by many types such as: Wide Column Store/Column Families, Document Store, and Key Value/Tuple Store among many others. The focus of this research has been set on a Wide Column Store/Column Families category, more specifically the popular Hadoop database called HBase.

It has also been reported in the literature that there is a difficulty when the time comes to convert an existing legacy system based on RDB technology to NoSQL database technologies for RDB-trained software engineers.

According to Maatuk, this specific problem of database conversion is typically solved using four steps (Maatuk et al., 2008):

1. Schema translation;
2. Data transformation;

3. Data conversion;
4. Data operations.

These steps are similar to the ones presented in Figure 3.1 with the exception that step 1 is missing in the Maatuk conversion steps. Step 1, in Figure 3.1, as mentioned earlier, consists of a source database assessment and analysis, which is an essential step since the subsequent steps (schema translation, data transformation, data conversion and operation) are based on the findings of the database assessment.

Another author reports that many tools and open source projects offer help with step 2 data transformation, step 3 data conversion, and step 4 data operation (Biswapesh Chattopadhyay, 2011). He highlights that there is little help or guidance for executing the schema translation.

Other authors (Hanine, Bendarag, and Boutkhoum, 2015) propose a data conversion methodology, from a relational to a NoSQL database, that suits most NoSQL database designs using only two steps. The first step identifies the logic of the source database, which aims at extracting the relationships by proposing an automated tool to load that logic using the table names, their attributes, as well as primary and foreign key constraints. The second step proposed is a mapping between the RDB and NoSQL database, which identifies which attributes of the RDB in the source database will be linked to the attributes of the target NoSQL database.

In addition, these authors (Hanine, Bendarag, and Boutkhoum, 2015) suggest aggregating related data within a single data structure in the target NoSQL database and eliminating the joins between tables as opposed to SQL, where the data is normalized by creating joins. The main reason for this solution is that the resulting NoSQL database does not need joins when the application has to retrieve a complete record. Even if the schema translation can be automated in this way, this proposal raises many issues and limitations; for example, it only considers converting a portion of the database and thus the resulting schema could be very different if the whole database is to be converted. Also, the proposed methodology is only suitable for a specific NoSQL database technology, in this case MongoDB, which is a document-based database and not a columnar one. As observed earlier, this conversion methodology does not address the application queries as it assumes they can be converted automatically. Lastly, this conversion proposal does not address which data is currently accessed together and which is not, for example Order_details or Order and Product together. This has a major impact on the target schema design.

Other authors (Zhao, Lin, Li, and Li, 2014) propose a much simpler schema conversion process between RDB and NoSQL databases based on the concept of a nested table. A nested table is basically a table within a table (Koopmann, 2008). The conversion proposal here aims at storing the structured data from an RDB into a NoSQL database using the concept of *references*. References, in NoSQL technology, are similar to relationships between tables in RDB with the exception that in a NoSQL database, these references are between parent and child layers. Layer concepts are simply the notion of smaller tables inside of bigger tables. In other words, when two tables in an RDB are related using a relationship, they need to be replicated in a NoSQL database as one table within another. The authors called this notion of layers semi-structured data in the target NoSQL database. The resulting NoSQL database in this model will have the same number of tables as the source relational database which somehow defeats the purpose of using a NoSQL database because relational properties such as ACID (Atomicity, Consistency, Isolation and Durability) are not maintained.

However, this conversion proposal has the advantage of converting any relational database into a non-relational database quickly and automatically. On the other hand, the resulting conversion approach has no conversion consideration for the application code that will be using this new database and more importantly, all the queries. The main issue with this proposal is that once the conversion is done, it does not ensure an optimal conversion. A second issue with this approach is that in most cases, keeping all the tables and references in a NoSQL database is not required or desired. For example, if we have two tables in an existing RDB that are related but the main query we want to convert and ensure its optimal performance only accesses a single table, then the new schema in a NoSQL database should be to keep these two tables separate (this means no notion of a nested table). A good point is that the proposal will work for column-oriented NoSQL databases such as HBase, Cassandra and Hypertable.

With the same goal of converting RDB to NoSQL database technology, other authors (Rocha, Vale, Cirilo, Barbosa and Mourão, 2015) propose a conversion approach composed of two modules, a conversion module and a mapping module. The proposed conversion module is responsible for automatically identifying all elements of the source database (tables, attributes, relationships, etc.) in order to create a target schema and a mapping module to act as an interface between the application, the source and target database. It intercepts all SQL transactions from the application, translates and redirects them to the target database. The conversion module will replicate the source database, mainly the table, into the target database and replace relationships from source database with references in the target database. In other words, the target NoSQL database will be identical to the source relational database but in a NoSQL format. More precisely, the framework will only work with a document-oriented database, MongoDB. The reason is that with the document-based database, each table in the RDB can be represented as a single document. This is a characteristic of MongoDB and not of all types of NoSQL databases. If we pick a column-oriented database, we would have to use a different approach, as recreating the same RDB structure in NoSQL is doable but not optimal. No consideration of access patterns has been studied nor any MongoDB specifications and limitations. The framework should be named to be a schema replicator from MySQL to MongoDB.

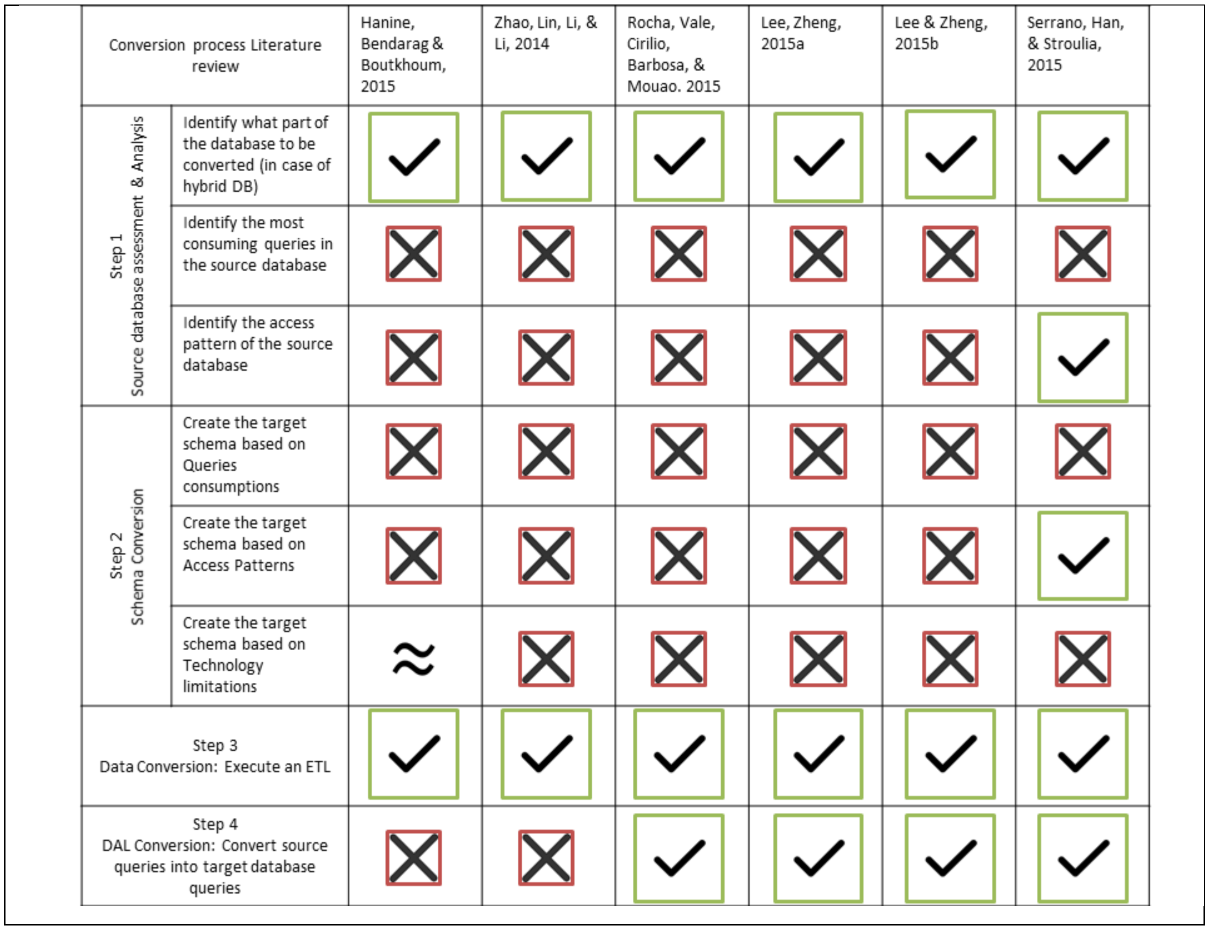
Next, Lee and Zheng (Lee & Zheng, 2015a) proposed an automatic SQL-to-NoSQL schema transformation approach using a MySQL to HBase database conversion as an example. The main objective of the proposed solution was to avoid cross-table queries. In this proposal, all tables that have a relationship were converted into a single HBase table. This approach breaks down the complexity of the conversion but it is hard to imagine that a large and complex RDB would practically fit into a single HBase table without affecting the performance of its queries. HBase is, after all, a ‘query first’ type of schema designed database.

The same authors (Lee & Zheng, 2015b) later propose an autonomous schema denormalization and conversion process demonstrated in the case study of a conversion involving a content management system to HBase. Again, the same objective of avoiding cross-table queries appears. In this paper, all the RDB tables that have a relationship were converted into a single HBase table. Once again, this is a simple solution to the conversion problem but it is hard to consider a solution that avoids the study of the access patterns of the application queries. Even if better performance is obtained using the target database technology over the RDB technology, as shown in the case study, the resulting schema cannot be optimal for all application queries that would have different requirements in terms of performance.

Another proposal found in the literature, by Serrano et al. (Serrano, Han and Stroulia, 2015), describes a conversion methodology in four steps. First, it proposes to convert every one-to-one and one-to-many relationship into a single merged HBase table. Second, it applies a recursive method to merge neighbouring tables. Third, it requires the design of a row key in HBase, and fourth, it creates views for different access patterns needed. In this proposal, an additional step is performed that consists of the extraction of access patterns using the query logs. This last step could have been considered an essential fifth step since access pattern extraction is a key step in a typical conversion process (as presented in the first step of Figure 3.1).

Lastly, Babu and Surendran (Babu & Surendran, 2017) proposed an overview of different methodologies to convert from RDB to NoSQL, each with different challenges and NoSQL database target. These conversion methodologies have already been presented and discussed above.

In summary, Figure 3.2 maps all the papers presented in this chapter to the conversion steps each article proposes. Most of the authors address the 3rd and 4th steps (i.e. data conversion ETL and DAL conversion) but they lack information about the first two steps of the conversion, which consists of the source database assessment and the schema conversion.



Conversion process Literature review		Hanine, Bendarag & Boutkhoum, 2015	Zhao, Lin, Li, & Li, 2014	Rocha, Vale, Cirilio, Barbosa, & Mouao. 2015	Lee, Zheng, 2015a	Lee & Zheng, 2015b	Serrano, Han, & Stroulia, 2015
Step 1 Source database assessment & Analysis	Identify what part of the database to be converted (in case of hybrid DB)	✓	✓	✓	✓	✓	✓
	Identify the most consuming queries in the source database	☒	☒	☒	☒	☒	☒
	Identify the access pattern of the source database	☒	☒	☒	☒	☒	✓
Step 2 Schema Conversion	Create the target schema based on Queries consumptions	☒	☒	☒	☒	☒	☒
	Create the target schema based on Access Patterns	☒	☒	☒	☒	☒	✓
	Create the target schema based on Technology limitations	≈	☒	☒	☒	☒	☒
Step 3 Data Conversion: Execute an ETL		✓	✓	✓	✓	✓	✓
Step 4 DAL Conversion: Convert source queries into target database queries		☒	☒	✓	✓	✓	✓

Figure 3.2 Literature Review Discussion Summary

As discussed previously, these conversion propositions have many pros and cons while they all missed many schema technology limitations. Even when some authors did consider some of these aspects, they only applied to their specific case study and technology. Finally, it would be very useful to have a generic conversion methodology that applies to all NoSQL database technologies.

One conclusion following this summary of conversion approaches is that a set of conversion rules would be useful and could encompass more situations. For example, these rules could be used as guidelines that would help to make the right decisions when creating the target schema. These rules would also be a guide to software engineers and database specialists to ensure they methodically consider all the conversion activities presented in Figure 3.1.

Many pertinent articles and discussions about this topic may not necessarily be published in academic conferences. The next section is dedicated to summarizing what is discussed about this topic in specialized books, blogs and popular web discussion sites regarding the conversion process from RDB to NoSQL.

3.4 Books, Blogs and Web Discussions

Many blogs, web discussions and slideshare presentations provide quick hints and high-level information on how to proceed with an RDB to HBase conversion. In most cases the knowledge proposed is domain specific and is not reusable at large. Cinnamond (Cinnamond, 2013) proposes 5 steps, where the third step is an HBase table mapping:

1. Model Creation: reverse engineer the source database UML
2. Code Generation: provision persistence and query-DSL java code
3. HBase Table Mapping: map data graphs and row keys to table(s)
4. Data Migration: MySQL to HBase
5. Services/App Creation: Build services, web app

The third step, HBase table mapping, consists of a configuration of a delimited, hashed, salted, formatted composite row key with xpath into the target data graphs and then mapping data graph roots to HBase tables.

This proposal applies to the lexical analysis of text, which is a very specialized use case. In addition, it was not mentioned how HBase tables were created.

The book Hadoop for Dummies (deRoos, 2014) includes a section entitled “transitioning from an RDBMS model to HBASE” where deRoos explains that this transitioning, meaning conversion, process requires denormalization, duplication and identifying intelligent keys.

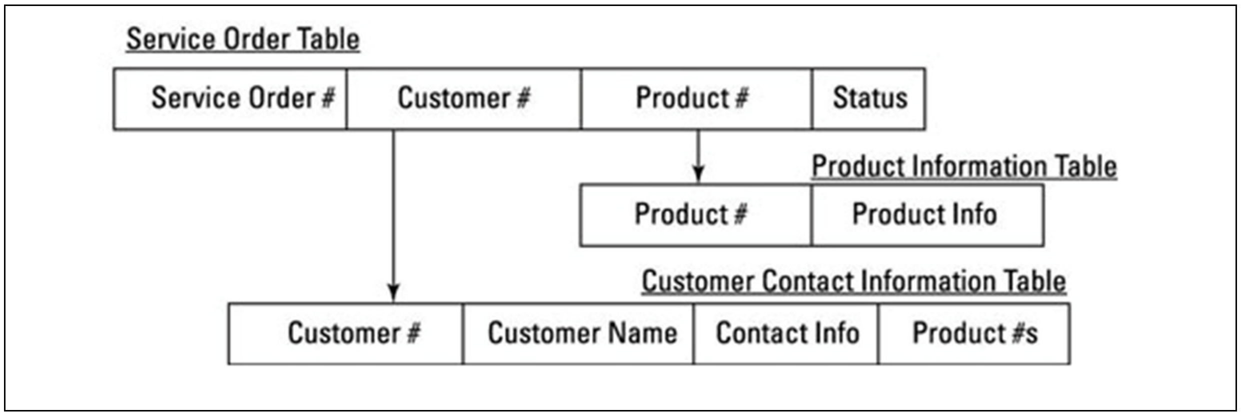


Figure 3.3 RDBMS Normalized Service Order Database (deRoos, 2014)

As shown in Figure 3.3, deRoos proposes an example where he states: "following the rules of RDBMS normalization, set up the sample Customer Contact Information table so that it is separate from the service order table in order to avoid losing customer data when service orders are closed and possibly deleted. Take the same approach for the Products table, which means that new products can be added to the fictional company database independently of service orders." The next figure illustrates a possible HBase schema—one that follows the DDI design pattern.

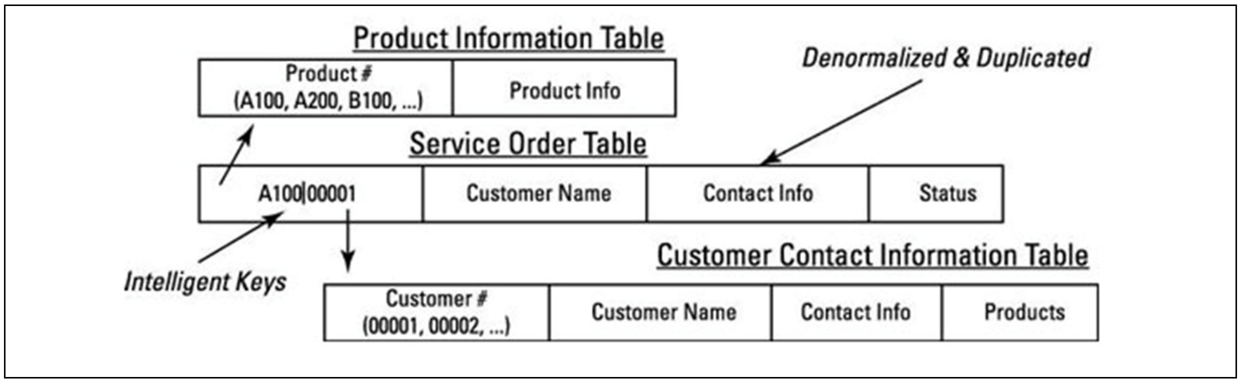


Figure 3.4 HBase Schema for the Service Order Database (deRoos, 2014)

In this proposed solution to the problem, the Customer Contact Information table has been denormalized by including the customer name and contact info in place of the previously used foreign keys. Also, as illustrated in Figure 3.4, the data is duplicated by keeping the Customer Contact Information table as is. Now, joins across the Service Order table and the Customer Contact Information table are not necessary. Additionally, an intelligent row key design has been employed that combines the product number with the customer number to form the service order number (A100|00001, for example). Using this intelligent key, the service order table can provide vital reports about product deficiencies and customers who are currently experiencing product issues.

The example provided by deRoos (deRoos, 2014) illustrates the need for conversion rules. He showed that each use case has to be studied and that for one use case, multiple schema can result, as shown in Figure 3.3 and Figure 3.4. The resulting schema depends on many factors as described in step 1 of Figure 3.1.

3.5 Conclusion

In summary, there are many proposals to resolve the schema conversion problem between relational and non-relational databases. Yet most, if not all, lack focus in explaining the steps to get the best conversion possible that benefits from the NoSQL features and specifications. Instead, they tend to propose keeping the same RDB structure or having the same type of database schema for each and every database conversion. In addition, almost all propositions missed the source database assessment and analysis step and therefore were not able to apply this information in the creation of the target schema database. In our view, we believe there is an opportunity for better conversion instructions of a database schema that ensures the best use of the different NoSQL database characteristics.

The main contribution of this thesis is to propose a list of rules for converting relational database schemas to the schema of a non-relational database, namely we have chosen HBase for our case studies.

The next chapter will present the methodology and activities associated with our research proposal to uncover these conversion rules.

CHAPTER 4

RESEARCH METHODOLOGY, ACTIVITIES AND EXPECTED RESULTS

4.1 Research Methodology

The research methodology used to tackle the research problems described in the Introduction uses empirical research in software engineering (Abran, Laframboise and Bourque, 2003). The framework is composed of four phases: definition, planning, operation and interpretation.

4.2 Definition Phase

The definition phase, depicted in Table 4.1, is composed of the following activities:

- The problem is identified and understood;
- A research topic where an original contribution to knowledge can be made within the timetable of a doctoral program is defined;
- The research question and research sub-questions are formulated;
- A potential solution to the problem is envisioned; and
- The beneficiaries of the research results are identified.

4.3 Planning phase

In the planning phase, a literature review was conducted, covering:

- The cloud computing concept and definitions;
- The state of the art of cloud computing technologies;
- The non-relational, distributed database called HBase state of the art;
- The database conversion between relational and non-relational state of the art.

Table 4.1 Definition Phase

Motivation	Objective	Proposal	Research Users
Clarify the cloud computing definition for our research purposes and how to help software engineers in their first conversion effort to convert a relational database to a non-relational database (NoSQL)	First, to clarify the essential and optional characteristics of cloud computing. Second, to contribute to a methodology that helps engineers in an RDB database conversion to NoSQL database.	• Clarify the types and components in the term cloud computing; • Extract a set of conversion rules that can be used to help software engineers in their first effort to convert an existing RDB to HBase.	Students, researchers and professionals in the information technology field who are interested in cloud computing and its technologies.

An initial research activity aims at clarifying the definitions of the term cloud computing for our research use. The objective is to better describe its essential and optional characteristics as perceived in our research.

The main research objective considers the technology used to implement a cloud computing solution. First, an introduction about the use of non-relational databases, also known as NoSQL database technology, especially HBase, to implement a cloud computing solution is presented. Following this introduction, a second overview chapter is developed to discuss a major issue reported with using this new paradigm technology, i.e. how to convert current relational database applications to a NoSQL technology such as HBase. During this phase, the research methodology for achieving the research objectives and answering the research sub-questions is planned and documented. This includes the research methods and validation activities.

The summary of the planning phase is presented in two parts:

1. Table 4.2 for the cloud computing definition; and
2. Table 4.3 for the RDB to NoSQL conversion rules

Table 4.2 Planning Phase: Cloud Computing Definition

Projects steps	Inputs	Deliverables
Research the different definitions of the term cloud computing	Literature review of: • Best known and available definition of cloud computing • Cloud computing usage model • Types of cloud computing • Other concepts similar to cloud computing (e.g., Grid Computing)	• A list of different definitions of cloud computing and the confirmation that the subject is still an active research topic • Study the optional and essential characteristics of cloud computing • Comparison with other concepts similar to cloud computing. Analogy with a definition of a car
Clarify the cloud computing definition	• Cloud computing properties • Cloud computing services • Essentials characteristics of cloud computing derived from case studies	• Clarified definition of cloud computing for our research • Publication in the *Journal of Cloud Computing* • Publication in the *Encyclopedia of Information Science and Technology*

Table 4.3 Planning Phase: RDB to NoSQL Conversion

Projects steps	Inputs	Deliverables
Review of Big Data technologies suitable to implement a cloud computing solution	Introduction to concepts and technologies: • Relational vs. non-relational database comparison • Review of the Hadoop project, a known cloud computing database • Review of HBase, the Hadoop column-oriented database	• Main strengths and weaknesses of the traditional relational database • Review of the new distributed, non-relational database known to overcome RDB limitations
Review the state of the art on relational database conversion to NoSQL database	Literature review of: • Overview of typical RDB to NoSQL conversion steps • Database conversion state of the art • Other opinions: books, blogs and Web discussions related to recent conversion attempts between RDB and NoSQL databases	• RDB to NoSQL database conversion process • List of conversion attempts between RDB and NoSQL with their evaluation • Literature review discussion summary • Few examples of promising case studies from blogs and Web discussion

The design of a methodology requires the participation of human beings. Therefore, a research protocol for the research activities to be conducted in order to evaluate the a priori version of the conversion rules (with and without the conversion rules) was elaborated and presented to the ÉTS Ethics Committee for Research (CÉR) for its evaluation within the ethics domain.

4.4 Operation Phase

This phase represents the core of the research work and consists of executing the research plan which includes two parts, one for each main deliverable:

1. The proposition of a revised version of the cloud computing definition;
2. The creation of a list of conversion rules.

Each of these parts includes the following steps:

- A validation of the deliverables;
- An analysis of the obtained results (refer to Table 4.4 and
- Table 4.5)

The development of an improved version of the cloud computing definition is based on 1) a review of the literature of the existing definitions, mainly: published articles, websites, cloud service providers and the National Institute of Standards and Technology, 2) the participation at ISO Joint Technical Committee 1, Subcommittee 38 (ISO/JTC/SC38) and 3) the usage of an analogy to support the proposed new cloud computing definition. The analogy used to simplify the cloud computing definition is the definition of a car. This analogy helped clarify the essential and optional characteristics of cloud computing. As stated by Arango et al. (Arango, Domingues, Policarpo and Hermeto, 2002), “the analogy is a powerful tool that enhances our insight. With a single flash, our comprehension of both concepts is instantaneously revealed.”

Table 4.4 Operation Phase: Proposition of a Revised Version of the Cloud Computing Definition

Development of a definition	Validation	Analysis
Development of an initial version of the cloud computing definition	Validation process is based on an analogy with a simple object. Going back to what is mandatory for a simple object's definition, an analogy is used to apply the same principle of mandatory vs. optional for the cloud computing concept.	Submission to various journals and ISO working groups for validation and publication
Development of a reviewed and validated improved version of the cloud computing definition	The reviewed version of the cloud computing definition is validated by a peer review of the cloud computing journal	Minor adjustments completed before final publication of the improved version of the cloud computing definition.

Table 4.5 Operation Phase: Creation of a List of Conversion Rules

Development of a set of conversion rules	Validation	Analysis
Development of a baseline for the conversion rules	The baseline version of conversion rules is validated by conducting an experiment using a heuristic approach. A survey is used after participants achieve a schema conversion without the use of conversion rules.	• Baseline for conversion rules. • Conclusion from the survey about the need to use or not to use a set of conversion rules to help with the conversion problem. • Publication
Rules extraction experiments	Rule extraction is validated by conducting an experiment with participants. A task is assigned to participants to achieve a schema conversion using all possible schema conversions for a single relation.	• Experiment results including new schemas and a set of timed executions for each schema. • A set of rules extracted by analysing the new schemas produced and the execution time.

4.5 Interpretation Phase

Finally, an interpretation will be based on the analysis of all of the findings, in this case the set of conversion rules. This section will also include revision of the research questions by providing a summary of the experimental results, as well as the identification of future work. This phase is summarized in Table 4.6.

Table 4.6 Interpretation Phase

Context	Extrapolation	Future Work
• The revised version of the cloud computing definition contributed to the effort of creating an internationally approved definition • The revised version of the definition of cloud computing identifies what is and what isn't a cloud computing solution • The list of conversion rules helps professionals in their conversion process	• The participants found the usage of the conversion rules helpful to conduct a schema conversion from a relational to non-relational database • Several ÉTS students collaborated with the project • The revised version of the definition of cloud computing as well as the schema conversion rules have been well received by journals for publications	• Further publications of the results in journals • Improve the revised version of the cloud computing definition for an internationally accepted definition • Conduct another experiment using the list of conversion rules for validation and compare with the baseline • Conduct more surveys with the list of conversion rules • Add and improve the list of conversion rules • Test the list of conversion rules with non-relational databases other than Hadoop HBase database • Generalize the list of conversion rules to be applied to all column-oriented databases

4.6 Summary of the Research Methodology

In this chapter, the research methodology was detailed using an empirical research approach in software engineering (Abran et al., 2003), explaining each step with the input and output of each activity as shown in Figure 4.1.

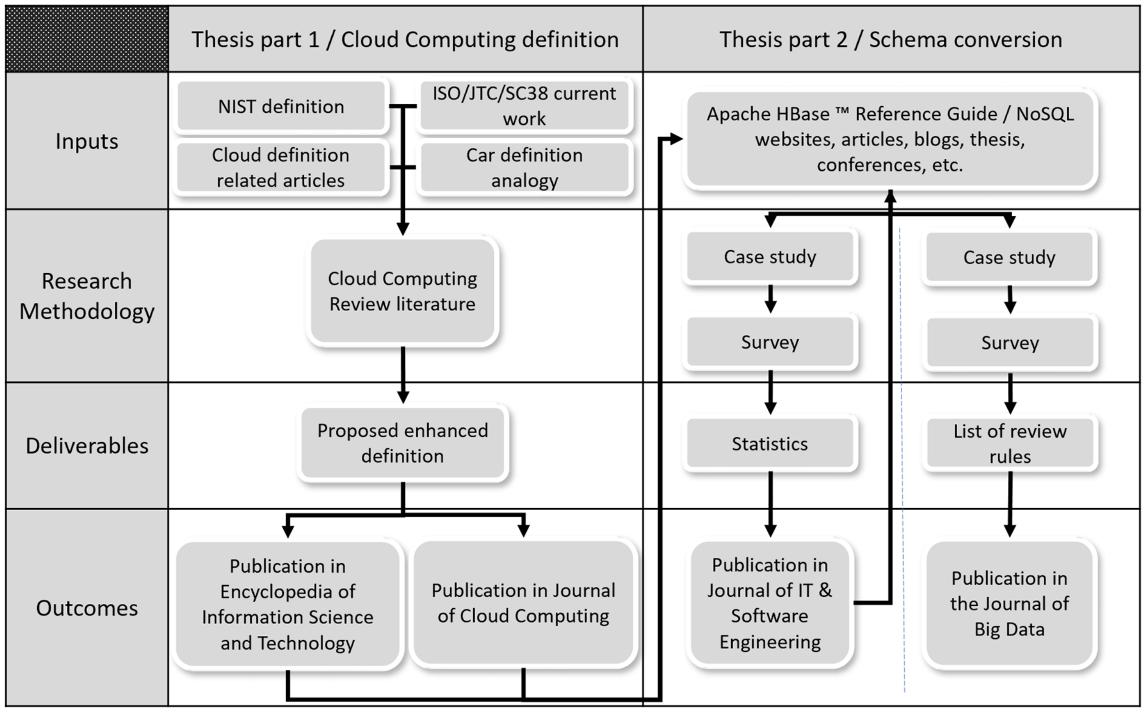

Thesis part 1 / Cloud Computing definition
Thesis part 2 / Schema conversion
Inputs
Research Methodology
Deliverables
Outcomes
NIST definition
ISO/JTC/SC38 current work
Cloud definition related articles
Car definition analogy
Apache HBase ™ Reference Guide / NoSQL websites, articles, blogs, thesis, conferences, etc.
Cloud Computing Review literature
Case study
Survey
Case study
Survey
Proposed enhanced definition
Statistics
List of review rules
Publication in Encyclopedia of Information Science and Technology
Publication in Journal of Cloud Computing
Publication in Journal of IT & Software Engineering
Publication in the Journal of Big Data

Figure 4.1 Summary of the Research Methodology

Chapter 5 describes the research activities conducted to clarify the cloud computing definition for our research by using an analogy with the definition of a car to highlight the notions of standard and optional features.

Chapter 6 presents the main research activities of this thesis, extracting, through experimentation, a list of conversion rules to help software engineers with their first database schema conversion (from a relational database to a NoSQL columnar database). Two experiments are conducted, the first aims at validating that there is indeed a need for such conversion rules and a second to uncover an initial version of these conversion rules.

CHAPTER 5

THE CLOUD COMPUTING DEFINITION

5.1 Introduction

This chapter first discusses the ISO and NIST international definition of cloud computing and its notion of key and essential characteristics. Second, a discussion is presented about the essential and optional characteristics raised by the perspective of this research (i.e. database conversion from RDB to NoSQL). Finally, an analogy is used to present useful clarifications of this term.

5.2 ISO and NIST Cloud Computing Definitions

The ISO Joint Technical Committee 1, Subcommittee 38 (ISO/JTC/SC38), is focused on Distributed Application Platforms and Services (DAPS) including a working group (WG3) which is responsible for identifying, developing, and maintaining Joint Technical Committee (JTC 1) deliverables in the field of cloud computing and is working with many Standards Development Organizations (SDO's) such as the National Institute of Standards and Technology (NIST). The first ISO/JTC1 report defined cloud computing as follows: "Cloud computing provides IT infrastructure and environment to develop/host/run services and applications, on demand, with pay-as-you-go pricing, as a service. It also provides resources and services to store data and run applications, on any device, anytime, anywhere, as a service." (ISO/IEC, 2009) It also provided a list of what should be considered as key characteristics presented in Table 5.1.

Table 5.1 Cloud Computing Key Characteristics (ISO/IEC, 2009)

Key Characteristics	Description
Agility	Agility improves with users who are able to rapidly and inexpensively re-provision technological infrastructure resources. The cost of overall computing is unchanged, however, and the providers will merely absorb upfront costs and spread costs over a longer period.

Table 5.1 Cloud Computing Key Characteristics (ISO/IEC, 2009) (Continued)

Key Characteristics	**Description**
Cost	Cost is claimed to be greatly reduced and capital expenditure is converted to operational expenditure. This ostensibly lowers barriers to entry, as infrastructure is typically provided by a third-party and does not need to be purchased for one-time or infrequent intensive computing tasks. Pricing on a utility computing basis is fine-grained with usage-based options and fewer IT skills are required for implementation (in-house). Some would argue that given the low cost of computing resources, the IT burden merely shifts the cost from in-house to outsourced providers. Furthermore, any cost reduction benefit must be weighed against a corresponding loss of control, access and security risks.
Device and location independence	Device and location independence enable users to access systems using a Web browser regardless of their location or what device they are using (e.g., PC, mobile). As infrastructure is off-site (typically provided by a third-party) and accessed via the Internet, users can connect from anywhere.
Multi-tenancy	Multi-tenancy enables sharing of resources and costs across a large pool of users thus allowing for: • Centralization of infrastructure in locations with lower costs (such as real estate, electricity, etc.) • Increase of Peak-load capacity (users need not engineer for highest possible load-levels) Utilization and efficiency improvements for systems that are often only 10–20% utilized.
Reliability	Reliability improves through the use of multiple redundant sites, which makes Cloud computing suitable for business continuity and disaster recovery. Nonetheless, many major Cloud computing services have suffered outages, and IT and business managers can at times do little when they are affected.
Scalability	Scalability via dynamic ("on-demand") provisioning of resources on a fine-grained, self-service basis near real-time, does not impose users to engineer for peak loads. Performance is monitored and consistent and loosely-coupled architectures are constructed using Web services as the system interface.

Table 5.1 Cloud Computing Key Characteristics (ISO/IEC, 2009) (Continued)

Key Characteristics	Description
Security	Security typically improves due to centralization of data, increased security-focused resources, etc., but concerns can persist about loss of control over certain sensitive data and verification of users' identity. Security is often as good as or better than traditional systems, in part because providers are able to devote resources to solving security issues that many customers cannot afford. Providers typically log accesses, but accessing the audit logs themselves can be difficult or impossible. Ownership, control and access to data controlled by "Cloud" providers may be made more difficult, just as it is sometimes difficult to gain access to "live" support with current utilities. Under the Cloud paradigm, management of sensitive data and other security-related functions (e.g., user's enrollment and verification of user's identity) could be placed in the hands of Cloud providers and third parties.
Sustainability	Sustainability comes about through improved resource utilization, more efficient systems, and carbon neutrality. Nonetheless, computers and associated infrastructure are major consumers of energy. A given (server-based) computing task will use X amount of energy whether it is on-, or off-site.

The ISO definition of Cloud computing must have all the key characteristics listed in Table 5.1 to be considered as such. As described in Chapter 1, the NIST definition of Cloud Computing follows the same description pattern as the one proposed by ISO, i.e. it starts with a short and quite dense description of what cloud computing is. The description is mainly composed of essential characteristics that are listed in some level of detail and concludes by presenting a list of service models, for example, everything as a service where any service can be offered on the cloud. This definition is described as follows: "Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model promotes availability and is composed of five essential characteristics, three service models, and four deployment models." (Mell & Grance, 2011). Similar to the ISO/JTC1 report, NIST has defined a set of essential characteristics as defined in Table 5.1.

Table 5.2 NIST Essential Characteristics

Essential characteristics	**Description**
On-demand self-service	A consumer can unilaterally provision computing capabilities, such as server time and network storage, as needed automatically without requiring human interaction with each service's provider.
Broad network access	Capabilities are available over the network and accessed through standard mechanisms that promote use by heterogeneous thin or thick client platforms (e.g., mobile phones, laptops, and PDAs).
Resource pooling	The provider's computing resources are pooled to serve multiple consumers using a multi-tenant model, with different physical and virtual resources dynamically assigned and reassigned according to consumer demand. There is a sense of location independence in that the customer generally has no control or knowledge over the exact location of the provided resources but may be able to specify location at a higher level of abstraction (e.g., country, state, or datacenter). Examples of resources include storage, processing, memory, network bandwidth, and virtual machines.
Rapid elasticity	Capabilities can be rapidly and elastically provisioned, in some cases automatically, to quickly scale out, and rapidly released to quickly scale in. To the consumer, the capabilities available for provisioning often appear to be unlimited and can be purchased in any quantity at any time.
Measured service	Cloud systems automatically control and optimize resource use by leveraging a metering capability1 at some level of abstraction appropriate to the type of service (e.g., storage, processing, bandwidth, and active user accounts). Resource usage can be monitored, controlled, and reported, providing transparency for both the provider and consumer of the utilized service.

The authors of the NIST cloud computing definition underline the difficulty in clearly defining this term, stating that "cloud computing is still an evolving paradigm" (Mell & Grance, 2011) and later concluding that the definition, its attributes and characteristics will evolve over time (Mell & Grance, 2011).

The intended NIST definition audience is people adopting the cloud computing model or providing cloud services that have highlighted, similar to the ISO/JTC1 report, the difficulty in establishing a clear and consensual list of essential characteristics based on real use cases.

The main weakness of this first ISO/JTC1 report (ISO/IEC, 2009) was the absence of use case and justifications regarding why each characteristic should be considered as essential.

The next section presents a clarification of the NIST definition as well as an analogy to practically show each proposed characteristic (mandatory and optional) as perceived when converting a relational database to a NoSQL database.

5.3 The Car Analogy

When reviewing the ISO definition (ISO/IEC, 2009) and the NIST definition of cloud computing (Mell & Grance, 2011), we notice that these two proposals have something in common; the quest to find an international consensus on which characteristics are either mandatory or optional. This section attempts to answer the question using the analogy of a long-time established definition, that of a car.

The automobile, which is referred to as a car today, is defined by Fowler (Fowler, 2001) as "a wheeled motor vehicle used for transporting passengers, which also carries its own engine or motor." Most existing definitions of a car agree on the following characteristic: "The car is designed to run primarily on roads, to have seating for one to 8 people, and to be constructed principally for the transport of people rather than goods."

Analyzing the above definition, in connection with the challenge of defining a clearer cloud computing definition, we note that the car, as described by Fowler, did not include any options, such as air conditioner, warranty, equipment features, financing modes, etc. The definition is restricted to the car's essential characteristics, such as wheels, motor, and seats for transporting people, as shown in Figure 5.1. Could this analogy be used to clarify the current NIST definition of cloud computing in our research context?

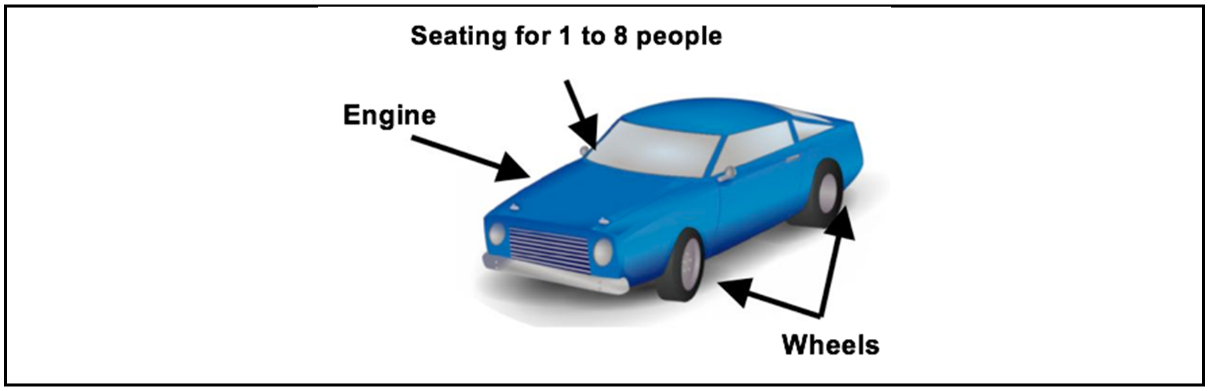


Figure 5.1 Illustration of the Essential Characteristics of a Car based on its Definition

As with the car, by identifying the essential and optional characteristics of cloud computing, we could, for example, study why organizations have recently converted existing applications to cloud computing technology, and, in particular, why they did not use another competing or existing technology instead. This approach may help us identify optional characteristics currently listed in the NIST definition.

5.3.1 Common Factors Observed when Converting to Cloud Computing

After considering many software engineering case studies that have been published involving the conversion by organizations of their applications to cloud computing technology, the main objective has mostly been to continue serving their constantly growing number of customers. In this study, we are looking for the factors in the conversion process that are common to all of these organizations based on their answers to the following questions: What problems had they been experiencing with their current technology? What limitations forced them to convert to the cloud? What essential characteristic were they looking for in a cloud computing technology?

Vendors like Google, Amazon, and others claim that their cloud computing technology provides many advantages like scalability, maintainability, performance, and reliability. Cloud computing technologies have been designed for growth, low latency, and robustness against failures (Jeffrey Dean, 2009). Cost is also raised as an important factor. For example, a great deal of attention has been paid to the low cost of running Google server hardware, with its built-in batteries that eliminate the need for a huge, centralized uninterrupted power supply (UPS), and their customized 12-volt power supplies that deliver energy efficiency.

Let’s first discuss the requirements for converting to cloud computing technology as seen in the literature. In the case of YouTube, the challenge mostly had to do with scalability and performance, since keeping up with website growth was a day-to-day battle for them before the company was bought by Google and their technology was converted to the cloud (i.e. Google proprietary technology (BigTable)). BigTable is a non-relational, high performance database that is fault tolerant over large sets of data, which solves the famous YouTube thumbnails issue as described by Cordes (Cordes, 2007). A second case study reveals how Twitter described how they scaled horizontally using cloud computing technologies (Higginbotham, 2011) and how they have managed to make their service 10,000 times faster than it was before (Cook, 2009). According to database pioneer Michael Stonebreaker (Harris, 2011), Facebook was stuck using its current technology, especially its relational database, which was rapidly growing to 4,000 shards and becoming extremely complex to manage. Database sharding is a partitioning scheme for large databases across a number of servers, enabling new levels of database performance and scalability (Agildata, 2011). Stonebreaker adds that this is fairly common practice, especially among start-ups, that start small and grow to epic proportions, and use Web-based applications where users can upload whatever they want. Most of the case studies we found involve organizations that offer or use services that had to convert to cloud computing because of their growing scalability, performance, or maintainability issues.

When forced to convert out of an existing technology, organizations look carefully at all the possible alternatives. Target technologies are typically assessed for each of the key non-functional requirements: scalability, maintainability, performance, and reliability. As Harris (Harris, 2011) points out, relational database technology is struggling to scale to handle petabyte-sized tables where NoSQL technology is an increasingly attractive solution to relational database scalability issues.

These case studies reveal that cloud computing is not only concerned with hosting software elsewhere, but also with networking computers to distribute processing power in newer ways. Table 5.3 summarizes the essential characteristics that are common in the publications mentioned in this section and are key in the decision to convert to cloud computing technology for each non-functional requirement that presents a serious problem.

Table 5.3 Essential Characteristics of Cloud Computing Derived from Case Studies (Jeffrey Dean, 2009)

Non-functional requirement	**Problem with the use of current technology**	**Case Study Solution provided by cloud computing technology**
Scalability	Limits database growth across servers	Plug and Play of new equipment with distributed, horizontally scalable NoSQL or hybrid database using both non-relational and relational database.
Maintainability	Managing the sharded clusters of server table requires a great deal of manpower	Minimal management effort required to change software/hardware components
Performance	Limit to affordable performance offered by database and IT	Good performance of NoSQL database accommodating petabytes of data on a shared pool of computers
Reliability	Limits to available and affordable recovery option in current IT infrastructure	Task replication and continuity ensured by cloud computing

Using these essential characteristics, a clarification of the current NIST definition is considered for our research purposes.

5.3.2 NIST Definition Clarifications for this Research

Section 5.3 show the definition of a car that includes only its essential characteristics and none of the optional ones. In a car, an option can be purchased and added to the base product, and so is not considered as part of its definition. Options are typically similar for a type of vehicle, a vendor, or in the car industry as a whole. The same approach is used to improve the NIST cloud computing definition. Table 5.4 displays our attempt to draw a parallel between the definition of a car and that of cloud computing, including the smallest possible set of essential characteristics.

Table 5.4 The Car vs. Cloud Computing Analogy for a Reduced Set of Essential Characteristics

Car definition	Proposed Cloud Computing definition
A car is a wheeled motor vehicle used for transporting passengers	Cloud computing is an approach to IT designed to deliver services reliably.
A car incorporates its own engine or motor.	Cloud computing operates on a scalable and shared pool of resources that can be rapidly and elastically provisioned. These resources are perceived to be exclusive when consumed by multiple cloud computing tenants.
A car is designed primarily to run on roads.	Cloud computing is designed primarily to be delivered over a network.
A car has seating for 1 to 8 people.	Cloud computing ensures good performance, even with petabytes of data.
A car is constructed principally for the transport of people rather than goods.	Cloud computing is designed principally to serve a growing and theoretically unlimited number of users.

Given a proposal for essential characteristics, the same approach is used for determining optional characteristics, as presented in Table 5.5. In some cases, optional characteristics may be required by the client and their selection depends upon what the organization is looking for as it converts its applications to cloud computing. In this proposal, optional characteristics are distinguished from essential characteristics only in the fact that they should not be included in the main definition of the term. Table 5.5 identifies the concepts (i.e. characteristics) currently included in the NIST definition that could be considered as optional characteristics in this research.

Table 5.5 The Car vs. Cloud Computing Analogy for Optional Characteristics

Concepts excluded from the car definition	**Concepts to exclude from the cloud computing definition**	**Concepts that could be excluded from the list of NIST essential characteristics**
Financing model, leases, loans, and rentals	Payment model or billing method used by the cloud computing provider	Measured service: Resource usage that is monitored, controlled, and reported, providing transparency for both the provider and the consumer of the service
Options	On-demand options	On-demand self-service: Computing capability, such as server time and network storage, unilaterally provisioned by a consumer as needed, and provided automatically, without human interaction with the service provider
Accessibility	Thin client access through any browser	Broad network access: including the use of cloud computing by heterogeneous thin or thick client platforms, such as mobile phones, laptops, and PDAs
Post-sale service, type of vehicle	Services and deployment models	Service models, such as SaaS, PaaS, and IaaS, and deployment models, such as private, community, public, and hybrid clouds

Table 5.4 and Table 5.5 propose to separate essential and optional cloud computing characteristics. Revisiting the NIST cloud computing definition with this new perspective, it is noted that some of the proposed essential characteristics may be not essential in our context, based on this car analogy and the perspectives of the case studies of Table 5.5. For example, the broad network access presented in the NIST definition that introduces the notions of the thick client and the thin client (i.e. mobile phone, laptop, PDA) could be optional for many organizations, as their needs may not be focused on the way they access a cloud service. It goes without saying that service access can evolve over time, as new devices appear and gain in popularity. In contrast, scalability seems to be a common expectation that organizations have of cloud computing services, as described in Table 5.3, and so it should be kept in the main definition.

The proposed clarified definition for this research is described as follows: "Cloud computing is an approach to IT designed to deliver services reliably and operates on a scalable and shared pool of resources that can be rapidly and elastically provisioned. These resources are perceived to be exclusive when consumed by multiple cloud computing tenants. Cloud computing is designed primarily to be delivered over a network and ensures good performance even with petabytes of data because it's designed principally to serve a growing and theoretically unlimited number of users." The enhancements are displayed in Table 5.6.

Table 5.6 NIST vs. Proposed Enhancements

NIST definition of cloud computing	**Clarifications of cloud computing concepts for this research**	**Discussion**
Cloud computing **is a model**	Cloud computing **is an approach to IT**	Models are defined by NIST as SAAS, PAAS and IAAS; these services can evolve and multiply with technology advancement where an approach is stable once defined.
Designed to enable **ubiquitous**, **convenient**, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services)	Designed to deliver services **reliably** and operates on a **scalable** and shared pool of resources	Ubiquitous and convenient can be left as optional since a cloud service can be offered on a limited type of devices. Reliable and scalable are found to be essentials in use cases where organization opt to convert to cloud computing.
that can be **rapidly provisioned** and released with minimal management effort or service provider interaction.	that can be **rapidly** and **elastically provisioned**.	Minimal management effort or service provider interaction can be mentioned as an optional characteristic as none of the use cases in the literature mention key characteristics.

Table 5.6 NIST vs. Proposed Enhancements (Continued)

NIST definition of cloud computing	**Clarifications of cloud computing concepts for this research**	**Discussion**
This cloud model promotes **availability** and is composed of five essential characteristics, three **service models**, and four deployment models.	These resources are perceived to be exclusive when consumed by multiple cloud computing tenants. Cloud computing is designed primarily to be delivered over a network and ensures good **performance** even with petabytes of data because it's designed principally to **serve a growing and theoretically unlimited number of users**.	Service models can be described as optional (as discussed in the first row of this table) and three service models should be mentioned as three examples of service models since these services can evolve and new ones can appear with technology advancement. Essential characteristics as described by NIST and in Table 5.2 are included in the NIST definition except for 'Measured Service'. This characteristic can also be described as optional since cloud service can be offered without limit. An unlimited number of users, on the other hand, should be included as a mandatory characteristic as it is critical for customers not to be limited in the number of users as per the use cases described in section 5.3.1.

5.4 Conclusion

This chapter considers the NIST definition of cloud computing from the research point of view of relational database conversion to NoSQL, using a car analogy to reassess its essential and optional characteristics. The ISO and NIST definitions were presented and discussed. Finally, using an analogy with the definition of a car and with the help of multiple use cases, a clarified version of the NIST definition that is better suited for this research is proposed.

In the next chapter, the first and main contribution of this thesis is presented; how to convert an application to cloud computing. More specifically, how to convert a relational database to a NoSQL database—a technology known to be suitable for cloud computing applications.

CHAPTER 6

RDB TO NOSQL CONVERSION PROBLEM

6.1 Background

The main research contribution of this thesis is to extract and propose a set of conversion rules that can be used to help software engineers in their first effort to convert an existing RDB to HBase, and in particular the column-oriented database named HBase. First, an experiment is conducted to justify the need for conversion rules by asking participants to convert an existing application to HBase without any guidelines. This first experiment served as a baseline for experiment 2. The second experiment is conducted to uncover an initial list of conversion rules by having participants conduct an actual conversion of an existing application to HBase using three possible conversion design patterns (mainly one-to-one and one-to-many relationships) as shown in Figure 6.1.

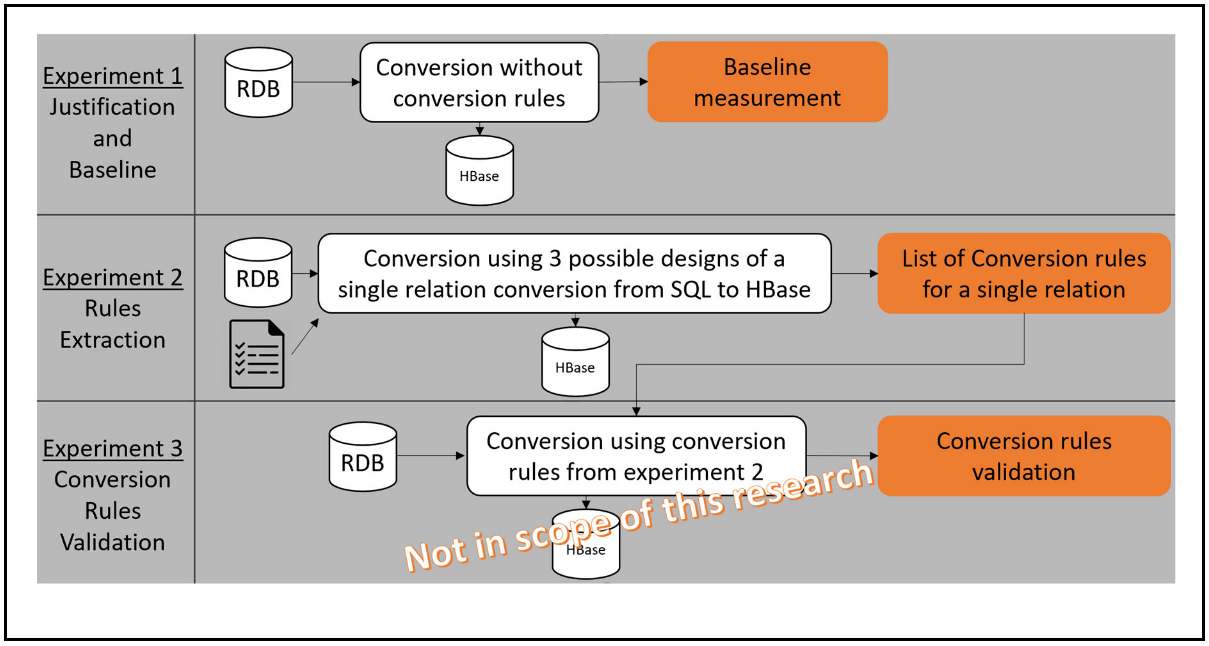

Experiment 1
Justification and Baseline
RDB
Conversion without conversion rules
Baseline measurement
HBase
Experiment 2
Rules Extraction
RDB
Conversion using 3 possible designs of a single relation conversion from SQL to HBase
List of Conversion rules for a single relation
HBase
Experiment 3
Conversion Rules Validation
RDB
Conversion using conversion rules from experiment 2
Conversion rules validation
HBase
Not in scope of this research

Figure 6.1 The Scope of this Research vs. the Overall Research Objective

Chapter 6 is structured as follows. Section 6.2 presents the HBase schema basics and design fundamentals. Section 6.3 describes the experiments: the first experiment with participants performing a conversion simply based on their experience tries to simulate how an individual would go about such a conversion without any guidelines; next, three RDB to HBase conversion design patterns are presented and offered for use in the second conversion experiment. Finally, Section 6.4 presents the main conclusions and future improvements.

6.2 HBase Schema Basics and Design Fundamentals

This section presents some of the key terms and concepts as well as some design fundamentals about HBase technology.

An RDB schema is composed of tables, columns and relationships between the tables: one-to-one, one-to-many and many-to-many. Such relationships do not exist in an HBase schema (Lars, 2013b). The HBase terminology refers to a row key, a column and a column family. To facilitate the understanding of this paper, the term "HTable" will refer to an HBase Table and the term "Table" will refer to any relational database table.

6.2.1 Row Key Design in HBase

The selection of the row key in an HTable is one of the most critical activities for a successful schema design (Lars, 2013a). A row key is used to fetch data, which means that when a row key is carefully selected, its rows are sorted in a way that regroups the data so as to be accessed together, ensuring a more efficient query (Lars, 2013a). A badly chosen row key could spread the data across the HTable and make fetching of the data time consuming and result in poor performance. Therefore, the selection of a row key is always based on the context of the table, as opposed to the relational model, where the primary key plays this important role. Typically, the primary key is unique in a table but this does not usually allow for grouping of data. Grouping can be performed using SQL statements such as ORDER BY on data contained in other columns (e.g., SELECT * FROM BOOKS ORDER BY AUTHORS).

6.2.2 Columns and Column Family in HBase

One of the first comparisons that can be made when converting a table to an HTable is the use of columns. The notion of a column in an HTable is quite different from that of a column in an RDB table. In an RDB table, a column is meant to hold a single piece of data type whereas a column, or rather a column family, in an HTable can contain multiple pieces of data via a key-value pair. A column family in an HTable should be considered as a container of key-value pairs. The column family is determined by the information in each row and it regroups key-value pairs of related data. For example, the column family “address” contains the following columns: house number, street name, city, province, country and postal code. One of the major differences between columns in an RDB and an HBase column family is that there is no strict definition for HBase columns and they can be added dynamically when needed. Therefore, if the address is different from one country to another, new columns can be added to the column family when needed. This flexibility would be very hard to build into an address table using relational databases, as we would need to modify the table in the future if new columns were needed. The nature of a column family lends itself well to this type of situation since the HTable will not need modifications to accommodate the differences in addresses. However, this imposes a restriction on the conversion of an RDB table, as it cannot be faithfully recreated in NoSQL.

6.2.3 HBase Design Fundamentals

RDB use three types of relationships between tables: one-to-one, one-to-many and many-to-many. The first two are designed using one table containing a foreign key that references the second table. As for the many-to-many relationship, it requires a third intermediate table that holds references to the other two tables, which can ultimately be broken down into two one-to-many relationships between the first and third table and between the second and third table. Therefore, the many-to-many relationship is simply a combination of one-to-many relations between three tables, which also means that there is no direct link between the first and second table as opposed to the one-to-one and one-to-many relationships. Thus, for conversion purposes, we can only consider a relation that has a direct link between two tables.

In HBase, it is quite different since such relationships do not exist. Without relations, the focus when creating or converting an RDB schema to HBase is to define the access pattern upfront and ask the important questions (also known as “what is your use case”), namely what is accessed, how is it accessed and when is it accessed. The following are well-known methods to convert a single RDB relationship: converted to either a one HTable or a two HTable design. A one HTable design results in two possible conversions: the first would be a merge of both RDB tables into one column family and the other would be to create two column families, one for each RDB table. The two HTable design conversion closely resembles its RDB table counterpart, as each HTable will hold one column family for each RDB table.

An HBase design schema can be summarized as follows:

1. Generally, all the data that are accessed at the same time should be grouped into a single column family to ensure that searching is efficient. Therefore, this would be represented in a single table containing a single column family. Otherwise, if the data being accessed that need to be close together are accessed at different points in time, it could be represented with a single table but with two column families.
2. If the data are related to each other but can be accessed independently, then two HBase tables having one column family each should be created.
3. Physically, column families are stored on a per-column family basis. Therefore, when accessing two column families in the same HTable, HBase will access two separate physical files.
4. Lastly, it is recommended to limit the number of column families within a table. Only introduce a second or third in the case where data access in usually column scoped.

6.3 Experiment Description and Results

6.3.1 Experiment 1

6.3.1.1 Experiment 1 Goal

The first experiment was conducted to investigate the need for schema conversion rules by asking participants to convert an existing RDB database to HBase without any guidelines (Gomez, Ouanouki, April and Abran, 2014).

6.3.1.2 Experiment 1 Description

This subsection describes the experimental design used according to the structure recommended by the following authors: (Easterbrook, Singer, Storey and Damian, 2008), (Marcos, 2005), and (Zelkowitz, Wallace and Binkley, 2012). This first experiment was conducted using a heuristic approach; the participants use their experience, educated guesses or plain common sense to do the conversion. The material provided to the participant for the execution of the experiment was:

1. A tutorial session and document (See Appendix I);
2. Participant's instructions guide (See Appendix VI);
3. An RDB schema to be converted (Figure 6.2);
4. A survey questionnaire (See Appendix III).

The tutorial session included a training document, which summarized the content of the tutorial session. These documents were distributed to each participant beforehand and included a summary of RDB and NoSQL concepts with practical examples of each. Then a 2-hour tutorial was conducted, going through the document step-by-step and answering questions.

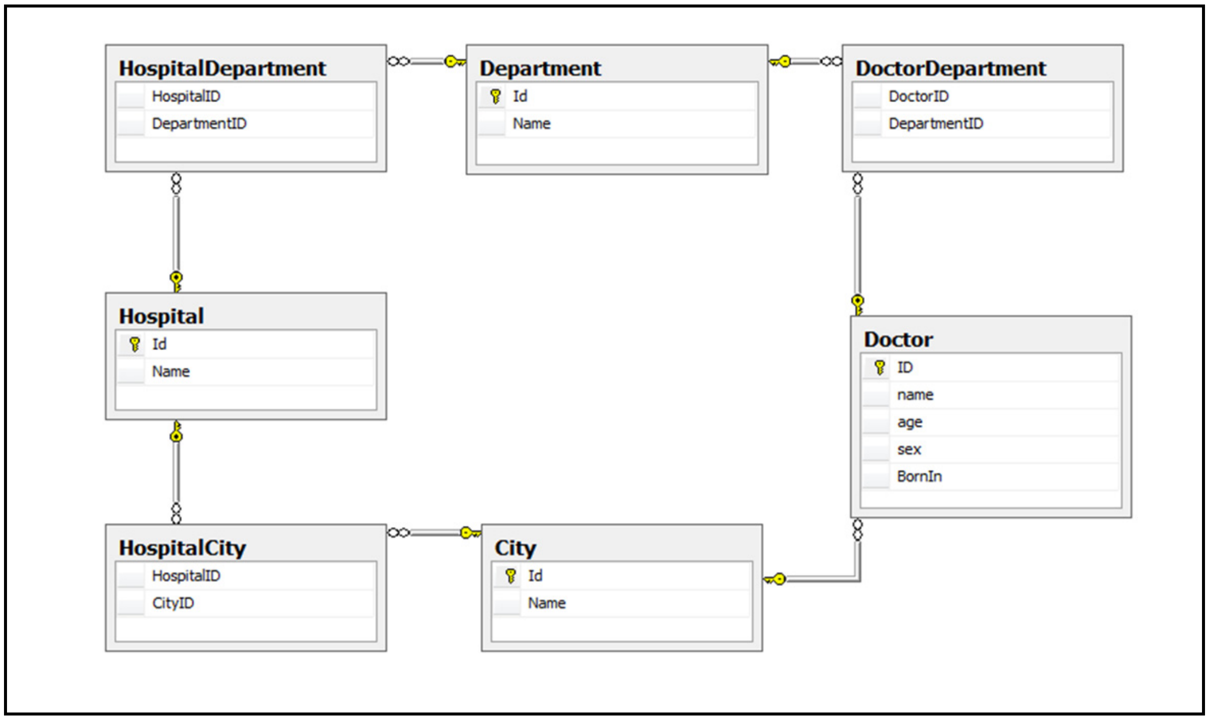


Figure 6.2 Relational Schema Given to the Participants

The participant guide included instructions and booklets of worksheets that the participant used during the experiment. One of these worksheets is entitled "The NoSQL solution" where the participant was asked to draw a representation of his proposed HBase schema (the target schema) for the conversion.

Figure 6.2 shows the RDB schema that had to be converted to NoSQL by the participants. This schema is the one already used in Singh's experiment (Tarandeep & Parvinder, 2011) and is composed of seven relational tables, where:

- Four are large tables (e.g., City, Department, Doctor and Hospital), and
- Three are junction tables (e.g., DoctorDeparment, HospitalCity and HospitalDepartment,).

The City, Department, Doctor and Hospital tables have an "Id, Name" structure, with "Id" as the primary key. The "Doctor" table contains "Id, Name, Age, Sex and BornIn". The last field is the "Id" of the city where the doctor was born. The junction tables allow for the expression of the “many-to-many” relationships indicated by each junction table’s title. It is important to note that each participant was offered the opportunity to choose between several sub-schemas from the main schema. For instance, a participant could choose only to convert the sub-schema composed of the entities Hospital – HospitalDepartment – Department or the sub-schema Doctor – DoctorDepartment – Department or the participant could select the entire schema. The five key aspects of this RDB conversion that best represent the items that must undergo conversion are: Tables, Constraints, Primary Keys (PK), Foreign Keys (FK) and other elements (such as fields, types of relationships, views, indexes, procedures and triggers).

Finally, a survey form was given to each participant after the experiment. The survey was designed and validated following the recommendations of Kasunic (Kasunic, 2005) and Lethbridge (Lethbridge, 1998). It contained 9 questions where the first four questions were oriented toward the “experience” of the participants:

1. The first question established the academic background with 4 possible answers: Graduate with PhD, Graduate with Master, Graduate with Bachelor or Undergraduate Student;
2. The second question enquired about the working status of the participant and had only 2 possible responses: working in industry or currently in academia;
3. The third question collected data about the number of years of work experience using RDB technology with 4 possible answers: no experience, little or low level of experience (e.g., a few days to one year), average amount of experience (categorized as from 2 to 5 years), and advanced experience (e.g., very good knowledge with more than 5 years of use);
4. The fourth question asked if the participant had any NoSQL experience and captured their answer using the same 4 answer choices as above;
5. The fifth question concerned the conversion process and was especially focused on reporting the first step that was used to initiate the conversion process;
6. The sixth question asked to report the amount of effort needed to perform the conversion (without the use of rules to guide the process). This question could be answered using values from 1 to 5, where 1 indicated that the process was easy to

achieve without major effort, a value of 3 indicated that it was achieved using a large amount of effort and a value of 5 meant that no matter how much effort was put in, it was not possible for the participant to successfully perform the conversion;

7. The seventh question was designed to evaluate the level of confusion experienced by the participant during the conversion process (e.g., no idea where to start or what the next step was). This question had 5 possible answers: 1) always confused during the process, 2) very often confused, 3) sometimes confused, 4) rarely confused, and 5) never confused;
8. The eighth question was presented in the form of a matrix to assess the conversion percentage achieved of the proposed solution regarding each of the relational aspects mentioned earlier (e.g., Table, Constraint, PK, and FK);
9. The ninth and last question asked the participant if having access to conversion guidelines would have improved the conversion process. This question had five possible answers: 1) strongly agree, 2) agree, 3) undecided, 4) disagree, and 5) strongly disagree.

Finally, eighteen participants participated in the conversion experiment, filled out a NoSQL solution sheet and expressed their opinions by completing the survey.

6.3.1.3 Experiment 1: Data and Analysis

Table 6.1 presents the educational level of the participants who were mostly working in industry and enrolled part-time in software engineering university programs: 6% of participants were taking an undergraduate course and 89% of participants were taking graduate courses (39% at the Bachelor level and 50% at the Master level).

Table 6.1 Educational Level of the Participants (N=18)

Educational Level		
Classification	*Response in percentage*	*Response in number*
PhD	5%	1
Master	50%	9
Bachelor	39%	7
Undergraduate	6%	1

Table 6.2 shows that 83% of the participants were currently working in industry: this is aligned with our goal in trying to simulate what actual software engineers, in the industry, would experience during such a conversion.

Table 6.2 Work Area of the Participants

Work Area	
Classification	*Response in percentage*
Industry	83%
Academic	17%

It can also be observed, in Table 6.3, that a great number of participants had previous experience with RDB technology and that 45% of them had more than 5 years of experience with RDB technology. As expected, Table 6.3 also shows that 94% of the participants had no previous knowledge of NoSQL database technology.

Table 6.3 Level of Experience in DB

Type of DB	Experience in years			
	No Exp	*Low Exp (< 1 Year)*	*Middle Exp (2-5 Years)*	*Advanced Exp (>5 Years)*
RDB	22%	11%	22%	45%
NoSQL	94%	0%	6%	0%

The experience profile of the participants indicates that a set of guidelines could be a valuable tool for practitioners in such conversion processes. Figure 6.3 shows the different approaches taken by the participants for the first step of their conversion. Considering that most participants had industry experience, Figure 6.3 indicates that 61% (i.e. 33% + 28%) of the participants chose to begin with the RDBMS “tables” element for the conversion.

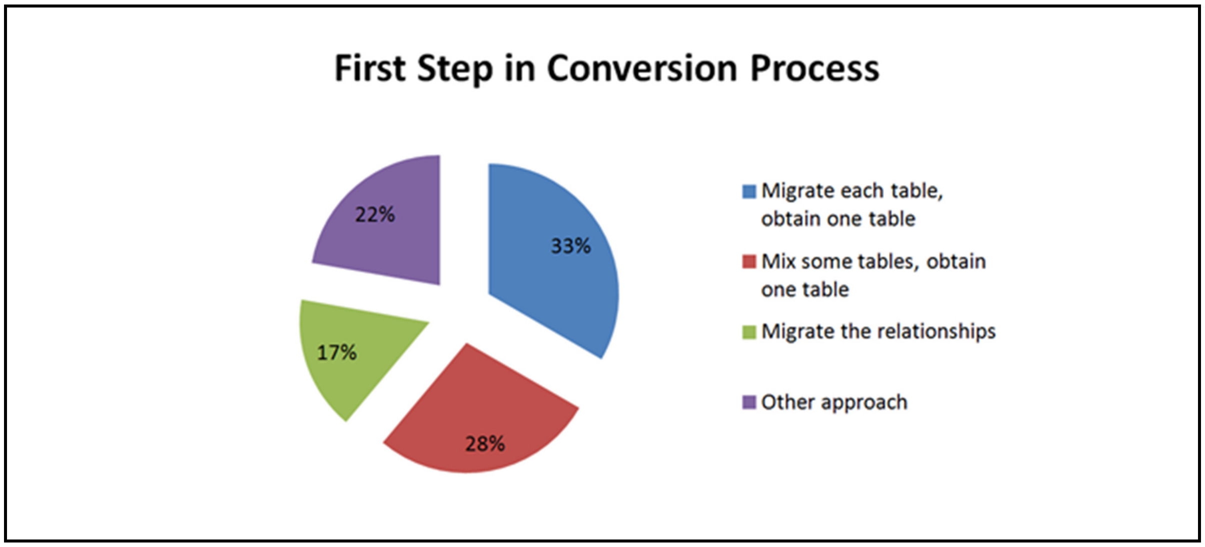


Figure 6.3 First Step in the Conversion Process

The next criterion analyzed is the perceived difficulty encountered during this conversion process. Depicted in Figure 6.4, the initial perception that this conversion is difficult was reported by 78% of the participants (i.e. 39% + 39%). The participants were of the opinion that this conversion process demanded a considerable amount of effort. This result is consistent with individuals who had very little exposure to NoSQL concepts beforehand (as reported in Table 6.3).

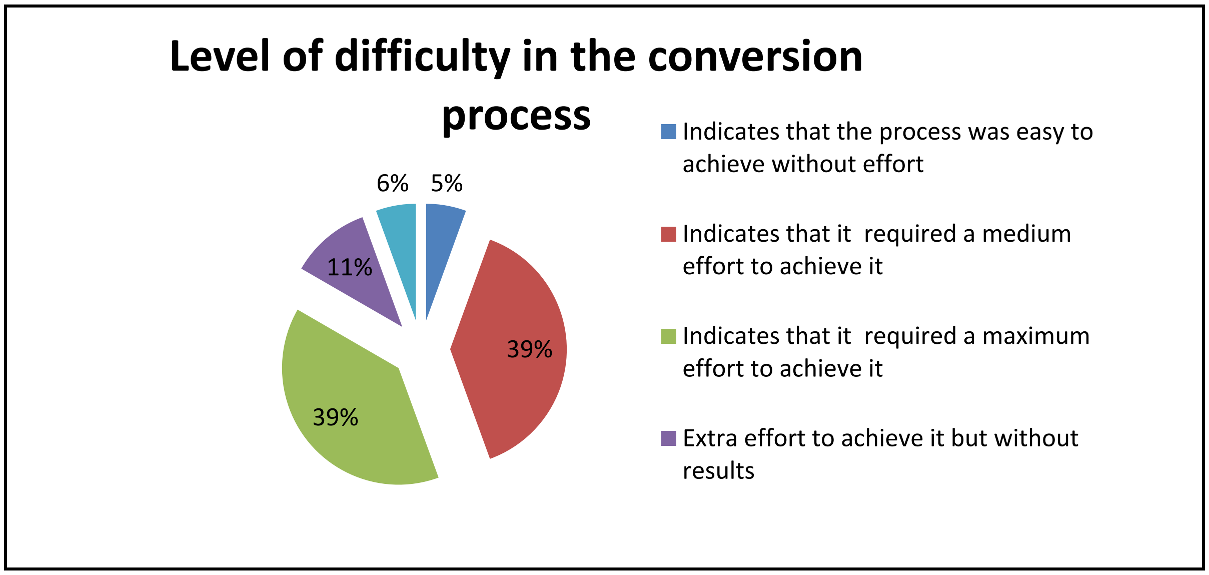


Figure 6.4 Level of Difficulty in the Conversion Process

Finally, Figure 6.5 evaluates the level of confusion experienced by the participants during this conversion process. It reports that the majority of the participants (56%) had felt sometimes or always confused, i.e. not knowing how to go about the process.

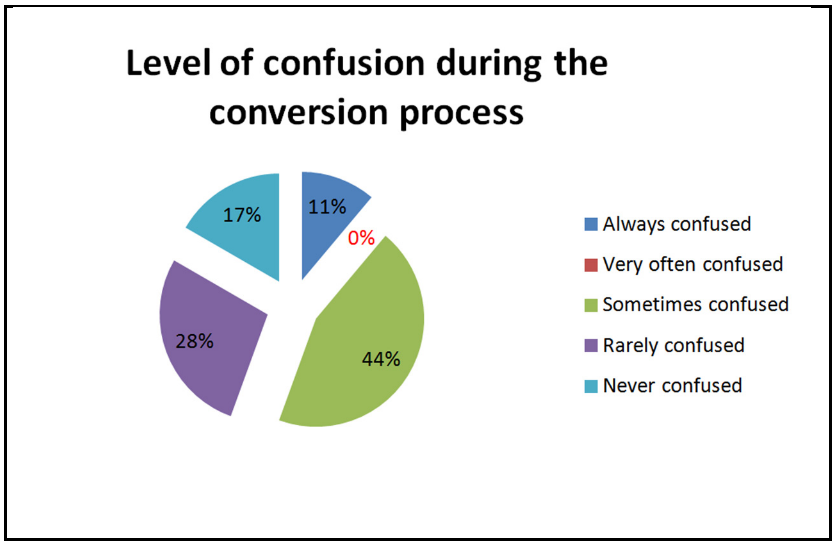


Figure 6.5 Level of Confusion During the Conversion Process

Figure 6.6 provides the opinion of the participants regarding whether the process would have been easier if a set of guidelines had been provided: 28% strongly agreed with their usefulness and 44% agreed with the relevance of this kind of tool to help in this conversion process. This was to be expected and was observed.

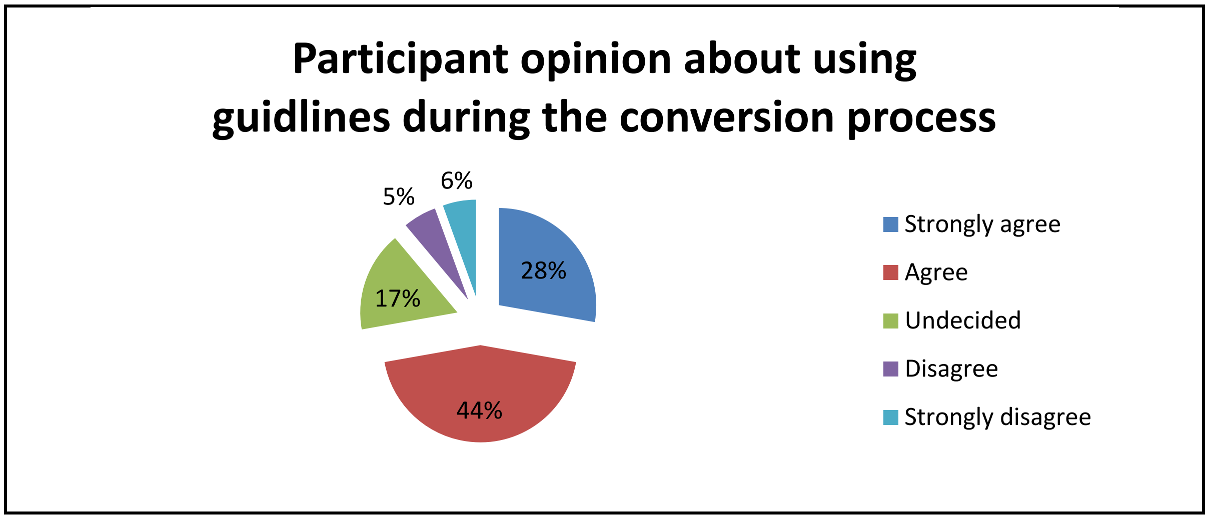


Figure 6.6 Participant Opinion about using Guidelines during the Conversion Process

Concerning the five key aspects of RDB that were studied in this conversion experiment: Tables, Constraints, Primary Keys (PK), Foreign Keys (FK) and other elements (fields, types of relationships, views, indexes, procedures and triggers), Table 6.4 summarizes, for each of these RDB aspects, the participants' reported percentage of coverage in intervals: 0%-24%, 25%-49%, 50%-74%, 75%-99% and 100%.

For example: 50% of the participants reported that their conversion solution proposal adequately covered the relational aspect "Tables" by 100%. In contrast, 22% considered that their proposed solution did not cover this aspect at all (0%). Moreover, Table 6.4 shows that 28% of the participants feel that their proposed solution covered 100% of the relational aspect "Constraints". On the other hand, 39% of the participants think that their solution did not cover this aspect at all (0%). Furthermore, Table 6.4 shows that 41% of the participants think that their proposed solution covers 100% of the relational aspect PK. However, 29% report that their solution did not cover this aspect at all (0%). Table 6.4 also reveals that 39% of the participants believe their solution covers 100% of the relational aspect FK. Conversely, 28% consider that their solution did not cover this aspect (0%). Other RDB elements aspects such as fields, store procedures or triggers were aggregated in the "others" category and 94% of the participants felt they had not fully covered these aspects during the conversion (see last row of Table 6.4).

Table 6.4 Level of Coverage in Different DB Aspects

Relational aspect covered	Percentage of coverage				
	0%	*25%*	*50%*	*75%*	*100%*
Table	22%	0%	6%	22%	50%
Constraint	39%	11%	17%	5%	28%
PK	29%	0%	18%	12%	41%
FK	28%	0%	11%	22%	39%
Others	94%	0%	0%	0%	6%

The next sub-section discusses the second experiment that followed this heuristic approach to the conversion.

6.3.2 Conversion Design Patterns

The approach selected to identify a preliminary list of conversion rules was to use a few samples of a relational schema and see how they could be converted into an HBase schema; the outcomes of this step will be called conversion designs patterns. This was prepared in a research lab by the author of the this thesis before initiating a second experiment with the other participants. These steps aimed at identifying the best corresponding HBase schema for each sample relational schema and next, extract/generalize rules to be used in experiment 2 that could help participants with the schema conversion. Through a number of iterations, three conversion design patterns were identified for each relationship as described in sections 6.3.2.1 and 6.3.2.2.

6.3.2.1 One-to-One Relationships

For the conversion of a one-to-one relationship, there are three possible ways to convert it into HBase (i.e. as described in 1a, 1b or 2):

1. **Create One HTable**: After merging the two RDB tables together into one HTable, the resulting HTable could have two resulting designs:

   a. Two-column family: One column family to store the first RDB table columns and a second column family to store the second RDB table columns.
   b. Single column family: This design pattern is recommended due to HBase limitations (Lars, 2013c).

2. **Create Two HTables**: The third possible design pattern is to create two HTables where each HTable contains one column family which contains all RDB columns. Finally, insert the row key of each HTable into both HTables.

The question that remains is how to choose one design over another. This is determined primarily by the context of the data access. If the original intent is to save space to optimize the caching of pages in an RDB, then this is no longer a consideration in HBase as the columns in the column family are dynamic and only added when needed. In this instance, merging two tables into one would be valid since saving space is no longer a concern in HBase given that null values are not saved.

An additional consideration would be whether there are two tables that contain information that becomes irrelevant when they are disconnected; for example, the relationship between a person and a passport as in Figure 6.7. This relationship is of no value when considered individually as a Passport needs to be attached to a Person to be valid and a Person needs to be attached to a Passport. For the purpose of this example, only a valid passport is considered.

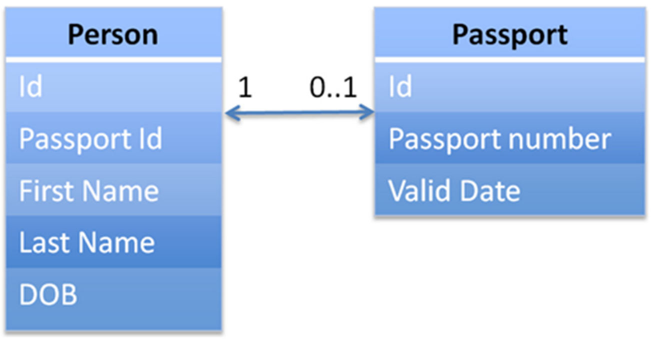
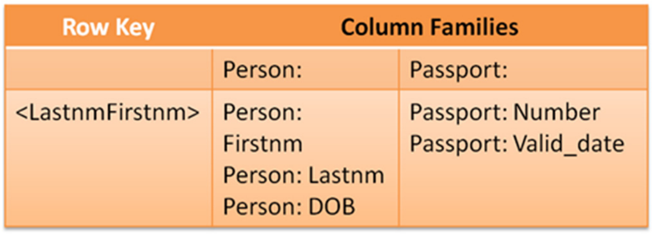


Figure 6.7 One-to-One Relationship in a Relational Database

As a result, a Person can only have one valid Passport. Two tables represent this relationship: one that holds Person information and the second that holds Passport information.

When converting this relationship into an HTable, the Person table would include another column family that would hold the passport information that belongs to a Person, thus reducing the two-table design in a relational database to one HTable as shown in Figure 6.8.



Row Key	Column Families	
	Person:	Passport:
<LastnmFirstnm>	Person: Firstnm Person: Lastnm Person: DOB	Passport: Number Passport: Valid_date

Figure 6.8 One HTable with Two Column Families

The reason for the inclusion of the Passport Number in the Person table is due to the context in which the information is usually accessed. To find the Passport number of an individual, a lookup of the Person is always required. A Passport number is meaningless without the Person information that accompanies it. In contrast, Person information is meaningful even without a Passport.

Another architectural decision to consider is whether to keep the design of two tables. This would happen if the secondary table contains information that is meaningful, in and of itself. Consider a relationship between an employee and a workstation and assume that an employee can only have one workstation. This relationship between the two tables should be represented in HBase as two tables as well. The reason for this is again context dependent since information about the workstation is relevant even without needing to know the employee information. Suppose that the address of the computer changes, an access to the workstation table would be required to update the address information. If the employee to whom the computer belongs is also required information, it will put an unreasonable burden on the lookup of the location of the workstation. In addition, a one table design would assume that every workstation is assigned to an employee although a company can have unused workstations that are not assigned to anyone and therefore would not be entered in the database.

6.3.2.2 One-to-Many Relationships

In the conversion of a one-to-many relationship, there are three possible ways to convert it into HBase:

1. Create One HTable with Two column families: This requires merging the information from the two related RDB tables. Each RDB table would be stored in a separate column family within the same HTable.
2. Create One HTable with a Single column family: This would entail that both RDB tables be merged into a single column family.
3. Create Two HTables: In this design, each RDB table would be added to a separate HTable with a single column family. The first HTable contains the source of the relationship and a second column family is added that contains the referenced Row Keys from the second HTable.

Once the conversion design patterns defined, the second experiment were ready to begin.

6.3.3 Experiment 2

6.3.3.1 Experiment 2 Goal

In experiment 2, the participants were divided into groups with each group assigned a distinct HBase conversion design pattern presented in section 6.3.2. The goal of assigning these conversion design patterns was to expose participants to different schema and thus avoid having the majority of participants using a similar design by following a trial and error method. Another objective was to build a body of knowledge by analysing the impact of all possible schemas, good and bad. Finally, having many participants helped ensure that every design pattern would be used.

6.3.3.2 Experiment 2 Use Case Description

A use case was developed for the experiment where participants were asked to convert a one-to-many relationship using the conversion patterns provided as presented in section 6.3.2. To accomplish this second experiment, a case study had to be developed simulating real life situations and characteristics as described below:

1. A large relational database currently experiencing performance issues that would benefit from a conversion to NoSQL technology.
2. An unknown industry domain to eliminate the experience factor in the experiment.
3. A population similar to experiment 1 to ensure valid and meaningful results.

A large RDB with such characteristics from the bioinformatics domain was available in our research lab. It originated from a hospital research laboratory where doctors and researchers were working on genomics and cancer research. This laboratory uses an RDB database with major performance issues. An ÉTS student, Anna Klos (Klos, 2012), had successfully solved this performance issue by converting the very large RDB tables to HBase.

a. **RDB for experiment 2:** The genomic database mainly stores information about human chromosomes and focuses primarily on the chromosome relationship with diseases that affect the brain and nervous system. The researchers investigated 13 tables in this RDB to locate the performance issues. In this experiment, two specific tables were found to create most of these issues as shown in Figure 6.9.

b. **Participants:** This time, the participants were also ÉTS students, with different education and experience levels enrolled in the undergraduate course “Introduction to Big Data”. The course objective was to introduce students to Big Data technologies including NoSQL databases. The class included 36 students that were divided into 9 teams.

c. **Experiment 2:** The experiment was introduced in the form of practical class assignments. The three assignments were as follows:

   i. Assignment 1: First replicate a portion of this genomic database using the same RDB technology (participants were asked to use PostgreSQL) as shown in Algorithm 6.1; then load the data provided (very large), execute the SQL query provided by the hospital lab that experienced performance problems and capture execution time (see Algorithm 6.1).

   ii. Assignment 2: Convert the PostgreSQL database schema created in assignment 1 to HBase, then load the same data into HBase and convert the same SQL query that was earlier provided to generate the same results. This resulting query should typically be a scan function in HBase, as described in section 6.3.3.3 (Design 1).

   iii. Assignment 3: Run the query created in assignment 2 on HBase; make sure it obtains the same results as previously with PostgreSQL and capture the execution time of the HBase implementation.

Each team was assigned a specific conversion design pattern as presented in section 6.3.2. The overall steps that were followed during this experiment are described in Figure 6.10.

Algorithm 6.1 SQL ngs_hg18_potion and ngs_position

```
SELECT hg18.chromosome_id,
       hg18.position
FROM   ngs_hg18_position hg18,
       ngs_position p
WHERE  hg18.id = p.hg18_id
       AND p.hg18_id > 100000
       AND p.hg18_id < 1000000
       AND p.hg19_id > 0;
```

The chosen relationship to be converted is located between table **ngs_hg18_position** and the table **ngs_position** as shown in Figure 6.9. These tables were chosen because of their large data size (1.1GB and 1.3GB respectively). When ignoring the multiple queries related to a specific schema, it is difficult to predict the most efficient schema design. However, the query in Algorithm 6.1 was provided as one that is frequently run against these particular tables. The query includes a join between the two tables as well as a selection of rows based on some simple criteria as shown in Algorithm 6.1.

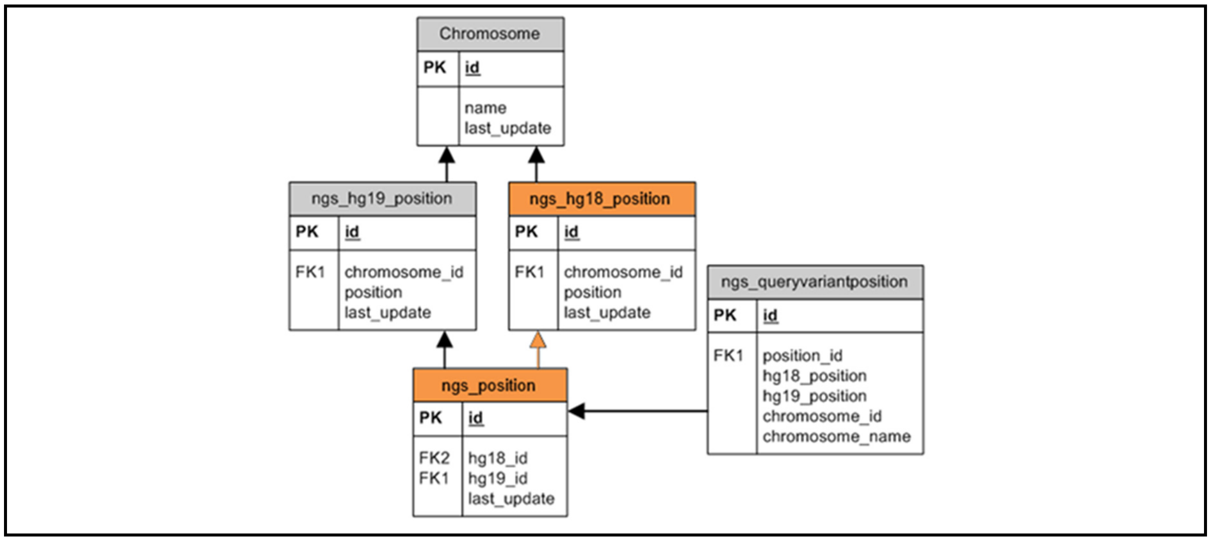


Figure 6.9 One-to-Many Relations between ngs_hg18_position and ngs_position

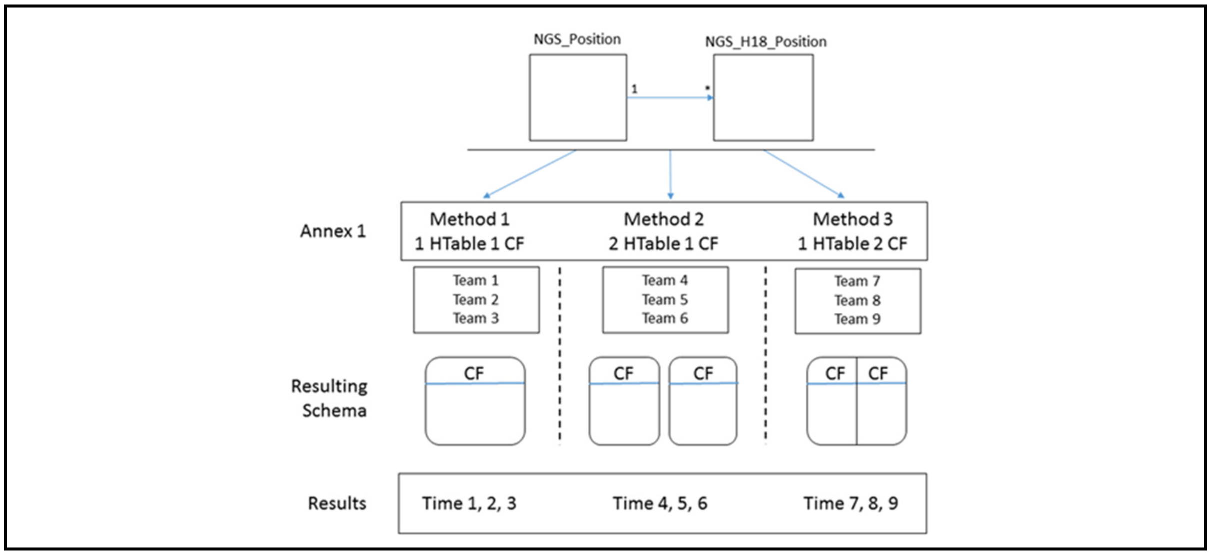


Figure 6.10 Overall Experimentation Steps

6.3.3.3 Experiment 2: Data

This subsection presents the data from the participants' attempts to convert the SQL query (see Algorithm 6.1) using the three conversion design patterns described in section 6.3.2.

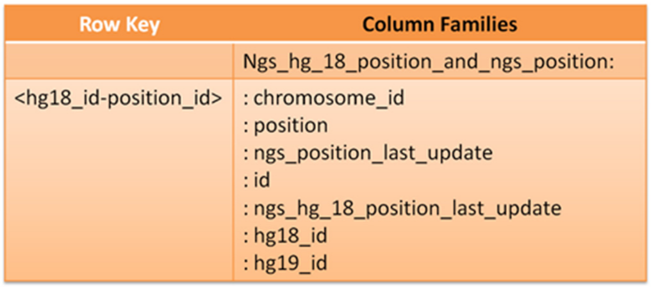
Design 1 - One HTable and One Column Family

Using the first schema of one HTable and one column family as shown in Figure 6.11, the participants converted the data and created the query below, which returned the same data as the originating SQL query in Algorithm 6.1.

Algorithm 6.2 Scan Query of ngs_hg18_and_position_t2_m1

```
Scan 'ngs_hg18_and_position_t2_m1',
(Kasunic, 2005 ENDROW=>'010000000', FILTER
=>"SingleColumnValueFilter('hg18_and_position_family','ngs_hg19_position_id',>,
'binary:0')")
```

The HBase scan that was executed yielded a query that returned 900,001 rows in 878 seconds.

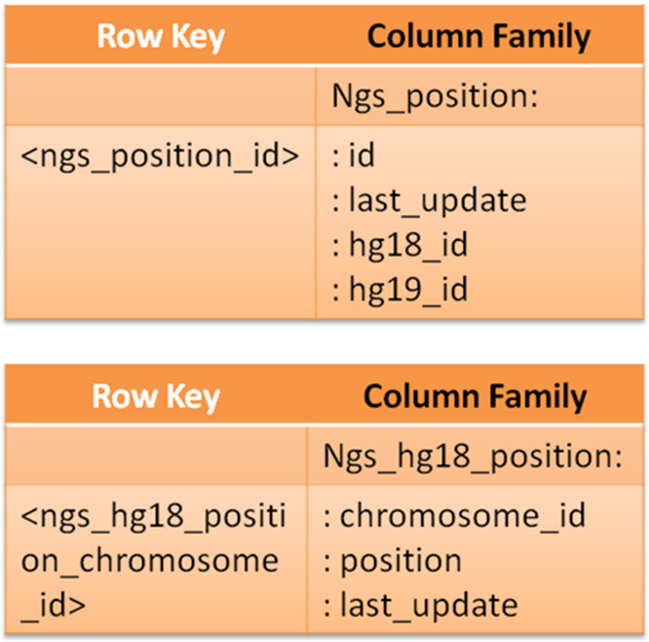


Row Key	Column Families
	Ngs_hg_18_position_and_ngs_position:
<hg18_id-position_id>	: chromosome_id : position : ngs_position_last_update : id : ngs_hg_18_position_last_update : hg18_id : hg19_id

Figure 6.11 One HTable with One Column Family Use Case

Design 2 - Two HTables with One Column Family per Table

In the two HTable design shown in Figure 6.12, data must be fetched from both HTables and then used in a comparison to filter out the wanted data to complete the SQL query requested in Algorithm 6.2. A join clause between the two HTables is necessary to accomplish this filter. Since HBase does not use relations in its design (Lars, 2013d), the concept of a query with a join is not possible. Therefore, none of the participants was able to find a matching query.



Row Key	Column Family
	Ngs_position:
<ngs_position_id>	: id : last_update : hg18_id : hg19_id

Row Key	Column Family
	Ngs_hg18_position:
<ngs_hg18_position_chromosome_id>	: chromosome_id : position : last_update

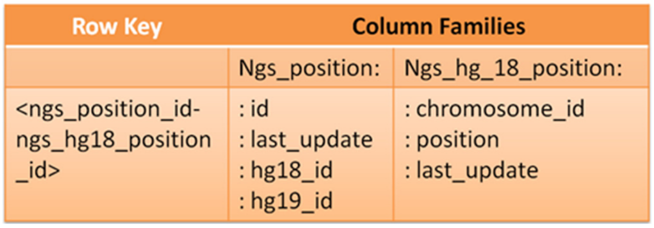
Figure 6.12 Two HTables with One Column Family Use Case

Design 3 - One HTable with Two Column Families

The participants in this group created an HTable as shown in Figure 6.13 and attempted to replicate the SQL query. The results obtained, using the scan below, timed out after a long execution period. We should state here that the infrastructure used in the experiment has not yet been described. The distribution and parallel processing of HBase was not taken into account since the lab infrastructure for the experiment was a single virtual machine running on the same server as experiment 1 (i.e. attempting to replicate the previous PostgresSQL environment). This query would not time out if it was executed in a multi-server, distributed environment. Nevertheless, this approach performed poorly in comparison with other schema design patterns, as we will see.

Algorithme 6.3 Scan Query t2_2:position

```
scan't2',{COLUMNS=>['t2_2:chromosome_id','t2_2:position','t2_1:
hg18_id','t2_1:hg19_id'],
FITER=>FilterList.new([SingleColumnValueFilter.new(Bytes.toBytes('t2_1'),Bytes.t
oBytes('hg18_id'),
CompareFilter::CompareOp.valueOf('GREATER'),Bytes.toBytes('100000')),
SingleColumnValueFilter.new(Bytes.toBytes('t2_1'),Bytes.toBytes('hg18_id'),
CompareFilter::CompareOp.valueOf('LESS'),Bytes.toBytes('1000000')),
SingleColumnValueFilter.new(Bytes.toBytes('t2_1'),
Bytes.toBytes('hg19_id'),CompareFilter:
:CompareOp.valueOf('GREATER'),Bytes.toBytes('0'))]) }
```



Row Key	Column Families	
	Ngs_position:	Ngs_hg_18_position:
<ngs_position_id-ngs_hg18_position_id>	: id : last_update : hg18_id : hg19_id	: chromosome_id : position : last_update

Figure 6.13 One HTable with Two Column Families Use Case (Ouanouki et al., 2017)

6.3.3.4 Experiment 2: Analysis

Design 1, which is "one HTable and one column family" yielded the requested data (of about 900k rows) in 878 seconds. A disadvantage of using a single column family per HTable is that if too many columns are used, the size needed to store one row will increase. Due to the HBase distributed architecture, each column family is broken up into files of a specific size that split when the column family reaches the size limit (Lars, 2013c). This means that the larger the data stored in a column family of a row, the fewer rows there will be per column family file. This in turn makes the scan time longer as more files could potentially be needed to complete the query.

Design 2, the reason why no group was able to fetch data when using two HTables is that joins are not inherently possible as in a traditional RDB approach. Since HBase is not a relational database, any relation between the information had to be made manually at the application level (Lars, 2013e). This translates into the need to write two queries. The first query would get the data from the ngs_position table and would use the returned result set as a filter with the second query on the ngs_hg18_position table. Therefore, this would call for extra logic to be added at the application level and would require more time to execute. Given that this SQL query is run very often, it would add extra time for the fetching of data.

For the third design, one HTable with two column families, the issues with this pattern are related to the limitations in the design of HBase itself. In HBase, a column family is stored in a single file split across HDFS (Lars, 2013c), which means that when accessing a row there is the potential of accessing two physical files. Considering the distributed nature of HBase, this implies that each column family file is not necessarily stored on the same server. Consequently, access time can be greatly increased if a scan needs to access both column families of the same table.

The result set that was obtained clearly points to the proper solution for this type of access pattern. Design 1 is the only situation that can provide meaningful results. Designs 2 and 3 were unable to yield any data, which demonstrates that the HBase schema is closely linked to the access patterns of the database. Therefore, this shows that when converting an RDB- based application to HBase, an access pattern analysis is required to avoid proposing an HBase design that would potentially include weaknesses that would need further refactoring and costs.

6.3.4 Rules Extraction

Using the knowledge gained from these two experiments and the three conversion design patterns experimented, we extracted considerations that should be taken into account when planning the conversion of an existing information system that uses RDB technology to the HBase technology.

6.3.4.1 Rule on Data Proximity

Firstly, every conversion will differ and depends heavily on the data access patterns currently used. Since HBase does not have any steadfast rules, it needs to have an understanding of how each data field relates to another. One of the most important rules that requires serious consideration is how the data will be accessed by the application. For that reason, the distance between frequently accessed data, in one scan, will definitely affect the query performance. Therefore, the data needs to be stored in the same column family, as much as possible, so as not to require multiple scans to obtain the desired information. This goes as far as maintaining a minimum amount of information pertaining to a secondary table and removing the need to go into the secondary table to get the information that is required. This rule can be inferred from the results of the two HTable design pattern of our use case. Since HBase has no implicit way of creating relationships when performing a scan (Biswapesh Chattopadhyay, 2011; Marcos, 2005), the application itself will need to complete this action. This introduces extra complexity/effort that could be avoided by using a more optimal schema designed specifically for the data access patterns.

Rule 1.1: The more the information is accessed at one time, the closer the data needs to be stored. More specifically, when the same data is often accessed together, it should go into a one HTable design within the same column family.

6.3.4.2 Rules on Column Families

The concept of RDB columns does not implicitly translate to a column family in HBase. As presented above, having many column families in one HTable can be detrimental to the schema translation due to the physical design limitations (Lars, 2013c) of HBase. Since each column family is stored in its own separate file and then stored in a distributed manner, the larger the distribution, the less likely that the physical column family files will be stored in the same region server (Lars, 2013c). Thus, the second rule inferred from these experiments is not to store information in column families that are of vastly different sizes (Lars, 2013c). As a column family is broken up into different files and distributed in the database, the larger each row is, the smaller the number of rows that will be stored within the same file and therein creates the need for more files. This in turn will require the scan to access even more files to fetch the information wanted. However, the larger the network, the less impact this will have as the information will be read on each node and aggregated together by the master (Lars, 2013c).

One major concern when designing an HTable is that the information will be distributed as evenly as possible. If there is a disparity in the number of rows needed to store information between two column families, this could lead to longer scans of the smaller column family as there are fewer records in the column and therefore, more empty space between rows (Chongxin, 2010).

Rule 2.1: The number of column families defined in each row should be no more than 2 or 3; otherwise, the performance will degrade due to the physical design of HBase. Specifically, because flushing and compression are done on a per region basis, if one column family is carrying the bulk of the data bringing on flushes, the adjacent families will also be flushed though the amount of data they carry is small. With many column families, the flushing and compression interaction can make for a number of needless I/O loading (Lars, 2013c).

Rule 2.2: The number of rows in each column family needs to be roughly the same; then the columns should be split and references placed in each column family to allow data to be linked.

6.3.4.3 Rules on Data Quantity

HBase is designed to handle large amounts of data. HBase was modelled on Google's own BigTable (Fay et al., 2006) for that purpose. It has incorporated the ability to handle large data requests and by doing so, also incorporated some physical limitations on how the data is stored. These limitations need to be considered when converting an RDB schema as the expected performance gains may be mitigated given the limitations. The limitations impose two design considerations. The first is with respect to the size of the column families in a single row. Each column family needs to be of relatively similar size as they are stored in separate physical files of a maximum defined size. Once the maximum file size is reached, it is split and then dispersed. This means that when data stored in one column family is significantly larger than the other, it will create a situation where fewer rows are stored per physical file and again cause longer scans (Lars, 2013a). The second consideration is that column families should generally have the same access pattern. Otherwise, when fetching data, the scan could take much longer if the access pattern to each column family is vastly different.

Rule 3.1: Data size between columns families, in one row, need to be of similar size, otherwise, the column family needs to be split into two separate HTables, each containing a column family.

Rule 3.2: Column families in the same HTable should generally have the same access patterns.

6.3.4.4 Rules on Access Patterns

The context in which the data is accessed in the application defines the access patterns and the accessibility requirements. These will be the foundation upon which the HBase database will be created. As shown in section 6.3.3.3 (Design 1), this conversion was able to fulfil the required access pattern defined by the use case. Understanding the data access patterns and the data flow models of the application will assist in the design, especially since there are no steadfast rules when converting to HBase. The resulting schema will depend heavily on the context in which the data is accessed.

Subsequently, an analysis of the most used access patterns and of the heavy queries regularly performed is also needed before attempting a conversion.

Rule 4.1: Access patterns will define the conversion and need to be defined and understood; an analysis is needed before a conversion.

Rule 4.2: Analysis of heavy queries is required to obtain a proper design.

6.4 Conclusions and Future Research

This chapter has identified and explored a set of rules to assist in the conversion of a relational database to one type of NoSQL database—the column-oriented database named HBase. It described a first experiment designed to evaluate if there was a need for a set of conversion rules. In this experiment, it was demonstrated that not all RDB practitioners could easily carry out such a conversion. Next, a second experiment showed how the replicated query reacted depending on the design pattern used. This second experiment provided an opportunity to uncover a first set of schema translation rules for this particular conversion. Future research could explore a conversion experiment using these conversion rules with the goal to expand this initial body of knowledge and further validate the conversion rules, patterns and guidelines.

CONCLUSION

This thesis has presented a clarification to the cloud computing definition that is better suited for this research purpose: the conversion of relational databases to NoSQL database technology. Two experiments where conducted to extract conversion rules specific to the research topic:

1. The first experiment consisted of asking participants (engineers from *École de technologie supérieure*) to convert a particular schema, without the use of a guide or conversion rules. Several questionnaires were completed by the participants during the experiment. The main goal was to evaluate the need for conversion rules to help software engineers with the conversion of relational schemas to non-relational schemas;
2. The second experiment consisted of having multiple groups of participants, with each group tasked to convert a single RDB relation to a possible schema in HBase. The goal was to test the results of a single RDB relationship to all schematic conversion possibilities in HBase, a particular non-relational database, and then extract the conversion rules. This experiment showed why a particular schema is better than another and allowed for the identification of conversion rules based on facts (e.g., performance measurement of a query on each of the resulting schemas).

Main contribution and outcomes

1. A list of conversion rules that help software engineers in their attempt at a conversion was established, more specifically, the database schema conversion. A first experiment confirmed that there is a need for conversion guidelines (or conversion rules) to help software engineers, especially because of the nature of the target schema, which does not support relationships between tables like a relational database. The second experiment helped in identifying conversion rules using a trial and error approach. The experiment engaged multiple groups in converting different possible schemas to unveil which resulting converted schema was more suitable and why. The outcome of this experiment

and the resulting conversion rules were published in the following journals and conferences:

a. Ouanouki, R., April, A., Abran, A., Gomez, A. and Desharnais, J. M. (2017). Toward building RDB to HBase conversion rules. Journal of Big Data, 4(1). doi:10.1186/s40537-017-0071-x
b. Ouanouki R, G. A. (2014). Building an Experiment Baseline in the Migration Process from SQL Databases to Column Oriented No-SQL Databases. Journal of Information Technology & Software Engineering, 04(02). doi:10.4172/2165-7866.1000137
c. Gomez, A., Ouanouki, R., Ravanello, A., April, A. and Abran, A. (2015). Experimental Validation as Support in the Migration from SQL Databases to NoSQL Databases presented in CLOUD COMPUTING 2015: The Sixth International Conference on Cloud Computing, GRIDs, and Virtualization.
   doi: 978-1-61208-388-9

2. A second, smaller contribution was to clarify the cloud computing definition in the context of database conversion by considering the mandatory and optional characteristics of cloud computing in this specific research context. Two publications resulted from this research activity; one as a book chapter in an encyclopedia and one in a journal paper:
   a. Ouanouki, R., Morales, A. and April, A. (2014). Rationalizing the Cloud Computing Concept: An Analogy with the Car. Journal of Cloud Computing, 1–8. doi:10.5171/2014.901075
   b. Rafik, O., Abraham Gomez, M. and Alain, A. (2015). Should the Cloud Computing Definition Include a Big Data Perspective? Dans D. B. A. Mehdi Khosrow-Pour (Éd.), Encyclopedia of Information Science and Technology, Third Edition (pp. 1088-1095). Hershey, PA, USA: IGI Global. doi: 10.4018/978-1-4666-5888-2.ch104

Limitations

Some limitations have been identified in this thesis. An overview of these limitations is presented as follows:

- More experimentation would help to create an improved list of conversion rules. Additional experiments would be required to further validate the effectiveness of these conversion rules.
- This list of conversion rules is limited to a specific type of NoSQL database, Wide Column Store/Column Families and for some conversion rules, limited to Hadoop HBase. Additional research could consider the applicability of these rules to other NoSQL database technologies;
- The conversion rules were developed and extracted from single database relation such as one to one and one to many relations. Research could be initiated to understand the effect of larger database schemas on these conversion rules;
- The conversion rules could also need to be adjusted if this NoSQL technology evolves or changes.

In conclusion, when this research started, we asked the NoSQL developer community if there was a best way to convert a relational database to NoSQL? The response was often that it depends on the way the data is accessed in the current relational database. This answer motivated this research work for the discovery of conversion rules based on data access. This thesis has proposed a set of conversion rules that can be used to help software engineers in their conversion effort to convert an existing RDB to HBase.

APPENDIX I

THE NIST DEFINTION OF CLOUD COMPUTING

1.1 Authority

The National Institute of Standards and Technology (NIST) developed this document in furtherance of its statutory responsibilities under the Federal Information Security Management Act (FISMA) of 2002, Public Law 107-347.

NIST is responsible for developing standards and guidelines, including minimum requirements, for providing adequate information security for all agency operations and assets; but such standards and guidelines shall not apply to national security systems. This guideline is consistent with the requirements of the Office of Management and Budget (OMB) Circular A-130, Section 8b(3), "Securing Agency Information Systems," as analyzed in A-130, Appendix IV: Analysis of Key Sections. Supplemental information is provided in A-130, Appendix III.

This guideline has been prepared for use by Federal agencies. It may be used by nongovernmental organizations on a voluntary basis and is not subject to copyright, though attribution is desired.

Nothing in this document should be taken to contradict standards and guidelines made mandatory and binding on Federal agencies by the Secretary of Commerce under statutory authority, nor should these guidelines be interpreted as altering or superseding the existing authorities of the Secretary of Commerce, Director of the OMB, or any other Federal official.

1.2 Purpose and Scope

The purpose of this publication is to provide the NIST definition of cloud computing. NIST intends this informal definition to enhance and inform the public debate on cloud computing. Cloud computing is still an evolving paradigm. Its definition, use cases, underlying technologies, issues, risks, and benefits will be refined and better understood with a spirited debate by the public and private sectors. This definition, its attributes, characteristics, and underlying rationale will evolve over time.

1.3 Audience

The intended audience is people adopting the cloud computing model or providing cloud services.

The NIST Definition of Cloud Computing

Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model promotes availability and is composed of five essential characteristics, three service models, and four deployment models.

Essential Characteristics:

- On-demand self-service. A consumer can unilaterally provision computing capabilities, such as server time and network storage, as needed automatically without requiring human interaction with each service's provider.
- Broad network access. Capabilities are available over the network and accessed through standard mechanisms that promote use by heterogeneous thin or thick client platforms (e.g., mobile phones, laptops, and PDAs).
- Resource pooling. The provider's computing resources are pooled to serve multiple consumers using a multi-tenant model, with different physical and virtual resources dynamically assigned and reassigned according to consumer demand. There is a sense of location independence in that the customer generally has no control or knowledge over the exact location of the provided resources but may be able to specify location at a higher level of abstraction (e.g., country, state, or datacenter). Examples of resources include storage, processing, memory, network bandwidth, and virtual machines.

- Rapid elasticity. Capabilities can be rapidly and elastically provisioned, in some cases automatically, to quickly scale out, and rapidly released to quickly scale in. To the consumer, the capabilities available for provisioning often appear to be unlimited and can be purchased in any quantity at any time.
- Measured Service. Cloud systems automatically control and optimize resource use by leveraging a metering capability1 at some level of abstraction appropriate to the type of service (e.g., storage, processing, bandwidth, and active user accounts). Resource usage can be monitored, controlled, and reported, providing transparency for both the provider and consumer of the utilized service.

Service Models:

- Cloud Software as a Service (SaaS). The capability provided to the consumer is to use the provider's applications running on a cloud infrastructure. The applications are accessible from various client devices through a thin client interface such as a Web browser (e.g., web-based email). The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, storage, or even individual application capabilities, with the possible exception of limited user-specific application configuration settings.
- Cloud Platform as a Service (PaaS). The capability provided to the consumer is to deploy onto the cloud infrastructure consumer-created or acquired applications created using programming languages and tools supported by the provider. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, or storage, but has control over the deployed applications and possibly application hosting environment configurations.

- Cloud Infrastructure as a Service (IaaS). The capability provided to the consumer is to provision processing, storage, networks, and other fundamental computing resources where the consumer is able to deploy and run arbitrary software, which can include operating systems and applications. The consumer does not manage or control the underlying cloud infrastructure but has control over operating systems, storage, deployed applications, and possibly limited control of select networking components (e.g., host firewalls).

Deployment Models:

- Private cloud. The cloud infrastructure is operated solely for an organization. It may be managed by the organization or a third party and may exist on premise or off premise.
- Community cloud. The cloud infrastructure is shared by several organizations and supports a specific community that has shared concerns (e.g., mission, security requirements, policy, and compliance considerations). It may be managed by the organizations or a third party and may exist on premise or off premise.
- Public cloud. The cloud infrastructure is made available to the general public or a large industry group and is owned by an organization selling cloud services.
- Hybrid cloud. The cloud infrastructure is a composition of two or more clouds (private, community, or public) that remain unique entities but are bound together by standardized or proprietary technology that enables data and application portability (e.g., cloud bursting for load balancing between clouds).

APPENDIX II

ISO/IEC JTC 1 N9687 – A STANDARDIZATION INITIATIVE FOR CLOUD COMPUTING

1. Introduction to Cloud computing

1.1. General concept and characteristics of Cloud computing

The concept of Cloud Computing is drawing a great attention from the Information and Communication Technology (ICT) community, thanks to the appearance of a set of services with common characteristics, provided by important industry players. However, some of the existing technologies that the Cloud Computing concept draws on (such as virtualization, utility computing or distributed computing) are not new.

The term Cloud is used as a metaphor for the Internet, based on how the Internet is depicted in computer network diagrams and is an abstraction for the complex infrastructure it conceals.

The followings are the key characteristics of Cloud computing.

- **Agility** improves with users who are able to rapidly and inexpensively re-provision technological infrastructure resources. The cost of overall computing is unchanged, however, and the providers will merely absorb upfront costs and spread costs over a longer period.
- **Cost** is claimed to be greatly reduced and capital expenditure is converted to operational expenditure. This ostensibly lowers barriers to entry, as infrastructure is typically provided by a third-party and does not need to be purchased for one-time or infrequent intensive computing tasks. Pricing on a utility computing basis is fine-grained with usage-based options and fewer IT skills are required for implementation (in-house). Some would argue that given the low cost of computing resources, the IT burden merely shifts the cost from in-house to outsourced providers. Furthermore, any

cost reduction benefit must be weighed against a corresponding loss of control, access and security risks.

- **Device and location independence** enable users to access systems using a Web browser regardless of their location or what device they are using (e.g., PC, mobile). As infrastructure is off-site (typically provided by a third-party) and accessed via the Internet, users can connect from anywhere.
- **Multi-tenancy** enables sharing of resources and costs across a large pool of users thus allowing for:
    - Centralization of infrastructure in locations with lower costs (such as real estate, electricity, etc.)
    - Increase of Peak-load capacity (users need not engineer for highest possible load-levels)
    - Utilization and efficiency improvements for systems that are often only 10–20% utilized.
- **Reliability** improves through the use of multiple redundant sites, which makes Cloud computing suitable for business continuity and disaster recovery. Nonetheless, many major Cloud computing services have suffered outages, and IT and business managers can at times do little when they are affected.
- **Scalability** via dynamic ("on-demand") provisioning of resources on a fine-grained, self-service basis near real-time, does not impose users to engineer for peak loads. Performance is monitored and consistent and loosely-coupled architectures are constructed using Web services as the system interface.
- **Security** typically improves due to centralization of data, increased security-focused resources, etc., but concerns can persist about loss of control over certain sensitive data and verification of users' identity. Security is often as good as or better than traditional systems, in part because providers are able to devote resources to solving security issues that many customers cannot afford. Providers typically log accesses, but accessing the audit logs themselves can be difficult or impossible. Ownership, control and access to data controlled by "Cloud" providers may be made more difficult, just as it is sometimes difficult to gain access to "live" support with current utilities. Under the

Cloud paradigm, management of sensitive data and other security-related functions (e.g., user's enrolment and verification of users' identity) could be placed in the hands of Cloud providers and third parties.

- **Sustainability** comes about through improved resource utilization, more efficient systems, and carbon neutrality. Nonetheless, computers and associated infrastructure are major consumers of energy. A given (server-based) computing task will use X amount of energy whether it is on, or off-site.

1.2. Definition of Cloud computing

There are many definitions of Cloud Computing, but they all seem to focus on just certain aspects of the technology. So, more than 20 definitions of Cloud computing have been developed allowing for the extraction of a consensus definition as well as a minimum definition containing the essential characteristics. The following are the remarkable definitions and descriptions of Cloud Computing.

- A pool of abstracted, highly scalable, and managed compute infrastructure capable of hosting end-customer applications and billed by consumption.
- A style of computing in which dynamically scalable and often virtualized resources are provided as a service over the Internet. Users need not have knowledge of, expertise in, or control over the technology infrastructure in the "Cloud" that supports them.
- Cloud computing is an approach to shared infrastructure in which large pools of systems are linked together to provide IT services (Press release on “Blue Cloud”, IBM).
- A paradigm in which information is permanently stored in servers on the Internet and temporarily cached on clients that include desktops, entertainment centers, table computers, notebooks, wall computers, handhelds, etc. (ORGs for Scalable, Robust, Privacy-Friendly Client Cloud computing, IEEE Internet Computing).

In conclusion, for the business perspective, Cloud computing is providing IT infrastructure and environment to develop/host/run services and applications, on demand, with pay-as-you-go pricing, as a service. And, from the users point of view, Cloud computing is providing resource and services to store data and run application, in any devices, anytime, anywhere, as a service. Nowadays, the usage of Cloud computing is extended for many domain-specific areas including network services, mobile services, media service etc. So, it is expected that there will be lots of variations for future Cloud service and related standardisation issues as well.

APPENDIX III

EXPERIMENT 1 – SURVEY

RDB to No-SQL Survey

Participant Code: ___________________________

Experience Classification: Please fill with an "X" in the answer column.

1. You are:

Question	**Answer**
Graduate with PhD	
Graduate with Master	
Graduate	
Undergraduate Student	

2. You work in:

Question	**Answer**
Industry	
Academic	
Research Center	

3. How many experience years do you have working in relational database environment or programming?

Question	**Answer**
No Experience	
Low Experience (<1 Year)	
Middle Experience (2-5 Years)	
Advanced Experience (>5 Years)	

4. How many experience years do you have working in or related to any No-SQL database?

Question	**Answer**
No Experience	
Low Experience (<1 Year)	

Middle Experience (2-5 Years)	
Advanced Experience (>5 Years)	

5. Migration process: Please fill with an "X" in the answer column.

Question	**Answer**
Did you try migrate each table in the source and obtaining one corresponding table in the target?	
Did you try to mix some tables of the source and obtain one corresponding table in the target?	
Did you try to migrate in some way the relationships from the source to target?	
Another option? Which one? (Please fill out in print, rather than handwritten)	

6. Please rate how easy is to carry out the entire migration process, without a well-established method? A value of 1 indicates that the process was easy to achieve without effort, a value of 3 indicates that it was required a maximum effort to achieve it and a value of 5 means that no matter how comprehensive the effort, it was no possible to achieve it.

1	**2**	**3**	**4**	**5**

7. Did you feel confused (e.g. no idea where to start or what the next step was) on how to carry out the entire migration process?

1	**2**	**3**	**4**	**5**
Always confused	Very often confused	Sometimes confused	Rarely confused	Never confused

8. In your opinion, your solution (No-SQL Schema in the green sheet) is covering all aspects developed by the relational schema? Please fill with an "X" the percentage that you think was covered for your solution schema.

Percentage of coverage	0%	25%	50%	75%	100%
Relational aspect covered					
Table					

Constraint					
PK					
FK					
Other					
Other					

9. In your opinion, if you had received guidelines about how to make the conversion/migration process, this had improved your task?

1	**2**	**3**	**4**	**5**
Strongly agree	Agree	Undecided	Disagree	Strongly disagree

APPENDIX IV

EXPERIMENT 2 – ASSIGNMENT DESCRIPTION

- **Introduction**
  L'objectif de ce second laboratoire est de vous familiariser avec les bases de données HBase, ainsi que le processus de migration de données d'une base de données relationnelle vers une base de données HBASE. HBase est une base de données non-relationnelle, en logiciel libre, utilisant Java. Elle a été conçue à partir des spécifications de la base de données BigTable de Google. HBase fait partie du projet Apache **Hadoop**. Ce laboratoire introduit quelques concepts de HBase tels que : les tables, les « get » et les « scan ». En particulier, ce travail pratique se concentrera sur l'utilisation d'HBase comme solution potentielle aux problèmes observés lors de l'utilisation de la base de données relationnelle (PostgreSQL) de l'application de Bio-Bigdata. L'ensemble des travaux pratiques réalisés ici s'inspire des travaux effectués dans le laboratoire précédent. Ainsi, la base de données relationnelle sera utile pour élaborer un nouveau schéma pour HBase.

- **Tables et relations**
  Pour ce deuxième TP, nous allons seulement tenir compte de relations présentes dans les schémas suivants.

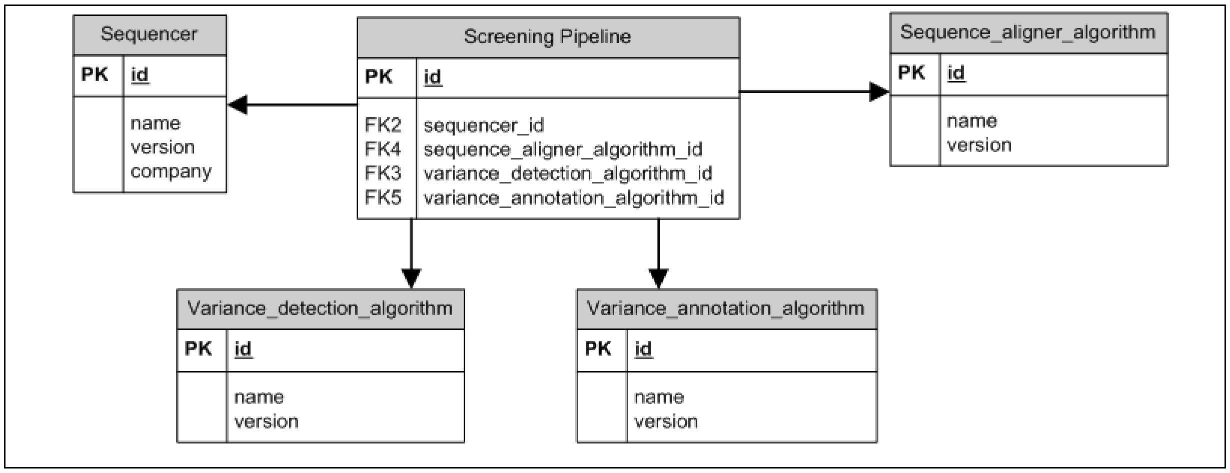

Sequencer
PK id
name
version
company
Screening Pipeline
PK id
FK2 sequencer_id
FK4 sequence_aligner_algorithm_id
FK3 variance_detection_algorithm_id
FK5 variance_annotation_algorithm_id
Sequence_aligner_algorithm
PK id
name
version
Variance_detection_algorithm
PK id
name
version
Variance_annotation_algorithm
PK id
name
version

Figure 1: Schéma de la base de données – Pipeline et Algorithme

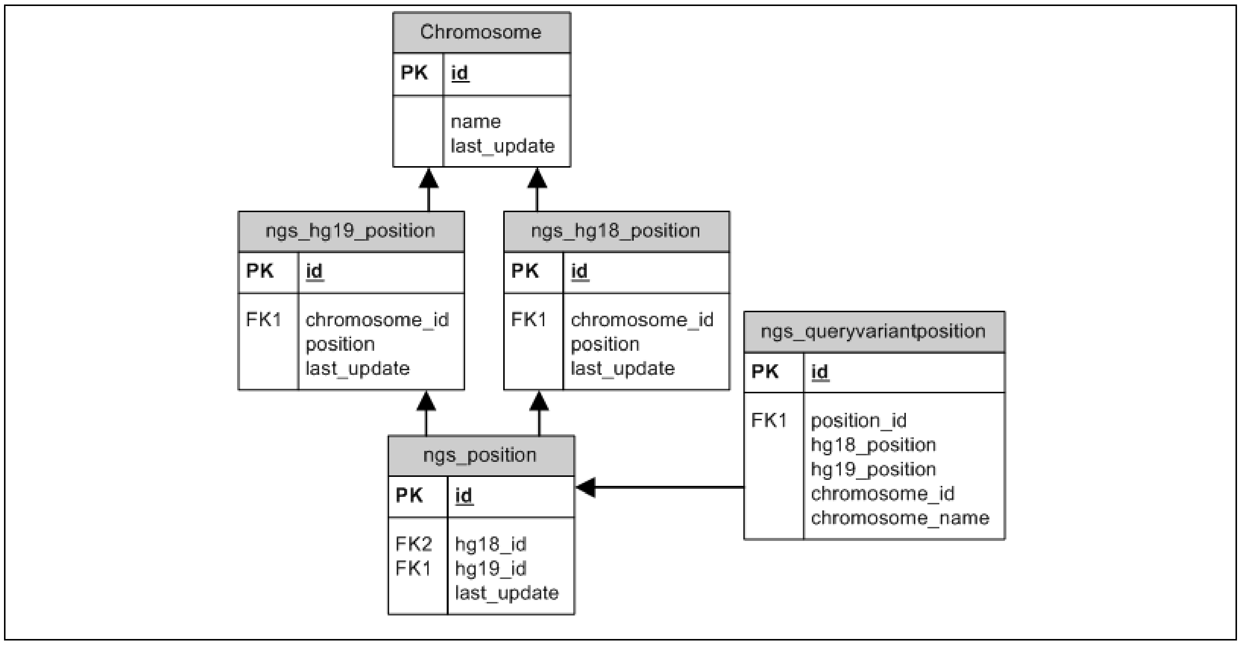


Figure 2: Schéma de la base de données – Chromosome, HG 18, HG19 et Position

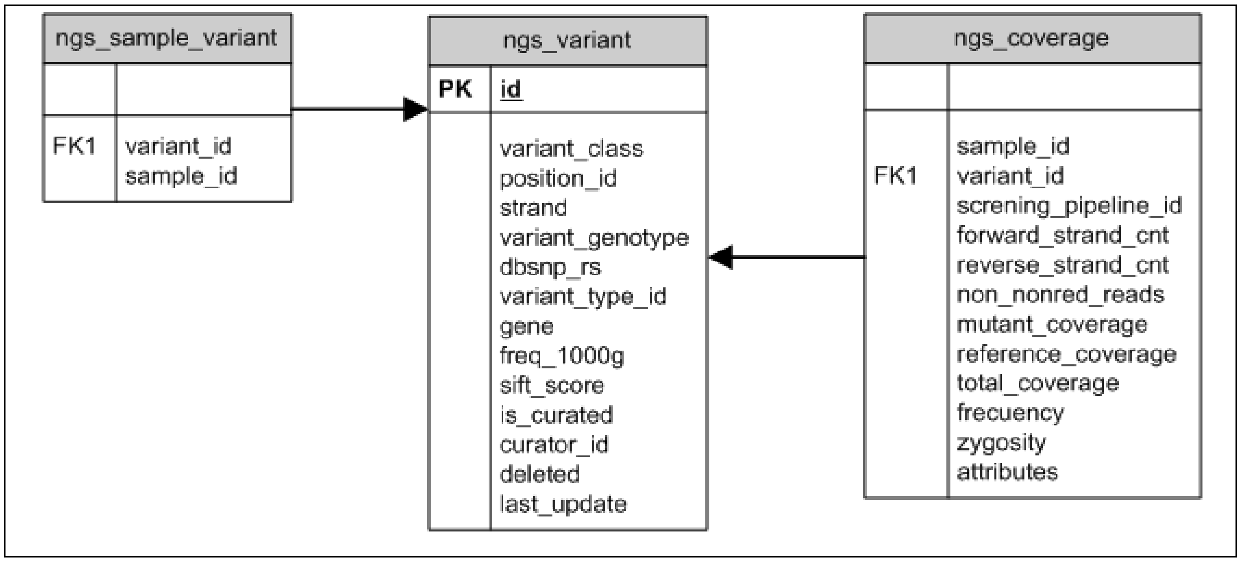


Figure 3: Schéma de la base de données – Variation et Coverage

- **Travail à réaliser :**

  - **Tâche 1: Expérimenter avec les types de migration d'un schéma relationnel vers HBase**
    La première tâche est pour vous familiariser avec les différentes techniques de migration SQL à NoSQL. Pour cette tâche vous devez sélectionner 2 tables de la Figure 1 « *Pipeline et Algorithme* » et effectuer leur migration de PostgreSQL vers HBase en utilisant les 3 techniques de migration décrites à l'annexe A de ce document. L'annexe A contient également des informations techniques concernant l'importation.

    Réalisation attendue :
    D'abord, sélectionner 2 tables du schéma 1 qui partage une relation « 1 à plusieurs » dans le schéma relationnel. Par exemple, les tables « Sequencer » et « Screening Pipeline ». **Indiquer les tables sélectionnées.**

    Ensuite, pour chacune des 3 techniques :
    1. Pour les méthodes 1 et 3, créer 1 vue dans PostgreSQL qui combine les données de ces tables. Pour la méthode 2, deux vues seront nécessaires (1 vue pour chaque table). **Indiquez le code pour les vues**. La définition du RowKey de HBase est laissé à votre discrétion.
    2. Importer les données des vues dans HBASE à l'aide de Sqoop. **Indiquez le code pour l'import.**
    3. Faire un scan d'une rangée des valeurs dans HBASE. **Indiquez le code pour scan ainsi que le résultat** (capture d'écran du résultat)**.**

  - **Tâche 2: Migration de 2 larges tables avec une technique particulière**

    Pour cette tâche, chaque équipe utilisera une méthode de migration différente pour importer dans HBase les tables « **NGS_position** » et « **NGS_hg18_position** », ainsi que leur relation. Voici la liste des équipes qui sont assignées à chacune des méthodes :

    - **Méthode 1.** Le résultat attendu est : une table HBase avec une famille de colonnes.

Équipe 1	**Équipe 2**	**Équipe 3**
Julien Béliveau	Ronald Lessage	Cédric St-Onge
Danny Boyer	Vincent Larose	Francis Gagnon-Tessier
David Lafrance	Alex Lévesque	Reda Benhsaïn
Riad Chebli	Thanh Lam Hoang	Guy Hounsa

    - **Méthode 2.** Le résultat attendu est : deux tables HBase.

Équipe 4	**Équipe 5**	**Équipe 6**
Sébastien Bonami	Francis Olivier Laporte	Stéphane Nganyo Noulala
Olivier Rivard	Yan Vigeant Bruyère	Jonathan Hallée
Francis Poirier	Simon Larose	Gabriel Forget

Bunpa-Henri Tan	Oualid Ben Yahia	Karen Fitzpatrick

- **Méthode 3**. Le résultat attendu est : une table HBase avec deux familles de colonnes.

Équipe 7	**Équipe 8**	**Équipe 9**
Guillaume Lépine	Gontran Segue Nzouba	Francis Bonneau
Robin Caron	Pier-Olivier Clément	Vincent Beausoleil
Bogdan Alexandru Marinescu	Fabrice Houle	Raby Chaabani
Alix Pedneault-Plourde	Pierre-Mary Bien-Aimé	

Réalisations attendues :

1. En utilisant la méthode de migration demandée à votre équipe, créer une table HBASE permettant de répondre le plus efficacement possible à une requête HBase Scan équivalente au code SQL suivant :
   SELECT hg18.chromosome_id, hg18.position
   FROM ngs_hg18_position hg18, ngs_position p
   WHERE hg18.id = p.hg18_id AND p.hg18_id > @range1 AND p.hg18_id < @range2 AND p.hg19_id > 0;

   Note 1: @range1 et @range2 sont des valeurs qui peuvent être choisies par l'utilisateur.

2. Créer la ou les vues SQL nécessaires pour combiner les données de ces tables (selon la méthode qui est assignée à votre équipe). **Indiquez le code pour les vues**. Indiquez le RowKey que vous avez choisi et pourquoi.
   Note : vous pouvez utiliser une technique alternative à des vues SQL. Dans ce cas expliquez pourquoi, et expliquez en détails votre technique.
3. Importer les données dans HBASE à partir des vues (ou de la technique alternative). **Indiquez le code pour l'import.**
4. Faire un scan des valeurs dans HBASE pour les valeurs de range : **@range1 = 100000 et @range2 = 1000000**. Afin de sauvegarder vos résultats dans hbase shell utiliser :
   echo "<requête de hbase shell>" | hbase shell > test.txt
5. **Indiquez le code pour scan, le nombre de résultats, ainsi que le temps d'exécution de la requête.**

- **Tâche 3: Optimisation d'une requête SQL à l'aide d'HBase.**

Vous devez réaliser un schéma de table(s) HBASE permettant de solutionner la requête suivante :

SELECT v.id, v.variant_class, v.strand, v.variant_genotype, v.dbSNP_rs,

v.variant_type_id as s2d_type, v.gene, v.freq_1000g, v.Sift_score, qvp.chromosome_name as chromosome, qvp.hg18_position as hg18_chrom_position, qvp.hg19_position as hg19_chrom_position
FROM NGS_variant v, NGS_queryVariantPosition qvp
WHERE v.position_id = qvp.position_id AND v.id in (@resultSet) AND v.variant_type_id = @variantType
GROUP BY v.id

Les variables : @variantType est une constante fournie, @resultSet est une liste fournie. L'objectif est de pouvoir exécuter l'équivalent de la requête SQL le plus rapidement possible dans HBASE.

Réalisations Attendues :

1. Utiliser la ou les méthode(s) de migration de votre choix. Décrivez pourquoi vous pensez que cette méthode est la plus efficace.
2. Faire un diagramme du schéma de(s) table(s) HBASE que vous avez créé pour cette tâche. Ce diagramme doit montrer clairement le **nom des tables**, les **familles de colonnes**, et **leurs colonnes**. La **composition de la RowKey** doit être précisée également.
3. Documenter la méthode d'importation des données dans HBase. Si vous utilisez Sqoop, **indiquez le code pour les vues SQL créées, ansi que le code Sqoop pour l'importation**. Si vous n'utilisez pas Sqoop, documenter votre méthode d'importation de façon détaillée.
4. Faite un / des scans dans HBASE pour les valeurs suivantes :
   @variantType = 4
   @resultSet =
   494121,494122,494123,494124,494125,494128,494129,494130,494131,
   494132,494133,494134,494136,494137,494138,494139,494140,494141
5. **Indiquez le code de la requête Scan, le temps total pris pour l'exécuter, ainsi qu'une capture d'écran des 5 premiers résultats**.

- **Instructions pour la soumission du rapport**
  - Remettez votre rapport papier avant de commencer le deuxième laboratoire sur le comptoir du chargé de laboratoire.
  - Les documents électroniques à remettre sont : le rapport ainsi que tous les scripts générés lors du laboratoire (les commandes HBase, les vues PostgreSQL et les commandes Sqoop). Envoyer tous les documents à l'adresse **gti780.e2014@gmail.com**.
  - **Note 5 :** Tous les fichiers doivent être compressés et être nommés GTI780-Labo 2-ÉquipeX, où X est le numéro de votre équipe. **Vous allez perdre des points si vous ne suivez pas ces consignes**.

- **Conseils techniques**

Voici des références utiles pour le laboratoire 2:

- Pour configurer votre machine / VM de l'école voir le document **GTI780 Instructions**, partie LABO 2.
- Créer le schéma relationnel et les tables. Le document de contexte explique les schémas. Le site officiel de HBase contient la documentation à l'adresse http://hbase.apache.org/book/quickstart.html.
- Pour charger les données, l'outil de Sqoop doit être utilisé http://sqoop.apache.org/.
- Chargement des données: Les données complètes sont disponibles à l'adresse **http://10.194.32.150/datadb**. Il est possible d'utiliser des scripts pour charger les données.

- **Grille d'évaluation**
  Toutes les informations relatives à l'évaluation du rapport et de la façon dont cette évaluation sera menée sont disponibles dans le fichier Grille de correction GTI780 BigData TP 2.

- **Pénalités de retard**
  Une pénalité de 10% par jour, y compris les jours de week-end, sera appliquée à tous les travaux soumis en retard. Ceci ne s'applique que pour le premier jour de retard. Dès le deuxième jour, tout rapport non soumis aura automatiquement une note de 0.

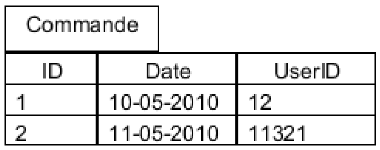
Méthodes de migration de SQL vers NoSQL

Exemple :

Il y a 3 méthodes de migrations que nous allons utiliser dans ce laboratoire :

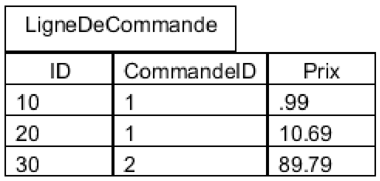
- Méthode 1 : une table avec 1 famille de colonnes (dénormalisation classique)
- Méthode 2 : 2 tables
- Méthode 3 : une table avec 2 (ou plus) familles de colonnes.

Afin de mieux illustrer les techniques nous allons assumer le schéma relationnel suivant :



Commande

ID	Date	UserID
1	10-05-2010	12
2	11-05-2010	11321



LigneDeCommande

ID	CommandeID	Prix
10	1	.99
20	1	10.69
30	2	89.79

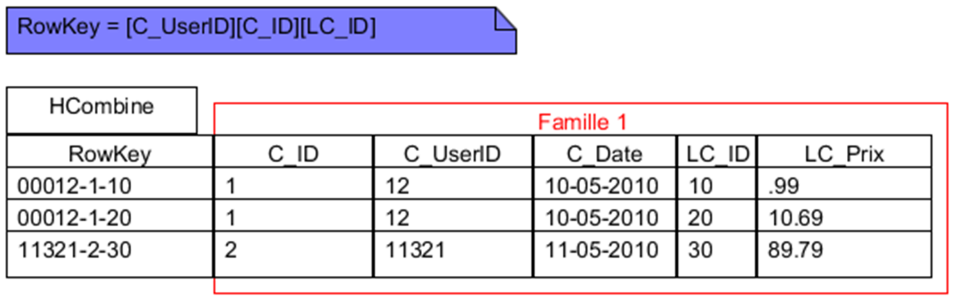
Une commande est identifiée de façon unique (clef primaire) avec le champs Commande.ID . Le UserID est l'identifiant de l'acheteur, alors que la date est la date d'achat. Une commande est composée de plusieurs lignes de commandes.

Une Ligne de Commande est identifiée de façon unique avec le champ LigneDeCommande.ID et peut être reliée à sa Commande à l'aide de CommandeID. Pour simplifier l'exemple, tous les champs ont été retirés à l'exception du Prix.
Dans notre cas d'utilisation, on connaît toujours le UserID lorsqu'on veut accéder à une commande ou une ligne de commande.

Méthodes de migrations :

Méthode 1 : Dénormalisation classique

La 1ière méthode consiste à combiner 2+ tables relationnelles pour créer une table HBase avec une famille de colonnes. Voici ce qu'on obtiendrait si on combinait les 2 tables Commande et LigneDeCommande de l'exemple.



RowKey = [C_UserID][C_ID][LC_ID]

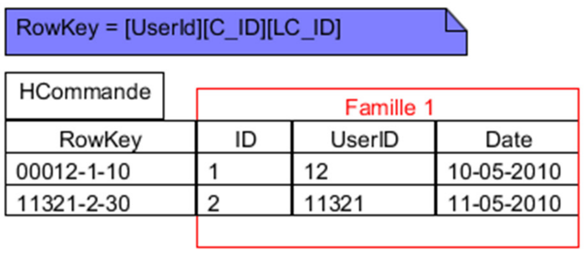
HCombine

	Famille 1				
RowKey	C_ID	C_UserID	C_Date	LC_ID	LC_Prix
00012-1-10	1	12	10-05-2010	10	.99
00012-1-20	1	12	10-05-2010	20	10.69
11321-2-30	2	11321	11-05-2010	30	89.79

Le RowKey est une combinaison des champs Commande.USER_ID, Commande.ID, et LigneDeCommande.ID . Remarquez que la partie USER_ID de la RowKey est paddé avec des zéros pour que HBase stocke les données triées par RowKey. Vous devez faire les *paddings* nécessaire dans vos vues SQL pour vous assurez que la taille des données de chaque colonnes est la même.

Méthode 2 : Deux tables relationnelles donnent 2 tables dans HBase

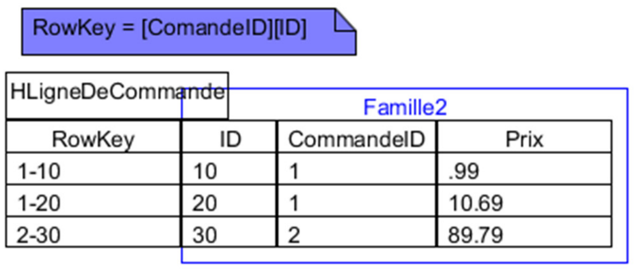
La 2éme méthode consiste à créer deux tables HBase avec le même nombre de colonnes que les tables Postgres. L'idée est d'intégrer la relation dans le RowKey. Si on prend les tables initiales de l'exemple, on obtiendrait :



RowKey = [UserId][C_ID][LC_ID]

HCommande

	Famille 1		
RowKey	ID	UserID	Date
00012-1-10	1	12	10-05-2010
11321-2-30	2	11321	11-05-2010



RowKey = [ComandeID][ID]

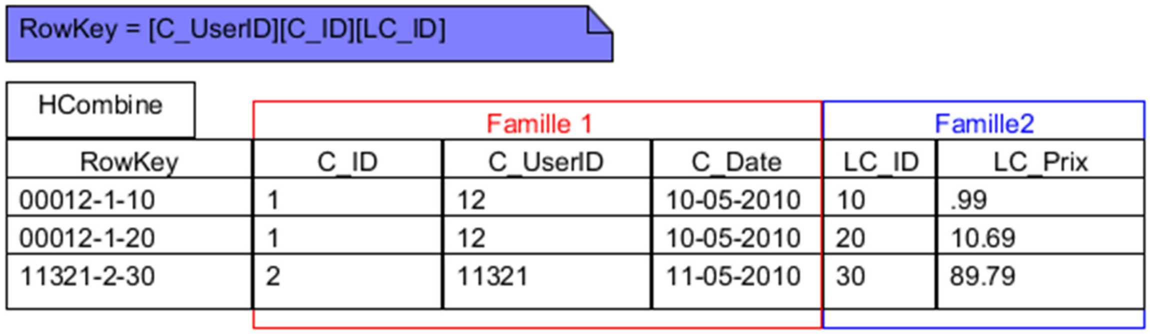
HLigneDeCommande

	Famille2		
RowKey	ID	CommandeID	Prix
1-10	10	1	.99
1-20	20	1	10.69
2-30	30	2	89.79

Comme vous pouvez le constater la première partie du RowKey de HLigneDeCommade est le id de commande. Donc si vous voulez que toutes les lignes de commande d'une commande en particulier, c'est facile de l'obtenir avec un objet scan. Par exemple, pour obtenir toutes les lignes de commande de la commande #1, on peut faire un Scan dans HBase sur les RowKey entre « 1-00 » et « 1-99 ».

Méthode 3 : Une table HBase avec 2 familles de colonnes

La 3ème méthode consiste à combiner les deux tables Postgres en une table HBase qui a deux familles de colonnes, 1 famille de colonne par table SQL. Voici le résultat dans HBase :



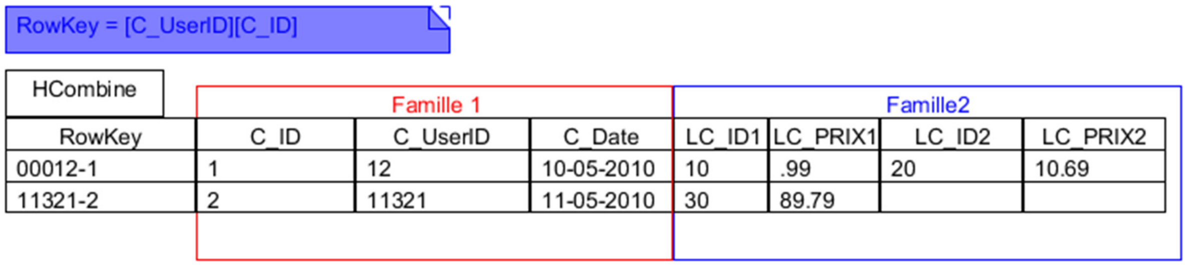
RowKey = [C_UserID][C_ID][LC_ID]

HCombine

	Famille 1			Famille2	
RowKey	C_ID	C_UserID	C_Date	LC_ID	LC_Prix
00012-1-10	1	12	10-05-2010	10	.99
00012-1-20	1	12	10-05-2010	20	10.69
11321-2-30	2	11321	11-05-2010	30	89.79

Normalement, on regroupe les colonnes dans les familles de colonnes pour des raisons de performance. Cependant, pour ce laboratoire veuillez utiliser 1 famille de colonne par table SQL.

Note : Avez-vous constatez que les données de la table Commande sont répétées plusieurs fois dans la méthode 1 et 3? Il existe une variante qui permet de contourner ce problème et de minimiser le nombre de rangées en incluant le LC_ID dans le nom de la colonne (HBase permet de créer un nombre variable de colonnes pour chaque rangée). Voici un exemple :



RowKey = [C_UserID][C_ID]

HCombine

	Famille 1			Famille2			
RowKey	C_ID	C_UserID	C_Date	LC_ID1	LC_PRIX1	LC_ID2	LC_PRIX2
00012-1	1	12	10-05-2010	10	.99	20	10.69
11321-2	2	11321	11-05-2010	30	89.79		

En regardant la famille 2, on remarque que la ligne 1 a deux lignes de commandes, et 1 seule pour la ligne 2. Toutefois, vous n'êtes pas tenu d'utiliser cette variante car elle est un peu plus complexe à migrer. Pour plus d'information, voir le document « **How_To_Migrate_SQL_to_NoSQL.pdf** ».

Comment migrer les données de PostgreSQL vers HBASE:

a. Créez une vue ou une table dans POSTGRES créez une vue qui combine les données (voir http://www.postgresql.org/docs/9.1/static/sql-createview.html) pour la doc.

   Exemple de création de vue pour la table screening_pipeline (avec *padding* sur les champs id et sequencer_id):

```
CREATE VIEW david4 AS
SELECT
   CONCAT(
      lpad(id::text, 3, '0'),
      '-',
      lpad(sequencer_id::text, 3, '0')
   ) AS pipeline_sequencer_id,
   *
FROM screening_pipeline;
```

   Cela nous permet de générer facilement le champ **pipeline_sequencer_id** qu'on désire utiliser comme RowKey dans HBase. Le fait de « *padder* » les champs id et sequencer_id avec des zero permet de conserver les données triées dans l'ordre avec HBase. Lorsqu'on fera une opération scan, les données retournées seront triées par RowKey.

   Avant de faire une vue, il est recommandé de tester si votre fonction de *padding* fonctionne correctement. Par exemple :

```
SELECT CONCAT( lpad(id::text, 3, '0'), '-', lpad(sequencer_id::text, 3, '0') ) AS
pipeline_sequencer_id, *
FROM screening_pipeline
LIMIT 10;
```

   IMPORTANT: Si vous trouver que les vues sont trop lentes, vous pouvez créer une autre table en utilisant : **SELECT INTO … ,** Voir http://www.postgresql.org/docs/9.1/static/sql-selectinto.html pour plus d'info.

b. Pour sqoop :
   i. Assurez vous que vous êtes loggués en tant que hduser :
      sudo su - hduser

   ii. Tapez **jps** pour vous assurer que tous les processus Hadoop sont activés :

```
13864 HMaster
13074 JobTracker
13325 TaskTracker
14137 HRegionServer
12484 NameNode
12993 SecondaryNameNode
12727 DataNode
```

20245 Jps
13801 HQuorumPeer
Sinon démarrer hadoop et hbase :
start-dfs.sh
start-mapred.sh
start-hbase.sh

iii. Si vous avez suivi le document technique vous avez créer l'utilisateur **hduser** dans Postgres. Rendez le SUPERUSER :
psql --host=127.0.0.1 --username=dbadmin postgres
Pass: **bonjour@123**
ALTER ROLE hduser WITH SUPERUSER;

iv. **Tester l'importation des dans HBase à l'aide de Sqoop**

sqoop import --connect jdbc:postgresql://localhost:5432/postgres --username hduser --password ChangeIt --table david4 --target-dir /hbase/ --hbase-table david4b --columns pipeline_sequencer_id,id,sequencer_id,sequence_aligner_algorithm_id,variant_detection_algorithm_id,variant_annotation_algorithm_id --hbase-row-key pipeline_sequencer_id --split-by pipeline_sequencer_id --column-family d --hbase-create-table –verbose

Explications :
--connect la dernière partie à la fin indique le nom de la BD. Dans notre cas c'est **postgres**
--username : Utilisateur dans postgres
--password : Pass du utilisateur dans postgres
--table : table / vues dans la bd
--target-dir : Répertoire racine de HBase dans HDFS. Ne pas changer.
--hbase-table : Nom de la table dans HBASE
--columns : colonnes qu'on veut importer dans HBASE
--split-by : La colonne à utiliser comme RowKey dans HBase
--column-family : Le nom de la column family dans lesquels les colonnes vont être ajoutées.

Note 1 : Vous pouvez spécifier un seul column family par import. (Si vous avez 2 familles de colonnes vous faites 2 import).
Note 2 : Attention aux espaces en trop. Scoop à de la misère à s'exécuter.

--hbase-create-table : mettez ce flag si vous voulez créer la table dans HBASE quand vous

Pour plus d'informations :
http://sqoop.apache.org/docs/1.4.4/SqoopUserGuide.html

c. Tester que ça fonctionne:

```
hbase shell
scan 'david4', {STARTROW => '001', ENDROW => '005'}
```

Résultat :

```
001-003    column=d:id, timestamp=1400823152974, value=1
001-003    column=d:sequence_aligner_algorithm_id, timestamp=1400823152974, value=5
001-003    column=d:sequencer_id, timestamp=1400823152974, value=3
001-003    column=d:variant_annotation_algorithm_id,timestamp=1400823152974, value=3
001-003    column=d:variant_detection_algorithm_id, timestamp=1400823152974, value=7
002-004    column=d:id, timestamp=1400823152974, value=2
002-004    column=d:sequence_aligner_algorithm_id, timestamp=1400823152974, value=5
002-004    column=d:sequencer_id, timestamp=1400823152974, value=4
002-004    column=d:variant_annotation_algorithm_id, timestamp=1400823152974, value=3
002-004    column=d:variant_detection_algorithm_id, timestamp=1400823152974, value=7
```

APPENDIX V

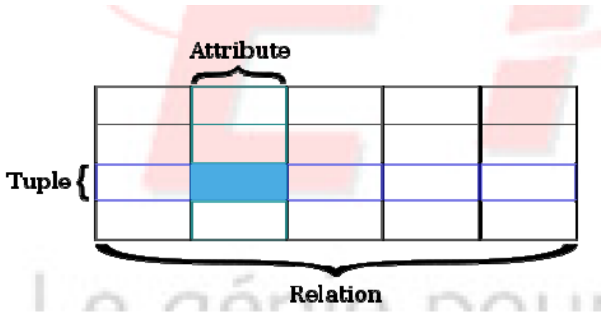
TRAINING DOCUMENT

Migration from relational databases to no-sql databases Prepared by: Abraham GÓMEZ

RELATIONAL DATABASES

Overview

A relational database is a database that has a collection of tables of data items, all of which is formally described and org nized according to the relational model.
In the relational model, each table schema must identify a primary column used for identifying a row called the primary key. Tables can relate by using a foreign key that points to the primary key of another table. The relational model offers various levels of refinement of the table relati ns called database normalization. The database management system (DBMS) of a relational database is called an RDBMS, and is the software of a relational database. Here a figure of this model:



Tables

A table is defined as a set of tuples that have the same attributes. A tuple usually represents an object and information about that object. Objects are typically physical objects or concepts. The tables are organized into rows and columns. All the data referenced by an attribute are in the same domain and conform to the same constraints. The relational model specifies that the tuples of a table have no specific order and that the tuples, in turn, impose no order on the attributes. Applications access data by specifying queries, which use operations such as select to identify tuples, project to identify attributes, and join to combine tables. Tables can be modified using the insert, delete, and update operators.

Constraints

Constraints make it possible to further restrict the domain of an attribute. For instance, a constraint can restrict a given integer attribute to values between 1 and 10. Constraints provide one method of implementing business rules in the database. SQL implements constraint functionality in the form of check constraints.

Primary keys

A primary key uniquely specifies a tuple within a table. In order for an attribute to be a good primary key it must not repeat. While natural attributes (attributes used to describe the data being entered) are sometimes good primary keys, surrogate keys are often used instead. A surrogate key is an artificial attribute assigned to an object

Foreign key

A foreign key is a field in a relational table that matches the primary key column of another table. The foreign key can be used to cross-reference tables. Foreign keys need not have unique values in the referencing relation.

Stored procedures

A stored procedure is executable code that is associated with, and generally stored in, the database. Stored procedures usually collect and customize common operations, like inserting a tuple into a relation, gathering statistical information about usage patterns, or encapsulating complex business logic and calculations.

Index

An index is one way of providing quicker access to data. Indices can be created on any combination of attributes on a relation. Queries that filter using those attributes can find matching tuples randomly using the index, without having to check each tuple in turn. This is analogous to using the index of a book to go directly to the page on which the information you are looking for is found, so that you do not have to read the entire book to find what you are looking for.

Cardinality

The cardinality of one data table with respect to another data table is a critical aspect of database design. Relationships between data tables define cardinality when explaining how each table links to another.
In the relational model, tables can be related as any of: one-to-one, many-to-one (or one-to- many), and many-to-many.

For example, consider a database designed to keep track of hospital records. Such a database could have many tables like:

- A Doctor table full of doctor information
- A Patient table with patient information
- And a Department table with an entry for each department of the hospital. In that model:
- There is a many-to-many relationship between the records in the doctor table and records in the patient table (Doctors have many patients, and a patient could have several doctors);
- A one-to-many relation between the department table and the doctor table (each doctor works for one department, but one department could have many doctors).

The one-to-one relationship is mostly used to split a table in two in order to optimize access or limit the visibility of some information. In the hospital example, such a relationship could be used to keep apart doctors' personal or administrative information.

No-SQL Databases: HBase

Overview

HBase is an open-source, non-relational, distributed database modeled after Google's BigTable and is written in Java. It is developed as part of Apache Software Foundation's Apache Hadoop project and runs on top of HDFS (Hadoop Distributed Filesystem), providing BigTable-like capabilities for Hadoop. That is, it provides a fault-tolerant way of storing large quantities of sparse data. Since HBase is a distributed database the main database will be in the master server and the others server will be called region servers.

HBase is column oriented

A regular SQL schema can be designed as follows:

Student Table			
student_ID varchar(2) PK	name varchar(30)	age integer	Sex char(1)
1	John	25	M
2	Mike	32	M
3	Anna	19	F
4	Steve	28	M

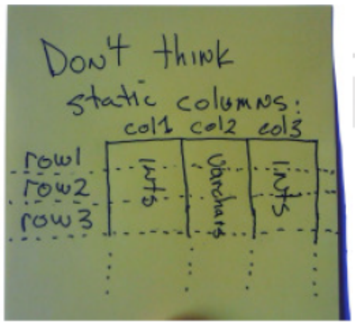
The relational databases have row-oriented storage (they are organized by rows):

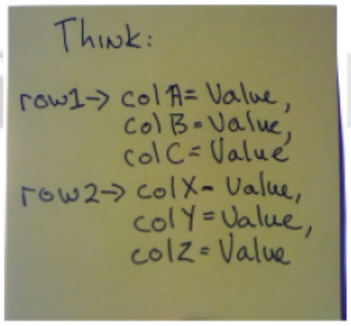
Row 1	1	John	25	M
Row 2	2	Mike	32	M
Row 3	3	Anna	19	F

HBase has column oriented storage, it means, it is organized by columns:

Col 1: name	John	Mike	Anna
Col 2: age	25	32	19
Col 3: sex	M	M	F

Columns in HBase are grouped into column families. All column members of a col family have the same prefix. For example, the columns info:name and info:age are members of the info column family. The colon character (:) delimits the column family the columns of the same family are recorded in the same region server.





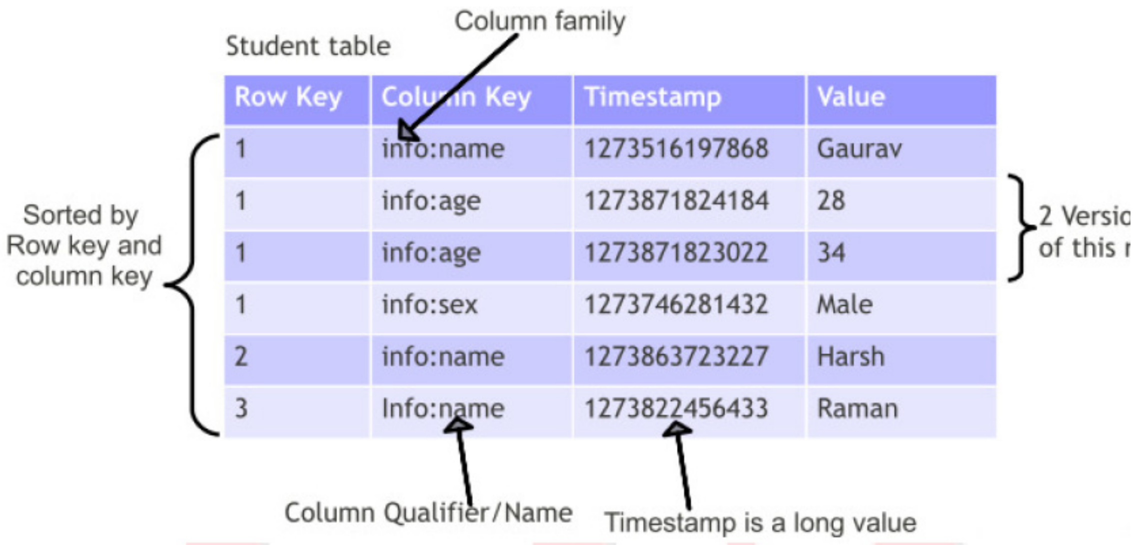
How does HBase work?

HBase has two types of nodes: the master and the region server. HBase only can have one master at a time. The master manages the cluster operations, the assignment, the load balancing and the splitting. It does not part of the read/write operation.

HBase can have one or more region servers. They host the tables; performs the reads, manage the buffers writes. Also, the clients can talk directly to them for reads/writes.

HBase schema design

HBase is a big sorted map and to obtain a cell value, you have to enter the Row Key+ Column Key + timestamp.

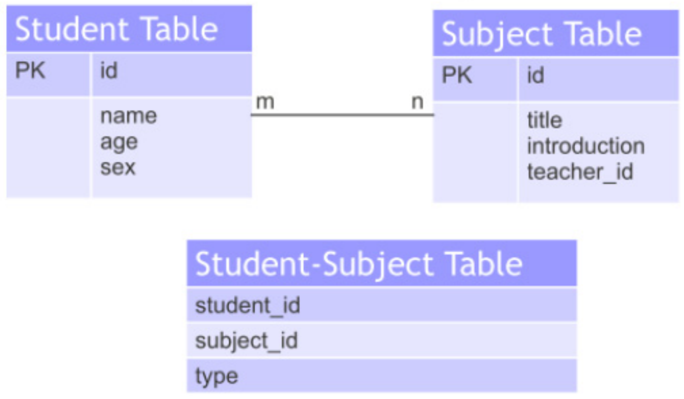

Student table
Column family
Sorted by Row key and column key
2 Versio
of this r
Column Qualifier/Name
Timestamp is a long value

Row Key	Column Key	Timestamp	Value
1	info:name	1273516197868	Gaurav
1	info:age	1273871824184	28
1	info:age	1273871823022	34
1	info:sex	1273746281432	Male
2	info:name	1273863723227	Harsh
3	Info:name	1273822456433	Raman

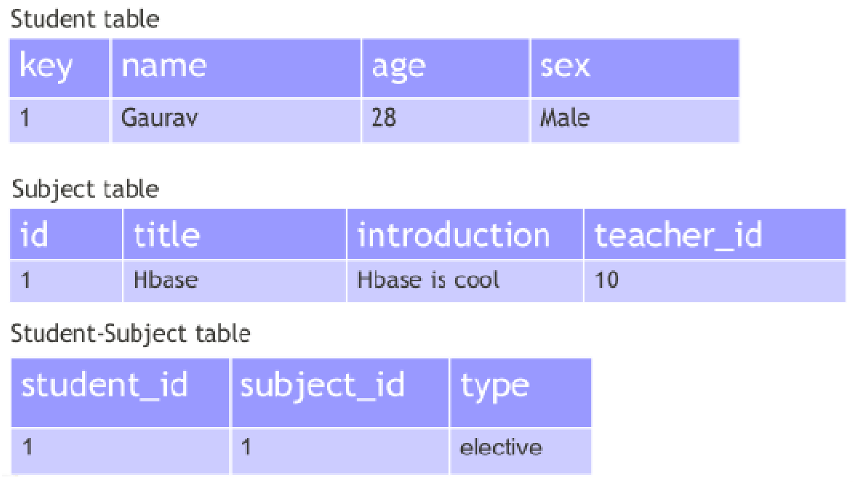
Migration Use Cases

Relational and No-SQL are two different worlds, to obtain a good enough migration y must implement at least the idea of denormalize and duplicate data and build a good r key.

Example of a Student and Subject



Data Examples



Student table

key	name	age	sex
1	Gaurav	28	Male

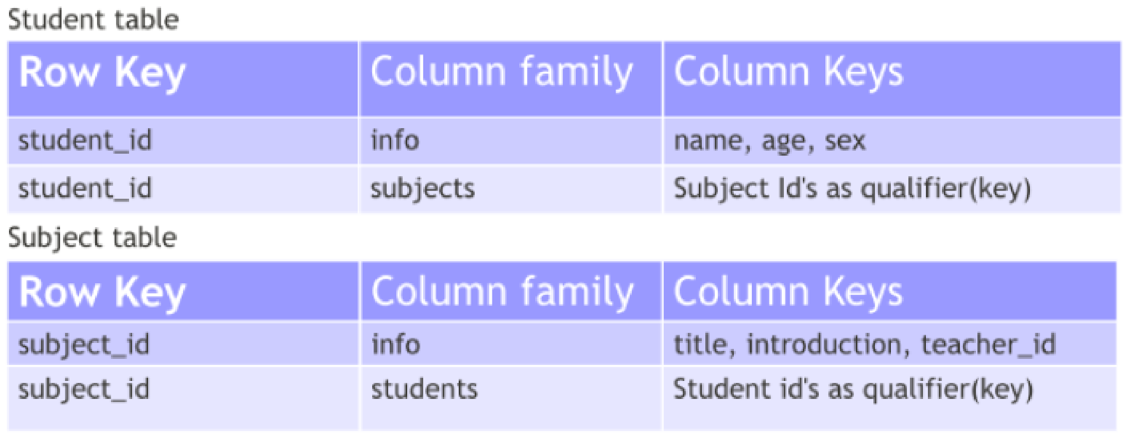
Subject table

id	title	introduction	teacher_id
1	Hbase	Hbase is cool	10

Student-Subject table

student_id	subject_id	type
1	1	elective

HBase Schema



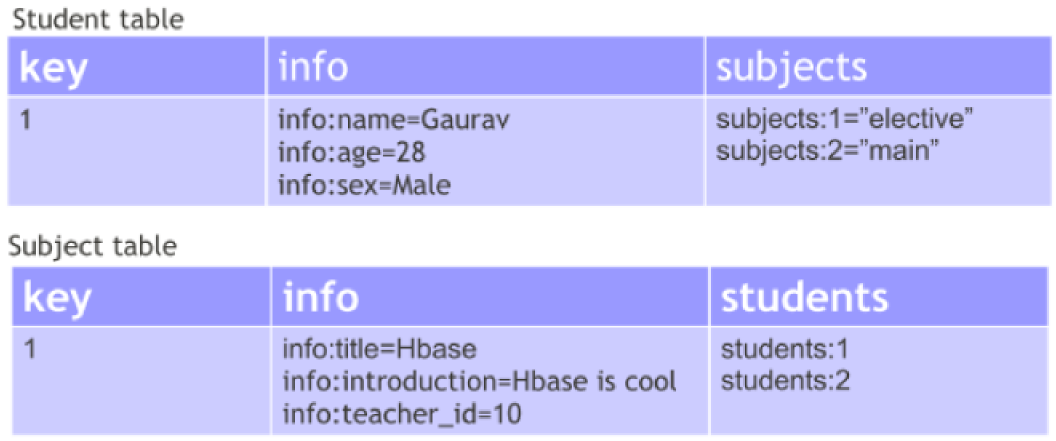
Student table

Row Key	Column family	Column Keys
student_id	info	name, age, sex
student_id	subjects	Subject Id's as qualifier(key)

Subject table

Row Key	Column family	Column Keys
subject_id	info	title, introduction, teacher_id
subject_id	students	Student id's as qualifier(key)

Data Examples



Student table

key	info	subjects
1	info:name=Gaurav info:age=28 info:sex=Male	subjects:1="elective" subjects:2="main"

Subject table

key	info	students
1	info:title=Hbase info:introduction=Hbase is cool info:teacher_id=10	students:1 students:2

APPENDIX VI

GENERAL INSTRUCTIONS FOR THE PARTICIPANTS

Instructions to follow during the session:

- The participation in the experimentation is a volunteer work. If you decide, for any reason, to leave the session, please inform to the organizer in order to return back all documents related with the experiment and destroy them.

Please do not communicate with other participants during the session.

1. All the experiment will have four parts: Introduction, training session, experiment and survey.

2. Introduction: The goal of the initial introduction is to provide the context.

3. The training session: Listen carefully all the instructions provided by the session organizer. If you have any questions, do not hesitate to ask. In the training session you will receive an overview about relation DB, No-SQL database and an example of the migration process (at schema's level).

4. The experiment: At the beginning of the experiment each participant will receive:
    - A "participant code", please write this code in all your documents that you are going to receive.
    - A yellow envelope with four types of documents:
    - The document with the training example (white sheets).
    - One blue sheet with the synthetic relational schema that will be migrated to No-SQL. This database schema is totally different to the other database schema, presented in the previous training document.
    - One green sheet where the participant will write the No-SQL schema resulting from the conversion/migration process.
    - Several yellow sheets that can be used as drafts.

The first recommended step is to read the document "training document: migration from relational databases to no-SQL databases".

- Analyze how the example in the document was used to make the migration from the relational database to No-SQL environment.
- After finishing the migration process and design your response schema in the green sheet, please make sure that your "participant code" is written in all the documents used in the experiment.
- Besides, return all the documents used and not used in the experimentation to the organizer into the yellow envelope.

5. Survey: After finishing the experiment part, please fill the "participant experience survey" form.

- If you have questions about this experiment, please contact: Abraham Gomez: abraham-segundo.gomez.1@ens.etsmtl.ca
- This experiment has been designed in accordance with the policies of the ETS Ethics committee.

BIBLIOGRAPHY

Abran, A., Laframboise, L., & Bourque, P. (2003). A Risk Assessment Method and Grid for Software Measurement Programs.

Agildata. (2011). DATABASE SHARDING: The Rise Of DataBase Sharding. Retrieved on September 5, from http://www.agildata.com/database-sharding/

Anderson, A., Anthonio, S. (2016). Benefits and disadvantages of Cloud-Computing. Retrieved from https://www.linkedin.com/pulse

Arango, H., Domingues, E. G., Policarpo, G. A. J., & Hermeto, A. E. (2002). Analogies between quality improvement in multiphase electrical systems and financial markets. Dans *10th International Conference on Harmonics and Quality of Power. Proceedings (Cat. No.02EX630)* (Vol. 1, pp. 301-303 vol.301). doi: 10.1109/ICHQP.2002.1221449

Babu, A., & Surendran, S. (2017). *Relational to NoSQL Database Migration* présentée à National Conference on Advanced Computing, Communication and Electrical Systems - (NCACCES'17), KMEA Engineering College, Kerala- 683561, India. Retrieved from https://www.ijirset.com/upload/2017/ncacces/11_CAMERA%20RAEDY%20PAPER-TK001-004-PC04.pdf

Barrett, R. (2009). Transaction accross Datacenters - App Engine Track [Conference]. Retrieved on 2009 from https://snarfed.org/transactions_across_datacenters_io.html

Biswapesh Chattopadhyay, L. L., Weiran Liu, Sagar Mittal, Prathyusha Aragonda, Vera Lychagina, Younghee Kwon, Michael Wong (2011). Tenzing A SQL Implementation On The MapReduce Framework Dans (pp. pp. 1318-1327). VLDB Endowment. Repéré à http://research.google.com/pubs/pub37200.html

Bohn, R. B., Messina, J., Liu, F., Tong, J., & Mao, J. (2011). NIST Cloud Computing Reference Architecture. Dans *2011 IEEE World Congress on Services* (pp. 594-596). doi: 10.1109/SERVICES.2011.105

Chongxin, L. (2010). Transforming relational database into HBase: A case study. Dans *Software Engineering and Service Sciences (ICSESS), 2010 IEEE International Conference on* (pp. 683-687). doi: 10.1109/icsess.2010.5552465

Cinnamond, S. (Producteur). (2013, January 2017). MySql to HBase in 5 steps. *TerraMeta Software inc.* Retrieved from https://www.slideshare.net/scinnamond/wordnet-hbase

ComScore. (2016). Global Search Market Draws More than 100 Billion Searches per Month. Retrieved from https://www.comscore.com/por/Insights

Cook, B. (2009). Scaling Twitter: Making Twitter 10000 Percent Faster. Retrieved on September 5 from http://highscalability.com

Cordes, K. (2007). Google Tech Talk on scalability: YouTube Scalability Talk. Retrieved on September 5, from https://kylecordes.com/2007/youtube-scalability

Cryans, J.-D., April, A., & Abran, A. (2008). Criteria to compare cloud computing with current database technology. Dans *International Workshop on Software Measurement, IWSM 2008, DASMA Software Metrics Congress, MetriKon 2008, and International Conference on Software Process and Product Measurement, Mensura 2008, November 18, 2008 - November 19, 2008* (Vol. 5338 LNCS, pp. 114-126). Springer Verlag. Retrieved from http://dx.doi.org/10.1007/978-3-540-89403-2-11

Cryans, J., April, A., & Abran, A. (2008). Criteria to Compare Cloud Computing with Current Database Technology *IWSM / MetriKon / Mensura 2008, LNCS 5338*, 114-168.

Dean, J. (2009). Designs, Lessons and Advice from Building Large Distributed Systems. Retrieved from http://www.cs.cornell.edu/projects/ladis2009

Dean, J., & Ghemawat, S. (2004). MapReduce: Simplified Data Processing on Large Clusters *Proceedings of the 6th conference on Symposium on Operating Systems Design & Implementation, San Francisco, CA*, 1-13.

DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., . . . Vogels, W. (2007). Dynamo: amazon's highly available key-value store. *SIGOPS Oper. Syst. Rev., 41*(6), 205-220.

deRoos, D. (2014). *Hadoop For Dummies*. Wiley. Retrieved on September 5, 2017 from https://books.google.ca/books?id=wJApAgAAQBAJ

Donno, M. D., Tange, K., & Dragoni, N. (2019). Foundations and Evolution of Modern Computing Paradigms: Cloud, IoT, Edge, and Fog. IEEE Access, 7, 150936-150948. doi: 10.1109/ACCESS.2019.2947652

Driscoll, J., Sarnak, N., Sleator, D., & Tarjan, R. (1986). Making data structures persistent. *Proceedings of the eighteenth annual ACM symposium on Theory of computing*. doi: http://doi.acm.org/10.1145/12130.12142

Easterbrook, S., Singer, J., Storey, M.-A., & Damian, D. (2008). Selecting Empirical Methods for Software Engineering Research. Dans F. Shull, J. Singer & D. I. K. Sjøberg (Éds.), *Guide to Advanced Empirical Software Engineering* (pp. 285-311). London: Springer London. doi: 10.1007/978-1-84800-044-5_11. Retrieved from 10.1007

Fay, C., Dean, J., Ghemawat, S., Hsieh, W., Wallach, D., Burrows, M., . . . Robert., G. (2006, Seattle, Novembre 2006). *Bigtable: A Distributed Storage System for Structured Data* présentée à OSDI 06, 7th USENIX Symposium on Operating Systems Design and Implementation, Seattle, WA, Seattle, WA.

Fong, J., & Bloor, C. (1994). Data conversion rules from network to relational databases. *Information and Software Technology, 36*(3), 141-153. doi: 10.1016. Retrieved from http://www.sciencedirect.com/science/article/pii/0950584994900531

Fong, J., & Huang, S. M. (1997). *Information Systems Reengineering*. Singapore:Springer-Verlag Singapore Pte. Ltd.

Foster, I., Yong, Z., Raicu, I., & Lu, S. (2008). Cloud Computing and Grid Computing 360-Degree Compared. *Grid Computing Environments Workshop, 2008. GCE '08*, 1-10.

Fowler, H. W. (2001). The New Pocket Oxford Dictionary. Dans. Oxford, United Kingdom: Oxford University Press.

Fiore, S., & Aloisio, G. (2011). Grid and Cloud Database Management. Springer. Retrieved from http://books.google.ca/books?id=7iD4gPDke6IC

Gartner. (2019). Cloud Computing. Retrieved from https://www.gartner.com/

Ghemawat, S., Gobioff, H., & Leung, S. (2003). The Google File System. *19th ACM Symposium on Operating Systems Principles, New York, USA*, 15.

Godin, R. (2006). Systèmes de gestion de bases de données par l'exemple. *2*.

Gomez, A., Ouanouki, R., April, A., & Abran, A. (2014). Building an Experiment Baseline in Migration Process from SQL Databases to Column Oriented No-SQL Databases. *Journal of Information Technology & Software Engineering*.

Hanine, M., Bendarag, A., & Boutkhoum, O. (2015). Data Migration Methodology from Relational to NoSQL Databases. *International Journal of Computer and Information Engineering, 9*(12).Retrieved from https://waset.org/publications/10004179/data-migration-methodology-from-relational-to-nosql-databases

Harris, D. (2011). Facebook trapped in MySQL 'fate worse than death'. Retrieved from https://gigaom.com/2011/07/07/facebook-trapped-in-mysql-fate-worse-than-death/

Higginbotham, S. (2011, 2011). Once Again, See How Twitter Scales. Retrieved on September 5 from https://gigaom.com/2011/05/03/once-again-see-how-twitter-scales/

Hudicka, J. R. (1998). An Overview of Data Migration Methodology. Retrieved from http://dulcian.com/articles/overview_data_migration_methodology.htm

ISO/IEC. (2009). *A standardization initiative for Cloud computing*. Initiative.

Jimbojw. (2008). Understanding HBase and BigTable. Retrieved on September 5, 2015 from http://sandmann.sdf.org/toledo/bigdata/2015/lib/HBase%20-%20Modeling.pdf

Kasunic, M. (2005). Designing an Effective Survey. Retrieved from http://www.dtic.mil

Klos, A. (2012). *Optimisation de recherche grâce à Hbase sous Hadoop*. ETS University. Retrieved from http://publicationslist.org/data/a.april

Koopmann, J. (2008). Si what is an Oracle Nested Table? Retrieved on September 5, 2009 from https://www.databasejournal.com/features/oracle/article.php/3788331/So-what-is-an-Oracle-Nested-Table.htm

Lars, G. (2013a). The Apache HBase Reference Guide [Online]. Retrieved from http://hbase.apache.org/book/rowkey.design.html

Lars, G. (2013b). The Apache HBase Reference Guide #1 [Online]. Retrieved on March 11th, 2013 from http://hbase.apache.org/book.html#schema

Lars, G. (2013c). The Apache HBase Reference Guide #2 [Online]. Retrieved from March 11th, 2013 from http://hbase.apache.org/book/columnfamily.html

Lars, G. (2013d). The Apache HBase Reference Guide #3 [Online]. Retrieved from 11th, 2013 from http://hbase.apache.org/book.html#ops.capacity

Lars, G. (2013e). The Apache HBase Reference Guide #4 [Online]. Retrieved from 11th, 2013 from http://hbase.apache.org/book.html#schema.joins

Lee, C. H., & Zheng, Y. L. (2015a). Automatic SQL-to-NoSQL schema transformation over the MySQL and HBase databases. Dans *2015 IEEE International Conference on Consumer Electronics - Taiwan* (pp. 426-427). doi: 10.1109/ICCE-TW.2015.7216979

Lee, C. H., & Zheng, Y. L. (2015b). SQL-to-NoSQL Schema Denormalization and Migration: A Study on Content Management Systems. Dans *2015 IEEE International Conference on Systems, Man, and Cybernetics* (pp. 2022-2026). doi: 10.1109/SMC.2015.353. Retrieved from http://ieeexplore.ieee.org/document/7379485/

Lethbridge, T. C. (1998). *A Survey of the Relevance of Computer Science and Software Engineering Education* présentée à Proceedings of the 11th Conference on Software Engineering Education and Training.

Li, N., Xu, B., Zhao, X., & Deng, Z. (2011). Database Conversion Based on Relationship Schema Mapping. Dans *2011 International Conference on Internet Technology and Applications* (pp. 1-5). doi: 10.1109/ITAP.2011.6006302

Liming, L. (2008). Introduction To The Cloud Computing.

Maatuk, A., Ali, A., & Rossiter, N. (2008). *Relational Database Migration: A Perspective* présentée à Proceedings of the 19th international conference on Database and Expert Systems Applications, Turin, Italy. doi: 10.1007/978-3-540-85654-2_58

Marcos, E. (2005). Software engineering research versus software development. *SIGSOFT Softw. Eng. Notes, 30*(4), 1-7. doi: 10.1145/1082983.1083005

Mell, P., & Grance, T. (2011). The NIST Definition of Cloud Computing (Draft). *Recommendations of the National Institute of Standards and Technology*, 1-3. Retrieved from http://csrc.nist.gov/groups/SNS/cloud-computing/

Ouanouki, R., April, A., Abran, A., Gomez, A., & Desharnais, J. M. (2017). Toward building RDB to HBase conversion rules. *Journal of Big Data, 4*(1), 10. doi: 10.1186/s40537-017-0071-x. Retrieved from https://doi.org/10.1186/s40537-017-0071-x

Pescholl, A. (2018). Proposal for Economic Analysis of Cloud Computing in the Technical Wholesale. Dans *2018 International Conference on Information Technologies (InfoTech)* (pp. 1-4). doi: 10.1109/InfoTech.2018.8510729

Plummer, D. (2009). Cloud Computing: Enginnering the Requirements for "Everything as a Service". *Gartner Summit Event*. Retrieved from http://reqmon.cis.gsu.edu

Rimal, B., & Choi, E. (2009). A Conceptual Approach for Taxonomical Spectrum of Cloud Computing. *Ubiquitous Information Technologies & Applications, 2009. ICUT '09. Proceedings of the 4th International Conference on Digital Object Identifier*, 1-6.

Rimal, B., Eunmi, C., & Lumb, I. (2009). A Taxonomy and Survey of Cloud Computing Systems. *INC, IMS and IDC, 2009. NCM '09. Fifth International Joint Conference on Digital Object Identifier*, 44-51.

Rocha, L., Vale, F., Cirilo, E., Barbosa, D., & Mourão, F. (2015). A Framework for Migrating Relational Datasets to NoSQL1. *Procedia Computer Science, 51*, 2593-2602. doi: 10.1016. Retrieved from http://www.sciencedirect.com

Serrano, D., Han, D., & Stroulia, E. (2015). From Relations to Multi-dimensional Maps: Towards an SQL-to-HBase Transformation Methodology. Dans *2015 IEEE 8th International Conference on Cloud Computing* (pp. 81-89). doi: 10.1109

Shackelford, R., McGettrick, A., Sloan, R., Topi, H., Davies, G., Kamali, R., . . . Lunt, B. (2006). *Computing Curricula 2005: The Overview Report* présentée à Proceedings of the 37th SIGCSE technical symposium on Computer science education, Houston, Texas, USA. doi: 10.1145/1121341.1121482

Shay, T. (2018). Most popular databases in 2018 according to StackOverflow survey. Retrieved from https://www.eversql.com/most-popular-databases-in-2018-according-to-stackoverflow-survey/

Susilawati, E., & Surendro, K. (2017). A model design of information technology investment for the government sector (Case study: Government institutions in Indonesia). Dans *2017 International Conference on Information Technology Systems and Innovation (ICITSI)* (pp. 32-37). doi: 10.1109/ICITSI.2017.8267914

Stats, I. L. (2020). Real Time Statistics Project. Retrieved from https://internetlivestats.com

Tarandeep, S., & Parvinder, S. (2011). Cloud Computing Databases: Latest Trends and Architectural Concepts. *International Journal of Computer, Electrical, Automation, Control and Information Engineering, 5*, 85-89. Retrieved from http://waset.org

TechTarget. (2017). Relational database management system guide: RDBMS still on top. Retrieved from SearchDataManagement.com

Teekasap, P. (2016). Information technology investment and firm performance. Dans *2016 Management and Innovation Technology International Conference (MITicon)* (pp. MIT-157-MIT-160). doi: 10.1109/MITICON.2016.8025226

Vaquero, L., Rodero-Marino, L., Caceres, J., & Lindner, M. (2008). A break in the clouds: towards a cloud definition. *SIGCOMM Comput. Commun. Rev.,* pp. 50-55. doi: 10.1145/1496091.1496100

Wayner, P. (2008). Tour of Amazon, Google, AppNexus and GoGrid. Retrieved from https://mis-asia.com/resource/internet/internet-search/tour-of-amazon-google-appnexus-and-gogrid/

Weiss, A. (2007). Computing in the clouds. *netWorker, 11*(4), 16-25. doi: 10.1145

White, T. (2009a). *Hadoop: The Definitive Guide*. O'Reilly.

White, T. (2009b). MapReduce for the Cloud - Hadoop. *Yahoo! press*, 500.

Youseff, L., Butrico, M., & Da Silva, D. (2008). Toward a Unified Ontology of Cloud Computing. *Grid Computing Environments Workshop, 2008. GCE '08*, 1-10.

Zelkowitz, M. V., Wallace, D. R., & Binkley, D. W. (2012). Experimental Validation of New Software Technology. Dans *Lecture Notes on Empirical Software Engineering* (pp. 229-263). WORLD SCIENTIFIC. doi: 10.1142/9789812795588_0006. Retrieved from http://www.worldscientific.com/doi/abs/10.1142/9789812795588_0006

Zhao, G., Lin, Q., Li, L., & Li, Z. (2014). Schema Conversion Model of SQL Database to NoSQL. Dans *2014 Ninth International Conference on P2P, Parallel, Grid, Cloud and Internet Computing* (pp. 355-362). doi: 10.1109/3PGCIC.2014.137. Retrieved from http://ieeexplore.ieee.org/document/7024609/