Rules to Migrate a Relational Database to a Column-Oriented NoSQL Database

by

Abraham GOMEZ

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

MONTREAL, SEPTEMBER 21, 2023

ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

BOARD OF EXAMINERS

THIS THESIS HAS BEEN EVALUATED

BY THE FOLLOWING BOARD OF EXAMINERS

Mr. Alain April, Thesis Supervisor
Department of Software Engineering and Information Technology at École de technologie supérieure

Mr. Amin Chaabane, President of the Board of Examiners
Department of System Engineering at École de technologie supérieure

Mr. Alain Abran, Member of the jury
Department of Software Engineering and Information Technology at École de technologie supérieure

Mr. Robert Dupuis, External Evaluator
Département d'informatique, Université du Québec à Montréal

THIS THESIS WAS PRESENTED AND DEFENDED

IN THE PRESENCE OF A BOARD OF EXAMINERS AND PUBLIC

ON JULY 27, 2023

AT ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

Règles à suivre pour migrer une base de données relationnelle vers une base de données NoSQL

Abraham GOMEZ

RÉSUMÉ

La croissance importante des données liées aux applications Web a exposé les limitations des technologies de bases de données relationnelles. En effet, ces technologies rencontrent actuellement plusieurs défis, par exemple, les limites liées à l'augmentation de leur la taille et comment gérer les problèmes liés à un accès rapide dans ce contexte.

Ces problèmes peuvent être résolus soit à l'aide de solution matérielles ou logicielles. Des technologies logicielles de bases de données émergentes liées à l'infonuagique, plus précisément, les bases de données NoSQL promettent d'apporter des solutions à ces défis. Ce type de technologie de bases de données émergente est récemment apparu comme une solution aux limites des technologies de bases de données relationnelles qui font face à la gestion de très grandes quantités de données sur le Web. Plusieurs publications récentes décrivent ces problématiques, entre autres, dans le domaine des réseaux sociaux et de la génomique.

Mais, chaque nouvelle technologie comporte des défis d'utilisation et les ingénieurs logiciel, qui sont plutôt familiers avec les technologies de bases de données relationnelles, hésitent souvent, initialement, à utiliser ces nouvelles technologies par manque de connaissances. Cette recherche a pour objet d'expérimenter et de découvrir un ensemble de règles qui visent à aider à la migration des bases de données relationnelles vers des bases de données NoSQL orientées colonnes pour les ingénieurs logiciels qui font cette migration pour l première fois.

À la suite d'une expérimentation de migration d'une base de données relationnelle existante vers une base de données NoSQL orientée colonne, dans notre cas la technologie HBASE, des ingénieurs logiciels tentent d'effectuer cette migration à l'aide seulement de leur expérience. Cette expérimentation permet d'étudier les défis rencontrés et étapes effectuées par chaque participant et de découvrir sept (7) règles de migration qui ont le potentiel de mieux guider les migrations futures et qui ajoutent à la connaissance des publications récentes en découvrant trois (3) étapes additionnelles qui permettent une meilleure couverture des aspects relationnels lors de la migration.

Cette thèse propose donc ensemble de sept (7) règles de migration qui ont le potentiel d'aider les ingénieurs logiciels qui effectuent cette migration de base de données pour la première fois. La validation de l'utilité de ces règles seront validées dans une autre recherche pourraient les guider pour effectuer les activités de migration des parties problématiques.

Mots-clés : migration de base de données, migration de base de données relationelles (RDB) vers NoSQL, règles de migration de base de données, HBASE.

Rules to Migrate a Relational Database to a Column-Oriented NoSQL Database

Abraham GOMEZ

ABSTRACT

The significant growth of data related to web applications has exposed the limitations of relational database technologies. Indeed, these technologies currently face several challenges, for example, the limits related to the increase in their size and how to manage the problems related to fast access in this context.

These problems can be solved either with the help of hardware or software solutions. The database software technologies related to cloud computing, specifically, No-SQL databases promise to provide solutions to these challenges. This type of database technology was developed as a solution to the limitations of relational database technologies that face the management of very large amounts of data on the web. Several recent publications describe these issues, among others, in the field of social networks and genomics.

But each new technology brings challenges of use and software engineers, who are familiar with relational database technologies, are often initially hesitant to use these new technologies for lack of knowledge. This research aims to experiment and discover a set of rules that aim to help in the migration of relational databases to column-oriented No-SQL databases.

Following an experiment in migrating an existing relational database to a column-oriented No-SQL database, in our case HBASE technology, software engineers attempt to perform this migration using only their experience. This experiment allows to study the steps carried out by each participant and to discover seven (7) migration rules which have the potential to better guide future migrations, and which add to the knowledge of recent publications by discovering three (3) additional steps which allow better coverage of relational aspects during migration.

This thesis therefore proposes a set of seven (7) migration rules that have the potential to help software engineers who are performing this database migration for the first time. The validation of the usefulness of these rules will be validated in another research could guide them to carry out the migration activities of the problematic parts.

Keywords: database migration, RDBMS to No-SQL database migration, database migration rules, HBASE.

TABLE OF CONTENTS

Page

LIST OF TABLES

Page

LIST OF FIGURES

Page

LIST OF ABBREVIATIONS AND ACRONYMS

ACID	Atomicity, Consistency, Isolation, and Durability
API	Application Programming Interface
BASE	Basically Available, Soft-state and Eventual consistency
Bigtable	Gooogle sparsely populated table that can scale to billions of rows and thousands of columns
CAP	Consistency, Availability, and Partition tolerance
Cassandra	A free and open-source, distributed, wide-column store, NoSQL database
CC	Cloud Computing
CCA	Cloud Computing Applications
Cygwin	A POSIX-compatible programming and runtime environment that runs natively on Microsoft Windows
DBA	Database Administrator
DBMS	Database Management System
DDI	Design, Detail for Industrial machinery
DFS	Distributed File System
EB	Exabyte

EC2	AMAZON Elastic Compute Cloud
Freebase	A scalable, graph-shaped database (a No-SQL database)
GB	Gigabyte
GFS	Google File System
Hadoop	An open source framework from Apache that is used to store and process large datasets distributed across a cluster of servers
HBASE	A scalable, distributed column-oriented database that supports structured data storage for large tables (a No-SQL database)
HDFS	Hadoop Distributed File System
HIVEQL	Hive is a SQL-like query engine that runs MapReduce jobs on Hadoop
HPC	High-Performance Computing
IP	Infrastructure Provider
IT	Information Technology
ITE	Information Technology Enterprise
JADE	A single development environment where you define the code and the database
JIRB	Extensible jruby-based shell

NDFS	Nutch Distributed File System
No-SQL	A Database not based on the RDB
OpenQM	MultiValue database (a No-SQL database) developed by Ladybridge Systems
PB	Petabyte
PK	Primary Key
Protobuf	a free and open-source cross-platform data format used to serialize structured data
RAM	Random Access Memory
RDB	Relational Database
RDBMS	Relational Database Management System
RESTful	An architectural style for an application program interface (API)
ROOT	The top-level directory of a file system
SLA	Service Level Agreement
SP	Service Provider
SQL	Structured Quary Language
SU	Service User

TB	Terabyte
XML	Extensible Markup Language
ZB	Zetabyte
Zookeeper	Software that coordinates, communicates, and shares state between the Masters and RegionServers in HBASE

INTRODUCTION

Since the early 2000s, a lot of research has been released concerning cloud computing (CC), a highly publicized technology in information technology and one that is attracting attention from both academia and industry. This is partly because cloud computing promises economies of scale in computing power, energy consumption, cooling, and administration (Erdogmus, 2009). These reasons, and other benefits, suggest that the use of CC will become an integral part of our daily life soon. However, these technologies bring new challenges for software engineers, such as, using existing technologies to manage the growing amount of data that is now collected by CC applications, addressing the rapid growth of data as well as maintaining a good response time. Situations where very large amounts of data (i.e., petabytes or even zettabytes) are processed have been coined as Big Data applications. These recent Big Data applications collect, enrich, store, and analyze very large quantities of data, daily, which has led to the creation of a new research area.

Currently, companies extensively use relational database management system (RDBMS) technology to store and exploit their data. However, (Abadi, 2009) and (Al Mahruqi, 2020) states that accessing petabytes of data efficiently using RDBMS technologies, in the cloud, is becoming more and more challenging. Similarly, Lars reports that once an RDBMS starts to grow, the more complex SQL queries (e.g., the ones that access a very large amount of data and use more than one table) start to show performance degradation. Current solutions to this problem (e.g. sharding, horizontal growth and vertical growth), tend to generate many other problems and side effects (Lars, 2011). Big Data applications have recently popularized the use of No-SQL databases technologies as a solution to these challenges.

Several publications are available to help in understanding the migration from RDBMS to No-SQL technologies (Abdel-Fattah, Mohamed, & Abdelgaber, 2022; Alotaibi & Pardede, 2019; Chongxin, 2010; Kuszera, Peres, & Fabro, 2019; Serrano, Han, & Stroulia, 2015; Singh, 2010) and the approach of providing guidelines for this complex migration process has been

addressed by different researchers (Koto, Kono, & Yamada, 2014; Shuchih Ernest, Kuo-Ming, & Yu-Ching, 2015; Wagner, Hudic, Maksuti, Tauber, & Pallas, 2015). The objective of publishing tested migration step (also called rules or guidelines in this thesis) for software engineers could help to accelerate the acceptance of this new technology. At the time of writing this thesis, there have only been a few research publications addressing specific RDBMS migration aspects such as tables and relationship migration strategies. One such proposal, by (Chongxin, 2010), introduces migration guidelines for the HBase No-SQL technology. Chongxin presents three database migration steps to help software engineers. These rules fall short of covering all the RDB aspects that need to be addressed for a migration since they only focus on a few aspects like the table's "relationships", and ignore other relational aspects like the tables themselves, the table fields, the stored procedures as well as the triggers. One contribution of this research is to clarify all the aspects of an RDB that should be addressed in such a database migration.

Given this observation, Stonebraker states that more experimentation is needed, as a one-size-fits-all migration approach to this problem might not be possible (Stonebraker, 2008). More importantly, not all existing RDB may be good candidates for this type of migration (Stonebraker et al., 2007).

Due to the current lack of migration guidelines and the acknowledged difficulty to carry out this type of database migration (Abdel-Fattah et al., 2022; Aiyer et al., 2012; Chen & Lee, 2017; Chongxin, 2010; Kuszera et al., 2019; Serrano et al., 2015; Singh, 2010), there are a growing number of database migration services appearing on the market (Ippoliti, 2015; Pearl, 1984). These companies offer their services to conduct the migration using a heuristic approach, meaning that they use a method based only on their previous experience. This approach is not guaranteed to be optimal but is often sufficient for the immediate goal of solving a pressing issue. Using a heuristic approach to solve this problem will typically require a considerable investment in time as well as an in-depth knowledge of the targeted No-SQL technology. These specialized services do not reveal their approach so there is still an opportunity to offer migration guidelines to software engineers that are currently No-SQL

neophytes but have experience and good knowledge of RDBMS technologies.

To achieve this objective, this research has chosen to study and experiment RDB to No-SQL migration rules focussing on a specific and popular CC technology, Apache Hadoop, which is a “distributed processing of large data sets across clusters of computers using simple programming models. Hadoop uses HBase, which is one of the most popular Big Data No-SQL technology and is designed to scale up from single servers to thousands of machines, each offering local computation and storage. Rather than rely on hardware to deliver high-availability, the library itself is designed to detect and handle failures at the application layer, delivering a highly-available service on top of a cluster of computers, each of which may be prone to failures.”(Lam, 2011; Lars, 2011; White, 2009).

This thesis is organized as follows. The first chapter presents the research scope including its motivation, the problem definition, the research questions to be addressed and the overall research methodology that was used. The second chapter presents a literature review, focusing on RDBMS and No-SQL technologies, and especially the current state of the art concerning the migration of RDBMS to No-SQL technologies. Also, this chapter presents an overview of the Big Data Apache Hadoop framework, focussing on the HBase No-SQL database used in our experimentation. The goal of the third chapter is to describe the experiment designed to understand how developers currently attempt to do this migration (e.g., based on heuristics) and study how they generally approach the migration steps. Using this knowledge, chapter four identifies, explores, uncovers, and describes a proposed set of seven (7) migration guidelines and conducts a validation of relational aspects coverage and compares the seven guidelines with the 2014-2022 literature on the topic. Finally, the fifth and final chapter presents the conclusion, research contribution and future work.

CHAPTER 1

RESEARCH OVERVIEW

1.1 Motivation

Nowadays, our "*all-connected-everywhere*" society is based on companies that are extensively using data in order to improve their results in several areas such as customer experiences or marketing, or even create new processes with the aim to be more productive, and generate competitive advantage. Figure 1.1 describes the expected Information Increase by 2025 only for the USA.

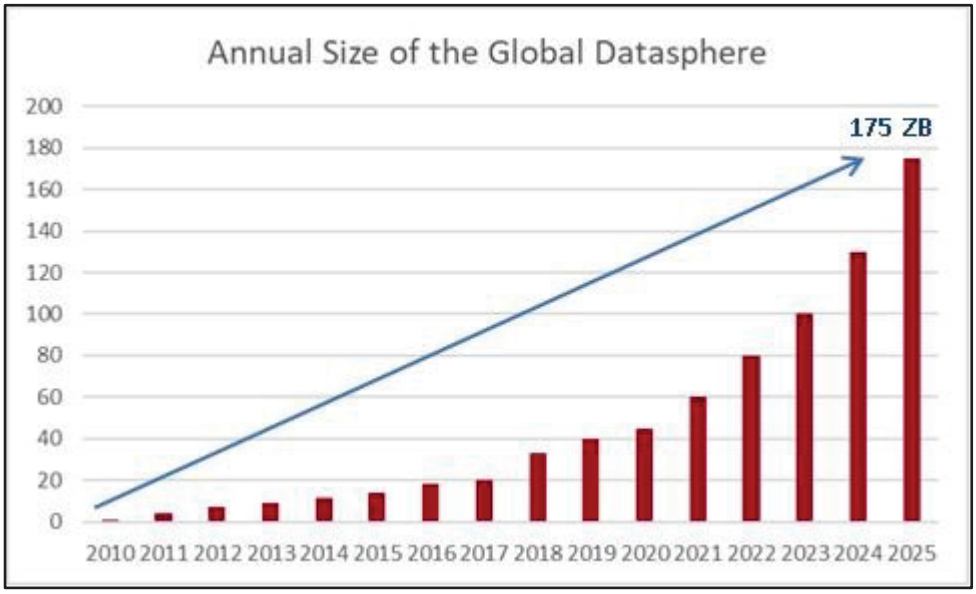


Figure 1.1 The Digital Universe Growth

In the figure the "y-axis" is measured in Zetabytes where 1 Zetabyte (ZB) is equivalent to 1021 bytes. The "x-axis" represent the years. Organizations are also making more and more applications available that use large amounts of data. The figure was taken from (Gantz, Reinsel, & Rydning, 2018) but similar forecasts were found in (Amanor-Boadu, 2022; Sandhu, 2022; Tawfik, Al-Zidi, Alsellami, Al-Hejri, & Nimbhore, 2021).

These applications typically use RDB technology to store and access their data. Applications are progressively migrating to the cloud and are now coined as Cloud Computing Applications (CCA).

The most important observation in Figure 1.1 is that this study predicts that data growth will reach 175 zetabytes of data by 2025. However, the reality is that sizes in the zetabytes have been occuring since 2010 (i.e. as a measure of the digital universe available on the internet). Figure 1.1. shows the information size available today, only for the USA. This study does not take into account the digital information from Europe, China, or Japan. Data available have become so big that a new research field was created and named as "*Big Data*". Future software applications, as well as legacy applications, will need to be adapted to allow the efficient transfer, processing and storage of data on such a large scale.

One immediate consequence of this trend is that legacy applications that currently use RDBMS technologies are gradually migrating to applications in the cloud (as shown by Figure 1.2). According to (Stonebraker, 2008; Stonebraker & Kepner, 2012; Stonebraker et al., 2007), although it is still very common to use the RDBMS technology for CCA, when the data is deployed in database servers, in the cloud, and the information grows beyond terabytes (TB), the RDBMS technology starts to struggle and show its limits. Reseachers are starting to report issues with the growing volume of data and especially problems associated with response time related to "*Big Data*" applications. These authors have established that a real-time centralized cloud database architecture, based on RDB technologies, can currently manage terabytes of data. At some point it becomes hard to keep a good level of service. Also, the large increase in the number of concurrent connected users on CCA applications can cause other problems, for example, transactional difficulties to execute thousands of "commits" that need millions of transactional logs, each one in complete coordination with the others. It is also reported that CCA applications are typically deployed using a shared-nothing architecture, that until now is not fully supported by RDBMS vendors (Abadi, 2009). This raises the issue that there are growing risks in storing transactional data on an untrusted host, as is implied by CC applications. For example, the database could contain data which is considered critical to the core processes. These data could include sensitive information such as private patient information (health domain), customer data (business domain) or credit card numbers (finance

domain). Any increase in potential security breaches or privacy violations would be considered unacceptable. Finally, the administration of these CC applications is more and more complex, as reported by Abadi (Abadi, 2009).

All these issues, as well as the RDBMS limitations, have slowed down the adoption of new technologies and have created resistance in the use of No-SQL technologies on a large scale, as reported by (Abadi, 2009; Cryans, April, & Abran, 2008; Kossmann, Kraska, & Loesing, 2010). Figure 1.2 graphically represents a migration of RDBMS to the cloud (No-SQL).

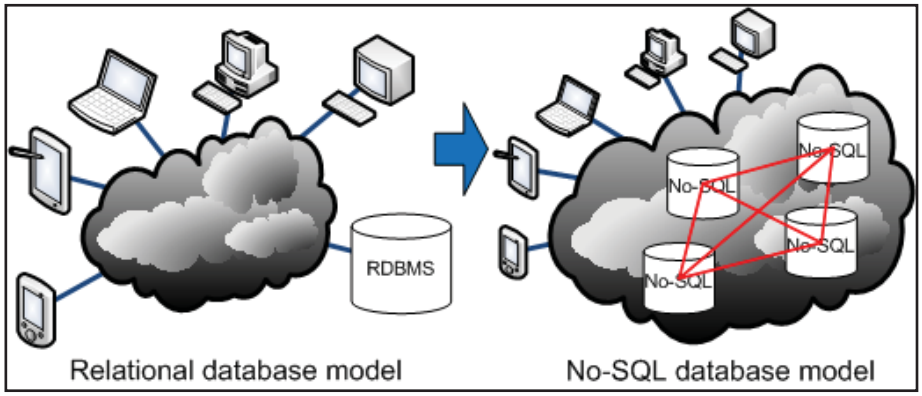


Figure 1.2 When a Database Technology Migration is Needed

If a No-SQL technology seems to be a potential solution to the growing RDBMS issues, how can a software engineer conduct the migration of an existing RDB application to a chosen No-SQL database technology as characterized by Figure 1.2? One of the many roadblocks to a quick acceptance of No-SQL database technologies is the ability to easily migrate and convert existing software systems to this new paradigm. One solution, when software engineers are unfamiliar with this new technology, is access to migration guidelines.

The motivation of this research is to investigate this migration activity so that knowledge that can be observed, validated and generalized to support software engineers who will have to execute this migration process for the first time can be provided.

We know that RDBMS have been around for more than 40 years now. During this time, several contending technologies, such as object databases, have tried to replace them. These predicted the end of the RDBMS era. In fact, no other technology has challenged this dominance;

RDBMS are still the main database paradigm at present and are still taught in all software engineering curriculums today. But this is now changing. Recent database technologies like in-memory databases, Hadoop/No-SQL, columnar databases and streaming databases do not intend to replace RDBMS technologies but they have been chosen in many situations where RDBMS could not meet the challenge. These new technologies can complement RDBMS, where needed, in overcoming specific challenges inherent to large scale CCA in a Big Data context.

1.2 Problem definition

Given that the industry uses mainly relational database management systems (RDBMS) and they are likely to migrate some of their existing large scale CCA from RDBMS to a No-SQL technology in the near future, this creates an opportunity to research the specific problem of how to help software engineers with this first migration tentative. Software engineers are currently RDBMS experts and No-SQL neophytes at this time. Consequently, there is an opportunity to identify migration guidelines that have the potential to help them in their first migration effort from RDBMS to a chosen No-SQL database technology. For this research project, we have chosen the HBase No-SQL technology as the target technology for a migration as it is currently a very popular Big Data columnar No-SQL database and freely available as part of the Apache Hadoop project.

1.3 Research questions

Once the research motivation and problem is clearly identified, it is helpful to further specify what exact research questions/hypothesis this thesis addresses. We have already discussed that in order to help software engineers with a first migration effort they could benefit from using database migration guidelines. Therefore our research aims at uncovering these database migration guidelines. Figure 1.3 shows, graphically, the track that will be covered in this research work. However, to uncover whether the proposed guidelines can be useful, it is recommended to use the same RDBMS application and study two different migration approaches as two separate experimental tracks: the first database migration (TRACK 1), will

be done without the use of guidelines, and the second database migration (TRACK 2) will be done with the use of the guidelines.

This PhD thesis is concerned only with TRACK 1. Another PhD research will undertake TRACK 2 experiment and look at the results to see if a noticeable benefit can be obtained by using the proposed guidelines uncovered in this thesis. Figure 1.3 provides an overview of the scope of this thesis (TRACK 1). TRACK 2 will be discussed on section *"Future Research"* of chapter four.

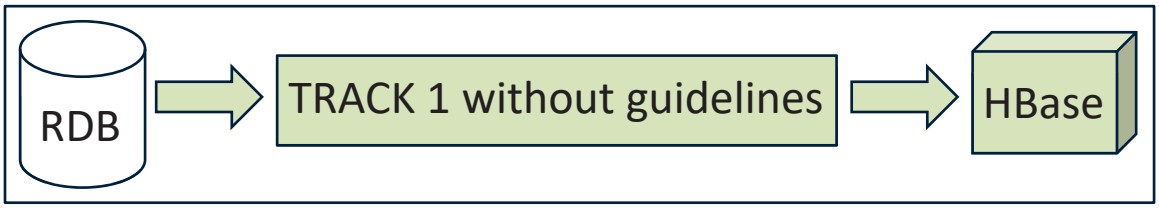


Figure 1.3 The scope of the research migration from RDBMS to No-SQL

In order to conduct a comparison between the two RDBMS application migration approaches and resulting databases ***HBase*** and ***HBase'***, some preliminary experiments need to take place in order to establish a valid baseline for comparison. The goal of this baseline is to allow a valid comparison between the migration process of an RDBMS database to a No-SQL database (HBase) without the use of guidelines (e.g. the heuristic approach) and with the use of guidelines (i.e. the proposed guidelines usefulness). This topic will be covered in detail in chapter 3.

1.4 Methodology

Victor Basili presented a software engineering research framework with the goal of providing a structured way to conduct experimental studies and the evaluation process for software engineering research (Basili, Selby, & Hutchens, 1986). It has subsequently been successfully used in several software engineering research projects, for example (Bourque & Cote, 1991; Desharnais, Pare, Maya, & St-Pierre, 1997). This research uses the modified and improved version of Basili's framework as proposed by (Abran, Laframboise, & Pierre, 2003). It consists

of a research definition phase, a research planning phase, a research operation phase and finally, a results interpretation phase.

1.4.1 Research Definition

The software engineering research definition phase consists of identifying the research problem, and possible solutions that will be explored. The research methodology will be presented along with the activities to develop it (see Table 1.1).

Table 1.1 Basili's framework – Research Definition

I. Definition			
Motivation	**Objective**	**Goal**	**Users**
• To support the migration from RDB to No-SQL DB for software engineers that do this migration for the first time.	• Uncover a set of migration rules that have a good coverage of the relational aspects in the target No-SQL database	• Conduct a migration case study with software engineers that will only use their experience for migrating a relational database to a No-SQL database and study their migration steps and difficulties to uncover migration rules.	• Developers. • DBA's • Researchers. **Domain** • Software Engineering • Software Maintenance.

1.4.2 Research Planning

This phase identifies the research activities as well as the deliverables for the planning phase of this research (see Table 1.2).

Table 1.2 Basili's framework – Research Planning

II. Planning		
Planning goal	**Scope of work**	**Results expected**
• Conduct bibliographic review (state of the art) of RDBMS technology, RDB aspects, No-SQL database technology, database migration theory and example of recent No-SQL migration attempts.	• Identify and list key features and technologies involved in the migration process from relational to No-SQL databases: o Overview of RDBMS technology; o Overview of No-SQL database technology; o Database migration theory; o Example of recent No-SQL migration attempts.	• Chapter 1 of thesis (DGA 1005); • DGA 1031 report; • Research questions; • Proposed papers: o **Journal Paper:** 2017, "Toward building RDB to HBase conversion rules", Journal of Big Data, Springer Open. o **Conference Paper:** 2015, "Experimental Validation as Support in the Migration from SQL Databases to NoSQL Databases", IARA Conference, Nice, France. o **Journal Paper:** 2014, "Building an Experiment Baseline in Migration Process from SQL Databases to Column Oriented No-SQL Databases". Journal of Information Technology & Software Engineering.
• Install research environment.	• Install Hadoop	• Laboratory environment installation.

1.4.3 Research Operation

The operation phase prepares the components that design a solution to answer the research question (see Table 1.3).

1.4.4 Interpretation

The interpretation phase consists of the interpretation of the experiment results to obtain conclusions, assess the potential of the proposed solution for the industry, and finally identify future research work. Table 1.4 presents the interpretation phase of this research.

Table 1.3 Basili's framework – Research Operation

III. Operation (rules, prototypes and experimentations)		
Research steps	**Execution (process)**	**Data Analysis (Output)**
Step 1: Chapter of thesis as a summary of previous knowledge.	• Propose RDB aspect coverage criteria to identify the proposed guidelines.	• Journal paper.
Step 2: Create a baseline that allows a valid comparison between the two methods that will be analyzed in the research program (TRACK 1 and TRACK 2). **Step 3:** Investigate the guidelines set for the migration from RDB to No-SQL databases.	• Research about schema, migration and generalize the set of guidelines.	• Chapter of thesis on migration guidelines. • Conference publication on RDB migration.
Step 4: Investigate the set of guidelines for the migration of different relational database (RDB) aspects of a relational database that should be found in the resulting No-SQL database after the migration.	• Research the different relational database (RDB) aspects of a database and how they can be migrated.	• Chapter of thesis on SQL migration guidelines. • Conference publication on RDB migration, including the relational aspects of a database.
Step 5: Experiment to validate the proposed solution.	• Study a database schema. • Study database queries. • Use guidelines to migrate the schema and queries to HBase. • Document the guide that provides migration guidelines to support individuals who want to convert CC DB.	• Chapter on demonstration of applying migration guidelines to a real case. • Conference publication: applying migration guidelines to a real case.
Step 6: Prepare and execute case study.	• Define the experimental method and steps for the case study. • Prepare data collection. • Prepare test environment. • Choose candidates for case study • Sign ÉTS ethics documentation for the experiment. • Conduct the experiment (case study). • Collect data on experiment results.	• Chapter on experiment description and preparation. • Journal publication of total thesis.

Table 1.4 Basili's framework – Interpretation Phase

IV. Interpretation (results and future work)		
Interpretation Context	**Extrapolation**	**Impact**
• This limited case study has shown good resulting RDB aspect coverage if the 7 rules are used.	• Rules must be validated by TRACK 2 research to extrapolate their potential use. • RDB aspects in migrated No-SQL show good promise.	• Conduct more experiments using the migration rules for validation and comparison with the baseline. • Conduct more surveys once the migration tasks have been completed. • Add and improve the uncovered set of migration rules.

1.4.5 Conclusion

This chapter has presented an introduction of the research topic, starting with its motivation as well as the problem it intends to solve. The research methodology was then presented which includes the goal of the research. The next chapter presents the literature review.

CHAPTER 2

LITERATURE REVIEW

This chapter presents the theoretical concepts concerning the technologies used in this research. Sources for this review include: books, journal papers, conference publications and articles. This chapter is divided into five sections. The first section introduces the concepts and technology of relational database management systems (RDBMS) including their current limitations. In the second section, an overview of the No-SQL database technology is presented along with its advantages and limitations, focusing on the HBase technology used in the experiment for this research. The third section presents the state of the art in database migration theory followed by section four, which highlights different No-SQL migration approaches and the issues faced with this type of migration. Finally, section five presents a conclusion summarizing the concepts that are at the basis of this research.

2.1 Concepts and RDBMS Technology

In this section, the definiton, characteristics, main aspects, advantages and limitations of current RDBMS are presented.

2.1.1 RDBMS Definition

A relational dabatase management system (RDBMS) is one of many types of database management systems (DBMS) and, by far, the most widely used in the industry today. It is based on the “relational model” theory developed by E. F. Codd (Codd, 1970). The acceptance of RDBMS technology can be attributed to several key factors such as: the maturity of the commercial products available, the simplicity of using the relational model, and the flexibility of its query language named the structured query language (SQL). E.F. Codd invented the concept that “relational” really means conformity with twelve rules, designed to define what is required from such a database management system (Codd, 1971a, 1971b). Below is a quick overview of Codd’s rules:

0. The RDBMS should use its relational facilities exclusively to manage the database;

1. *The information rule*. All information in the database is represented in one way: values in tables with rows and columns;

2. *The guaranteed access rule*. The RDBMS preserves the information as a combination of three key aspects: tables, primary keys and column names. All data must be accessible using the *primary key* aspect;

3. The RDBMS should support *null value*, which is a representation of *missing information* or *inapplicable information* and is totally different from all regular values, or numbers, including zero values;

4. The structure description of the RDBMS should be stored in a catalog: the data dictionary. The information on this catalog should be accessed only by authorized users using the appropriate relational query language;

5. The only way to access an RDBMS should be using a relational query language that allows data definition, data manipulation, and transaction management operations. The direct access to the RDBMS, without this language, is considered a transgression;

6. The RDBMS should allow the update of any view from scratch;

7. The following high-level operations should be allowed by the RDBMS: insert, update and delete;

8. One of the main properties of the relational system is the physical data independence, which means that how the data is stored in an RDBMS, e.g. arrays or linked lists, must be independent of the applications that access this data;

9. Similarly, there is logical data independence, which means that changes at the logical level, e.g. merge two tables or split one table into two different tables, should have no impact or change the user application. Logical data independence is more difficult to achieve than physical data independence, indeed, logical data independence is one of the most difficult rules to apply;

10. The integrity constraints should be specified separately from application programs and it

should be stored in the catalog. The user can change constraints independently without the need of any change in the application;

11. *The distribution independence rule*. The end-user should not be able to see that the data is distributed over various locations. This rule is strongly related to rule eight;

12. If the system provides a low-level interface, this interface should not be used to modify the data system bypassing security and integrity constraints.

In the early days of RDBMS, most of the implementations did not conform with all of Codd's recommended rules. In fact, most RDBMS commercial offerings offered a database model that satisfied, at a minimum, the following concepts:

- Present the information to the user in a tabular form, which means, a set of rows and columns;
- Provide relational operators to manipulate the data in tabular form.

These two concepts, among others, will be presented in next subsection.

2.1.2 RDBMS concepts

In this section, key fundamental concepts of RDBMS are explained using the approach taken by (Elmasri & Navathe, 2016). Elmasri uses a conceptual approach, which means that he teaches relational concepts using a top-down approach and presents RDBMS concepts in simple terms, facilitating their understanding (Giddens, 2017; Pierce, 2012). To help the reader follow these explanations, we use a simple example of a *Hospital-Doctor* database presented in Singh (Singh, 2010). The data used and displayed here are used purely for understaning the concepts; they contain information about hospitals, doctors, departments and cities:

- Table (Relation): According to Elmasri, a table is a physical way to organize information (an RDBMS concept) and a relation is a logical way to organize information (a relational database theory concept). However, in this subsection both concepts will be treated as synonyms. A table consists of rows (also known as tuples or records) and columns (attributes). Tables are used to store all the information about the objects. A specific table

should contain data of only one kind of objects (Elmasri & Navathe, 2016). In Figure 2.1, the *Hospital-Doctor* database shows two tables: 1) Hospital and 2) Departments. Each table contains information related to either hospital or department accordingly.

Hospital

ID	NAME
01	Jewish General Hospital
02	St. Mary's Hospital
03	Shriners Hospital for Children of Canada
04	Mount-Sinai Hospital

Departments

ID	NAME
01	Financial
02	Intensive care unit
03	Medical records
04	Pharmacy

Figure 2.1 The attributes, tuples and fields of a relation Hospital

- Row (Tuple, Record): The rows are a component of the tables and represent a collection of related values. Rows contain all the information about this object. In Figure 2.1, the second row, "*02 St. Mary's Hospital*", shows all the information about the above-mentioned hospital. So, there is only information of one hospital in one row in the Hospital table. The different synonyms for rows depend on the model used; whereas row is an RDBMS concept, the tuple is a relational model concept. The term "record" is an outdated concept from the early days of the relational theory (see Figure 2.2);

- Column (Attribute): Another component of the tables are the columns. They contain a particular type of information, one for each row of the table. A column has a name that describes the data of the column. In the Hospital table there are columns, e.g. id and name. A column is an RDBMS concept and an attribute is a relational model concept (see Figure 2.2);

- Field: According to Elmasri, a field is part of a row and it stores a single piece of information for the given row. From a graphical perspective, a field is the actual value that can be found at the intersection of the row (tuple, record) and the column (attribute). In Figure 2.2, the Hospital table database shows the field "St. Mary's Hospital" in the intersection of row "02 St. Mary's Hospital" and column (attribute) "Name" (Elmasri & Navathe, 2016).

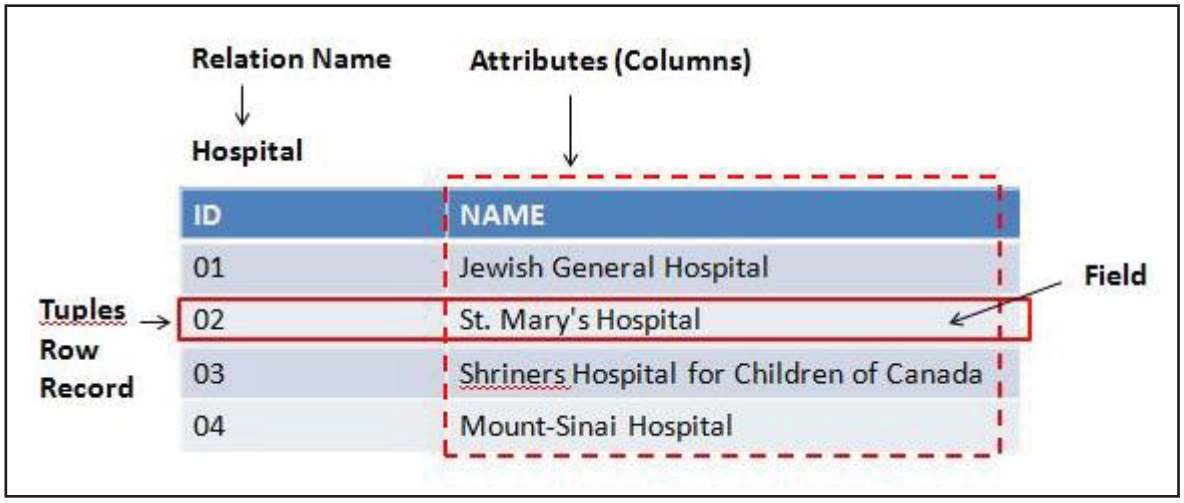


Figure 2.2 The field is represented by the intersection of row and column

- Constraints: According to Elmasri and Tow, there are specific rules used to force or restrain the type of information that will be saved in a table. This ensures the accuracy and reliability of the data in the RDBMS. The moment the constraints are applied there are two parts: the data involved and the constraint itself. If there is any violation between the data involved and the rule specified by constraint, the whole process is aborted by the constraint. The constraints could be applied at two levels: tables and columns. Table level constraints affect the data that could be saved in the whole table and column level constraints affect the data that could be saved in one specific column. The constraints are created, no matter the type, either when the table is created or after the table is created (Elmasri & Navathe, 2016; Tow, 2003). The common kinds of constraints are:

  - *Not Null Constraint*: Ensures that a column cannot have null values;
  - *Default Constraint*: Specifies a default value for a column when none is provided;
  - *Unique Constraint*: Ensures that all values in a column are different;
  - *Primary Key Constraint*: Is a combination of not null and unique constraints. The primary key is the column (or set of columns) whose values uniquely identify the row. All primary key fields have a different value in a specific table. A table should have at least one primary key. The primary key of the table Hospital is the Id. Two Hospitals cannot have the same Id (see Figure 2.2);

- *Foreign Key Constraint*: Uniquely identifies a row in any other table. The value(s) in specified column(s) must reference an existing row in another table (using either it's primary key in the reference table or some other unique constraint);
- *Check Constraint*: This constraint ensures that all values in a column satisfy certain conditions;
- *Index Constraint*: Used to create and retrieve data from the database very quickly.

2.1.3 ACID Properties

When a transaction processing system creates a transaction, that transaction should ensure certain characteristics. These characteristics are listed as ACID properties. The acronym stands for Atomicity, Consistency, Isolation and Durability. The software engineers that create a transaction must be assured that these properties are in place and provided for automatically by the RDBMS. If it is not the case, they will need to manage each property themselves in their source code. According to Elmasri, the ACID properties constitute an important concept for modern databases and, for RDBMS technologies, they allow for the safe sharing of data. Without them, everyday activities, such as using computer systems to buy products, would be difficult and the potential for data inaccuracy would be constant (Elmasri & Navathe, 2016):

- Atomicity: The atomicity property indicates an all-or-nothing unit of work, succeeding if and only if all contained operations succeed. The whole operation is either fully completed, or has not begun at all. Any updates that a transaction might effect on a system are completed in their entirety. If for any reason an error occurs and the transaction is unable to complete all its steps, the system is returned to the state it was in before the transaction was begun. This operation is called a rollback;
- Consistency: A transaction enforces consistency in the state of the system by ensuring at both the begining and at the end of the transaction that the state is valid, which means that all changes to the system have been properly made and the transaction has been successfully completed. If an error occurs in the transaction, then any changes already made will be automatically rolled back. This will return the system to its state before the transaction was

begun. Since the system was in a consistent state at that time, it will once again be in a consistent state;

- Isolation: When a transaction runs in isolation, it appears to be the only action that the system is carrying out at the time. If there are two transactions performing the same function and they are running at the same time, transaction isolation will ensure that each transaction "thinks" it has exclusive use of the system;
- Durability: A transaction is durable if, once it has been successfully completed, all the changes it made to the system are permanent. There are safeguards that will prevent the loss of information, even in the case of system failure.

2.1.4 CAP Theorem

Sometimes the RDBMS is requested to successfully complete its read and write processing in a specific time frame. When this becomes difficult to achieve because of lack of resources, e.g. computing power, storage, memory, etc, then, it requires an adjustment. A distributed RDBMS system, which is a collection of interconnected nodes that share data, is a potential solution to this problem. Ideally, a distributed system should achieve three desirable characteristics:

- Consistency: It means, a read in the system should return the most recent write for a given client;
- Availability: In the system, if there is neither error nor timeout, a non-failing node should return a response within a reasonable amount of time;
- Partition Tolerance: The system will continue to work even if there is a no communication between two nodes.

The CAP theorem states that no distributed system can achieve all three characteristics listed above at the same time. Indeed, one of them should be sacrificed and requires a tradeoff (Brewer, 2000b; Gilbert & Lynch, 2002).

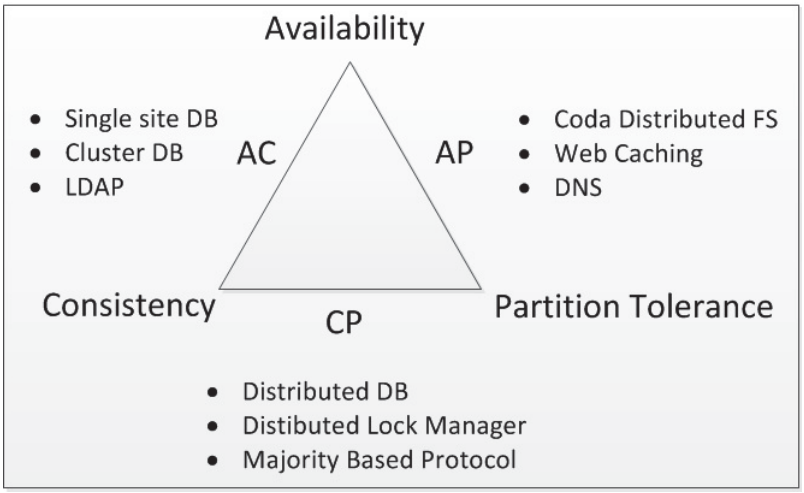


Figure 2.3 CAP Theorem
Taken from (Brewer, 2000a)

As a consequence of the CAP theorem, a distributed system can only achieve at most two of these properties, requiring a design trade-off, e.g., it is impossible to guarantee both availability and consistency in a system that was designed to use partitions. Another example is a distributed database with ACID properties (presented earlier in section 2.1.3) which provides a stronger consistency service, and cannot always provide availability when the workload of the system is high. According to Kong, each property pair refers to three subcategory levels (Kong et al., 2015), (see Figure 2.3):

- AC (Availability – Consistency): This is the subcategory level used for the regular RDBMS; it implements ACID properties and availability without a problem;
- CP (Consistency – Partition Tolerance): According to the CAP theorem, this is the subcategory level where the distributed database favors implementing the ACID properties rather than availability;
- AP (Availability – Partition Tolerance): The last subcategory level, which cannot implement the ACID properties, provides the BASE properties, and a weaker degree of reliability for transactions;

The consistency model known as BASE (Basically Available, Soft-state and Eventual consistency), is a model that does not provide strong consistency. Instead, for this model, it is enough to offer to be eventually in a consistent state. Systems with a BASE model may not be

suitable for all domains and applications, but are a flexible alternative to the traditional RDBMS (Kong et al., 2015). For example, in order to achieve higher performance and availability, many No-SQL DMS have adopted the BASE consistency model approach such as Bigtable (Chang et al., 2008), PNUTS (Cooper et al., 2008) and Cassandra (Lakshman & Malik, 2010).

In a distributed system, which uses the partition tolerance property, the decision between consistency and availability is a design trade-off and the software engineer could choose what to do in case a network partition fails (for example, a network outage). Designing a distributed system can become a complex task when determining which type of different trade-off is best. This requires a good comprehension of the goal of the software application and its domain constraints. Failing to decide the right property to privilege could affect the performance and success of the software development project.

2.1.5 Database Scalability

Before continuing with the current RDBMS limitations and the No-SQL database theory, it would be appropriate to address some essential concepts, thus enabling us to better comprehend the subjects covered in the following sections.

When it is said that a database has the characteristic of scalability, the given database has the ability to provide a reasonable performance as a response to growing technical demands, also known as “increased loads”. Examples of such increased loads are: rising traffic, increased data volume, or increased need for power computing to process their data. Also, a scalable database system reduces the need of having to redesign the database schema under such situations. Quite simply, adding more resources is the only way to handle the increased load on an application, but the big question is how to scale. What is the best way to achieve it? There are two methods of adding more resources for a particular application: vertical scaling and horizontal scaling (Kleppmann, 2017; Özsu & Valduriez, 2011).

- Vertical Scaling: Also known as “scaling up”, vertical scaling refers to adding more resources to a single unit and in that way expand its ability to handle increasing load. There are two approaches to achieve vertical scaling: at hardware level or at software level. The

hardware option includes adding processing power and memory to the physical machine running the server, or adding parallelizing hardware or optimizing a certain number of running processes. The software option includes optimizing algorithms and application code.

- Horizontal Scaling: Also known as “scaling out”, horizontal scaling refers to resource increment by the addition of units with the same performance of the current server, or even lower performance. Having multiple servers allows for the possibility of ensuring the response even if some servers go down, thus avoiding the “single point of failure” problem and increasing the availability of the RDBMS.
    - Load Balancing: The main issue with horizontal scaling is related to the multiple requests that arrive and the best way to decide which application/server or processing unit would respond to which request. The solution can come from a number of methods, which could be grouped under the technique of load balancing. A load balancer accepts requests from several users and then directs them to the right server. Here, the “right” server is decided by certain criteria/algorithms that depends on the load balancing strategy.

2.1.6 Distributed Computing and Shared Nothing Approach

Another important concept that needs to be discussed is distributed computing, which will be explained since the No-SQL databases are based on a specific architecture of distributed computing called *shared nothing*. According to Stonebraker, the distributed architecture is a model in which processes are divided on several networked computers, and they communicate and coordinate their actions using messages. All the processes interact with each other in order to achieve a common goal (Stonebraker, 1986). Currently, virtually all large computer-based systems are now distributed systems where the information is spread and processed over several computers rather than by using a single point of processing. This approach presents several advantages listed below (Kleppmann, 2017):

- The resources (hardware and software) are shared;
- The openness, since it is possible to use hardware and software from different vendors;

- The possibility of concurrent processing, since there are “n” processors available over the network;
- Scalability, considering the computing power could be increased by adding new resources;
- The fault tolerance, seeing that there is no single point of processing.

Despite these positive advantages, it is noted that some challenges exist, such as:

- The complexity, considering the distributed systems are more complex to handle than centralised systems;
- The security, all the information is spread to more than one place, so it is more susceptible to external attack;
- The manageability, considering more effort is required for system management.

In relation to what has been mentioned above, Stonebraker says that a distributed computing system could be organized with different kinds of architectures, e.g., shared memory, shared storage and shared nothing (Stonebraker, 1986).

In the shared nothing architecture, there is no central processor that controls the entire network. Indeed, each node is independent and self-sufficient, and they do not share memory (as may be possible in a shared memory architecture) or disk storage (possible in shared storage architecture). Distributed computing was designed for taking the information and spreading it into several computing units, i.e., machines or nodes. Normally, the real problems fall into two groups (Stonebraker, 2008):

I. Those where each node can work on its piece of the problem without having to exchange data with other nodes, except for some process of aggregation/combination at the end;

II. Those where each node finds that it must talk to others during the information processing.

The selection of the appropriate architecture for the distributed computing system is a tough issue. In any case, it depends on the “trade-offs” between processing cost and communication costs (e.g., bandwidth). If the cost of communication between each node is high, maybe minimizing any such communication could be a good idea; this is the approach behind the shared nothing architecture. The problems of type I can be easily solved with shared nothing

architecture, considering no communication between nodes is needed during the information processing.

On the other hand, if the node communication is cheap, e.g., the nodes are located in the same physical machine, and the main memory is equally cheaply accessible, then another architecture, different than share nothing, could be selected. In this scenario the problems of type II are easily solved.

2.1.7 Current RDBMS Limitations

According to White, for a majority of typical small to medium applications, there is no substitute, in terms of ease of use, flexibility, maturity and richness of features, than an RDBMS such as Oracle, MySQL or PostgreSQL to name just a few popular technologies. However, if an RDBMS needs to be scaled up in terms of dataset size, read or write concurrency, or both, RDBMS technologies have their limits. The scaling of an RDBMS usually involves breaking Codd's rules, meaning a loosening of the ACID properties, moving away from conventional database administration (DBA) wisdom and, along the way, bypassing some automated and desirable properties that make relational databases so convenient in the first place (White, 2009).

There are a number of challenges that an RDBMS will face in its attempt to scale:

- Migration from a local workstation to a shared, remotely hosted RDBMS with a well-defined schema;
- 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 proprieties;
- If the service continues to gain in popularity, and too many writes are hitting the database, a vertical scale 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 too many joins, and data denormalization has to be performed to reduce them;

- Increasing popularity can swamp the server, which 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. But the next step could be to remove indices altogether.

Scaling horizontally is an option. It typically involves an attempt to create some type of partitioning for the largest tables, or to look at some of the commercial solutions that provide multiple master capabilities. One example of this option is presented by YouTube. YouTube first used a MySQL master-slave replication approach to try to solve this issue, but eventually arrived at a point where the writes were using all the capacity of the slaves. Like many other organizations facing this growth problem, they tried partitioning their tables into shards, so that the sets of machines hosting the various databases were optimized for their tasks (Cryans et al., 2008).

According to White, the reality is that countless applications, businesses and websites have successfully achieved scalable, fault-tolerant and distributed data systems built on top of RDBMS, and have implemented many of the suggestions mentioned above. However, making such changes could result in a system that is no longer a true RDBMS, as compromises will have been made and complexities added at the expense of features, maintainability and convenience. Any form of slave replication or external caching reduces consistency in what are now denormalized data. The inefficiency of joins and secondary indices means that almost all queries become primary lookup keys. A multi-writer setup likely means no real joins at all, and distributed transactions are a nightmare. The result, in the near future, would be an incredibly complex network topology to manage, with an entirely separate cluster for caching. Moreover, in a short time, there would be 10 times the data and 10 times the load (White, 2009).

2.2 Overview of No-SQL Database Technology

No-SQL is the term applied to database management systems that do not use the relational concepts found in RDBMS. These databases may not require fixed table schemas, they usually avoid join operations and typically scale horizontally. Lam and Venner refer to these databases as structured storage, a term that would include classic relational databases as a subset (Lam, 2011; Venner, 2009). The term No-SQL is the name for these new database management systems that have the following characteristics:

- Open source licensing model;
- Initially did not have an SQL interface;
- Most importantly, distanced itself from the relational model altogether.

Also, this term is used to refer to a growing number of non-relational, distributed databases where often no attempt is made to provide the ACID properties. Today, the most common interpretation of No-SQL is that it is non-relational, although No-SQL should not be taken to mean anti-RDBMS. Rather, it is considered to complement the RDBMS. No-SQL implementations can be categorized by their manner of implementation:

- A consistent key-value store, such as Cassandra (a Hadoop project) or Dynamo;
- A hierarchical key-value store, like GT.M or GlobalsDB;
- Hosted services, like Freebase;
- A key-value cache using RAM, such as Oracle Coherence or Tuple space;
- A key-value store on disk, like BigTable;
- Multi-value databases, such as OpenQM;
- An object database, like JADE;
- An ordered key-value store, such as IBM Informix C-ISAM;
- A table-based store, like BigTable or HBase.

As we will see in this research, a good example of a No-SQL model is the Apache Hadoop HBase technology.

2.2.1 No-SQL History and State of the Art

Carlo Strozzi was the first to coin the term No-SQL, when, in 1998, he used it to name a relational database that did not have an SQL interface. He stresses that the No-SQL model is a complete departure from the relational model, and therefore thinks that a more appropriate name would have been “Non-relational SQL”, as the database content is not represented by mathematical relations. The next major milestone for the No-SQL database technology was reached in 2004, when Google published their MapReduce algorithm (Chang et al., 2008; Ghemawat, Gobioff, & Leung, 2003) for using massive numbers of low-cost CPUs.

Another milestone in the evolution of No-SQL was achieved in 2006, when Google published the BigTable paper (Chang et al., 2008), which is its No-SQL implementation for Google’s large data management system. Eric Evans, a Rackspace employee, reintroduced the term and popularized “No-SQL” in early 2009, when Johan Oskarsson of Last.fm wanted to organize an event to discuss open source distributed databases. The choice of name was an attempt to label the growing number of non-relational distributed data storage systems that were often not designed to provide the ACID properties (atomicity, consistency, isolation and durability), which are the key attributes of classic relational database systems. Also in 2009, the first No-SQL conference was held in Atlanta (Georgia-Tech-Research-Institute, 2009). At this conference, the most common interpretation of No-SQL was that this model is non-relational, and it was established that No-SQL is not meant to imply an anti-RDBMS. Rather it was intended to emphasize the advantages of Key-Value Stores, Document Databases and Graph Databases. Figure 2.4 shows the main milestones in the history of No-SQL databases.

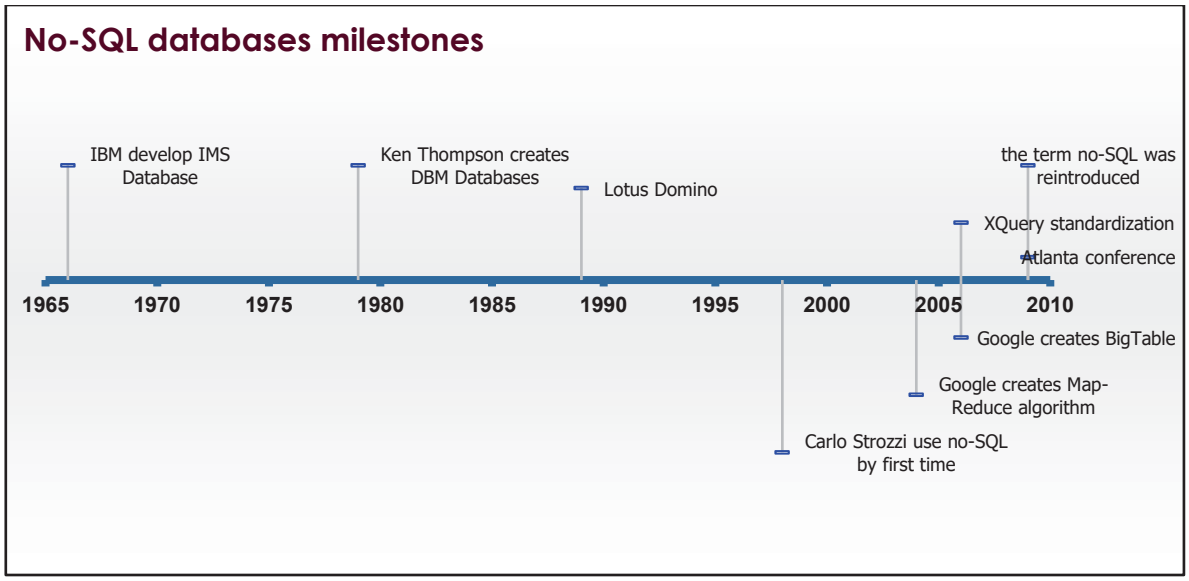


Figure 2.4 No-SQL database milestones

2.2.2 Advantages of No-SQL

Summarizing the various viewpoints using Cryans et al., Microsoft documentation, and White's publication, the new No-SQL model offers the following advantages (Cryans et al., 2008; Microsoft, 2011; White, 2009):

- Elastic scaling: For years, database administrators have relied on scale-up, buying bigger servers as the database load increases for instance, rather than scale-out through a distributed database across multiple hosts. However, as transaction rates and availability requirements increase, and as databases move into clouds or virtualized environments, the economic advantages of scaling out on commodity hardware will become irresistible. Unfortunately, the RDBMS might not scale out easily on commodity clusters, but the new breed of No-SQL databases is designed to expand transparently to take advantage of the new scheme;
- Big data: Just as transaction rates have grown, the volumes of data that are being stored have also increased massively. The RDBMS capacity has been growing to match these increases, but, as with transaction rates, the constraints of data volumes that can, for practical purposes, be managed by a single RDBMS are becoming intolerable for some

enterprises. Today, the huge volumes of data that can be handled by No-SQL systems like Hadoop outstrip what can be handled by the largest RDBMS;

- Decrease management: Compared with RDBMS, No-SQL databases are generally designed to require less management from the ground up: automatic repair, data distribution and simpler data models lead to lower administration and tuning requirements, in theory. In practice, someone will always be accountable for the performance and availability of any mission-critical data store;
- Economics: No-SQL databases typically use clusters of cheap commodity servers to manage the exploding data and transaction volumes, while RDBMS tend to rely on expensive proprietary servers and storage systems. The result is that the cost per gigabyte or transaction per second for No-SQL models can be many times less than the cost when using an RDBMS, making it possible to store and process more data at a much lower price point;
- Flexible data models: Change management is a big headache for a large production RDBMS. Even minor changes to the data model of an RDBMS have to be carefully managed and may necessitate downtime or reduced service levels. No-SQL databases have far more relaxed data model restrictions. No-SQL key-value stores and document databases allow the application to store virtually any structure in a data element. Even the more rigidly defined BigTable-based No-SQL databases, such as HBase and Cassandra, typically allow new columns to be created without too much effort.

2.2.3 Challenges of No-SQL

The promise of No-SQL databases has generated a great deal of enthusiasm, but there are many obstacles to overcome before they will appeal to mainstream enterprises. Some of the major challenges are:

- Maturity: For most companies, the very long period of time RDBMS have been operating for them means they have achieved a state of maturity, and that is reassuring for these companies. These systems are stable and highly functional. In comparison, most No-SQL

alternatives are currently offered in pre-production versions, with many key features yet to be implemented. Living on the technological leading edge is an exciting prospect for many developers, but enterprises should approach this state with extreme caution;

- Support: Enterprises want the reassurance that if a critical system fails they will be able to obtain timely and competent support. All RDBMS vendors go to great lengths to provide a high level of enterprise support. In contrast, most No-SQL systems are open source projects, and although there are usually one or more firms offering support for every No-SQL database, these companies often are small start-ups without global reach;
- Data analysis: No-SQL databases offer few facilities for ad hoc query and analysis. Even a simple query requires significant programming expertise, and commonly used business intelligence tools do not provide No-SQL connectivity. Some relief is provided by solutions like Hive or Pig, which can provide easier access to data held in Hadoop clusters, and perhaps eventually in other No-SQL databases;
- Management: The design goal for No-SQL may be to provide a zero administration solution, but the current reality falls well short of that goal. No-SQL today requires a great deal of skill to install and significant effort to maintain;
- Expertise: Almost every No-SQL developer is in learning mode. This situation will be addressed naturally over time, but for now, it is far easier to find experienced RDBMS programmers and administrators than a No-SQL expert.

2.2.4 The Hadoop Project

In the previous section, the cloud computing paradigm and its components were presented. Cloud computing uses its various service models to offer applications, application development and/or storage through its multiple technology approach. The Hadoop project is one of these cloud computing technologies. It offers a set of open source tools and libraries for implementing different kinds of services in each cloud computing service model. Ghemawat maintains that the Hadoop project has been widely adopted because it is an open source version of Google's technology (Ghemawat et al., 2003). Hadoop uses its various libraries to permit

the storage of vast amounts of information, and in fact it enables large data tables to be managed, as well as having its own data warehouse system. In the next sections, we analyze the nature of the Hadoop project, its origins, components and subprojects. In addition, we present some successful case studies and current research directions.

2.2.5 Definition

Hadoop is an open source, Java-based programming framework that supports the processing of large data sets in a distributed computing environment. It is sponsoring by the Apache Software Foundation. According Dean and Ghemawat, Hadoop was inspired by Google's MapReduce programming model as well as the Google File system, in which an application is broken down into numerous small parts. Any of these parts, also called fragments or blocks, can be run on any node in the cluster. Hadoop makes it possible to run applications on systems with thousands of nodes involving thousands of terabytes (Dean & Ghemawat, 2008; Ghemawat et al., 2003).

Lam lists the following as the key characteristics of the Hadoop project (Lam, 2011):

- Accessible: Hadoop runs on large utility computing clusters, or as PaaS or IaaS offered by providers like Amazon with their Elastic Compute Cloud (EC2).
- Robust: Because Hadoop is intended to run on utility computing systems, its architecture was built on the assumption that frequent hardware malfunctions would occur, and that it could handle most of these failures gracefully. According to Attebury, the Hadoop Distributed File System (HDFS) facilitates rapid data transfer rates among nodes and allows the system to continue operating uninterrupted in case of a node failure. The risk of catastrophic system failure is low, even if a significant number of nodes become inoperative (Attebury et al., 2009).
- Scalable: Hadoop scales linearly to handle larger amounts of data by adding more nodes to the cluster.
- Simple: Hadoop allows users to quickly write efficient parallel code.

In summary, according to Lam, Hadoop is an open source framework for writing and running

distributed applications that process large amounts of data (Lam, 2011). By using distributed storage and transferring code instead of data, Hadoop avoids the costly transmission step when working with large datasets.

2.2.6 History Overview of the Hadoop Project

According to White, the first version of the Hadoop framework was released by Doug Cutting, the creator of Apache Lucene, in collaboration with Mike Cafarella in 2002. Hadoop originated in the Apache project Nutch, which is an open source Web search engine, and was designed as part of the Lucene project (White, 2009).

This kind of search engine requires a complex system to enable crawling and indexing of websites, which means that the system would need to support an index of nearly a billion pages. The Hadoop project budget was estimated at around $500,000 in hardware, with a monthly running cost of $30,000. The creators of Hadoop realized that their architecture at the time would not scale to the billions of pages on the Web.

In 2003, Google Inc. presented its distributed file system (DFS), called the Google File System (GFS) (Ghemawat et al., 2003). Based on that system, Cutting and Cafarella wrote an open source implementation, the Nutch Distributed File System (NDFS), which was launched in 2004.

In 2004 as well, Google published a paper introducing the MapReduce programming model. Early in 2005, the Nutch developers worked on a MapReduce implementation for Nutch. White explains that the NDFS and the MapReduce implementation in Nutch were applied beyond the search domain, and, in February 2006, they converted the Nutch project into an independent subproject of Lucene, called Hadoop. In January 2008, Hadoop was made a top-level project at Apache, called the Apache Hadoop project, confirming its success. At the same time, the NDFS was relabelled the HDFS, to incorporate the Hadoop name (White, 2009).

There have been many milestones in the race to develop MapReduce implementations. For instance, in April 2008, it was announced that Apache Hadoop had the fastest implementations, because they could sort a terabyte (TB) of data in 209 seconds (3.8 minutes) on a 910-node

cluster. In November 2008, Google reported that its MapReduce implementation sorted 1 TB in 68 seconds. In May 2009, the Yahoo! team used Hadoop to sort 1 TB in 62 seconds.

Figure 2.5 shows the milestones achieved in the chronology of the Hadoop project based on (White, 2009). In late 2005, Hadoop ran reliably on 20 nodes. Early in 2006, Doug Cutting joined Yahoo! and the Apache Hadoop project officially began to support the stand-alone development of MapReduce and the HDFS. Similarly, the Yahoo! Grid Team adopted Hadoop. In mid-2006, Yahoo! set up a Hadoop research cluster with 300 nodes.

By early 2007, the research cluster had grown to 900 nodes, and by mid-2007, there were two research clusters of 1000 nodes each. By late 2008, the Hadoop project could load 10 TB of data per day onto research clusters. In mid-2009, Hadoop reached 17 clusters, with a total of 24000 nodes, and broke the '1 minute sort' barrier by sorting 500 GB in 59 seconds (on 1400 nodes) and a 100 TB sort in 173 minutes (on 3400 nodes).

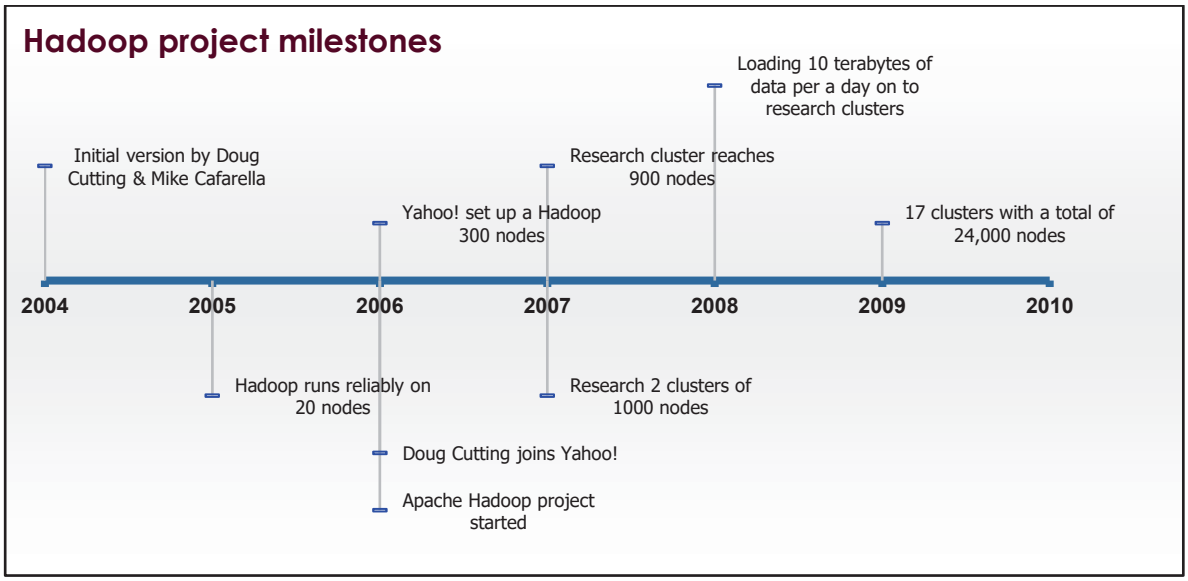


Figure 2.5 Hadoop project milestones

In this section, a historical overview of Hadoop was presented. The various Hadoop subprojects are described in the next section.

2.2.7 Hadoop subprojects

Currently, the Hadoop project is a collection of related subprojects hosted by the Apache Software Foundation, in an open source licensing environment, with the aim of supporting the distributed computing infrastructure. Figure 2.6 illustrates all the Hadoop subprojects, based on (Hadoop-webpage, 2011; White, 2009).

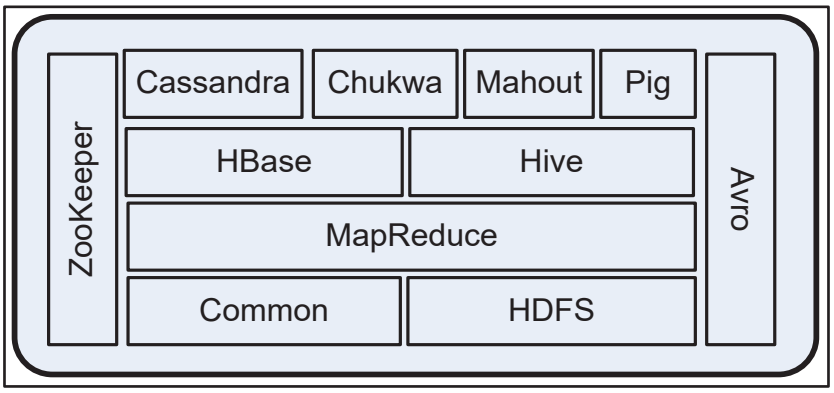


Figure 2.6 Hadoop subprojects

According to Figure 2.6, the subprojects are the following:

- Common: A set of familiar utilities that support the other Hadoop subprojects. It includes interfaces for distributed file systems, general input/output mechanisms, serialization libraries and persistent data structures.
- HDFS: A DFS that runs on large clusters of commodity machines (utility computing). The HDFS is the primary storage file system used by Hadoop applications. It creates multiple replicas of data blocks and distributes them on compute nodes throughout a cluster to enable reliable and extremely rapid computation.
- MapReduce: A software framework for the distributed processing of large datasets on compute clusters. Hadoop MapReduce is a programming model and a software framework for writing applications that rapidly process vast amounts of data in parallel on large clusters of compute nodes.
- HBase: A scalable, distributed column-oriented database that supports structured data

storage for large tables. HBase is the Hadoop database. It is used when random, real-time read/write access to large amounts of data is needed. The goal of this project is to host very large tables; for example, billions of rows by millions of columns, on top of clusters of commodity hardware (utility computing). According to Chang, HBase uses the HDFS for its underlying storage, and supports both batch-style computations using MapReduce and point queries (random reads). Just as BigTable leverages the distributed data storage provided by the GFS, HBase provides BigTable-like capabilities on top of Hadoop, including (Chang et al., 2008):

  - Convenient base classes for backing Hadoop MapReduce jobs with HBase tables, including Cascading, Hive and Pig source/sink modules;
  - Query predicate push down via server side Scan and Get filters;
  - Optimizations for real-time queries;
  - A Thrift gateway and a RESTful Web service that supports XML, Protobuf and binary data encoding options;
  - Extensible jruby-based (JIRB) shell;
  - Support for exporting metrics via the Hadoop metrics subsystem to files or Ganglia, or via JMX.

- Hive: A data warehouse system for Hadoop that facilitates easy data summarization, ad hoc queries and the analysis of large datasets stored in Hadoop-compatible file systems, like the HDFS. It provides a mechanism for projecting the structure onto these data and querying them using an SQL-like language called HiveQL. At the same time, this language allows traditional MapReduce programmers to plug in their custom mappers and reducers when it is inconvenient or inefficient to express this logic in HiveQL, based on (Hive-webpage, 2011).
- Avro: A data serialization system for efficient, cross-language RPC and persistent data storage.
- Cassandra: A scalable multi-master database with no single points of failure.

- Chukwa: A distributed data collection and analysis system. It runs collectors that store data in the HDFS, and uses MapReduce to produce reports.
- Mahout: A scalable machine learning and data mining library.
- Pig: A data flow language and execution environment for exploring very large datasets. Pig runs on the HDFS and MapReduce clusters.
- ZooKeeper: A distributed, highly available coordination service. It provides primitives, such as distributed locks, that can be used for building distributed applications.

This section provided an overview of Hadoop subprojects. In the next section, a more detailed description of HBase is presented.

2.2.8 HBase as a No-SQL databases

The goal of this section is to lay the theoretical foundations for the subsequent sections. We explore the HBase structure in detail, explaining what HBase is, how it originated and how it functions.

Definition of HBase

On the HBase website, we learn that HBase is a non-relational, distributed and column-oriented database based on Google's BigTable. It is written in Java in an open-source model, and is an Apache Software Foundation project. HBase is built on top of Hadoop for its MapReduce and distributed file system implementations. The goal of HBase is to provide a fault-tolerant way of storing large quantities of sparse data (HBase-webpage, 2011). It features compression, in-memory operation and Bloom filters on a per-column basis, as outlined in the original BigTable paper in (Chang et al., 2008).

According to White, HBase tables can serve as the input and output subsystem for MapReduce jobs run in Hadoop, and may be accessed through the Java API, but also through the REST, Avro or Thrift gateway APIs (White, 2009).

In Venner's opinion, HBase is not a direct replacement for a classic SQL database, although its performance has improved recently, and it is now serving several data-driven websites, including the Facebook Messaging Platform (Venner, 2009).

HBase is one of the most popular No-SQL database technologies, it works in very close cooperation with Hadoop and, like others distributed large-scale platforms, they mainly focus on X-nix environments for production installations (Al Mahruqi, 2020). However, being developed in Java, both projects are fully portable across platforms, and so to the Windows operating system as well. For ease of development, these projects rely on Cygwin to secure an X-nix-like environment on Windows for running the shell scripts.

HBase Historical Overview

The history of HBase is too recent to be long. In White's opinion, the most important milestones in this short story were reached as early as the end of 2006, by Chad Walters and Jim Kellerman of Powerset, who coded the first version. It was based on Google's BigTable (Chang et al., 2008). In February 2007, Mike Cafarella performed a code drop, consisting mostly of a working system that Jim Kellerman then carried forward. The first HBase release was bundled with Hadoop 0.15.0. At the start of 2008, HBase became a Hadoop subproject (HBase-webpage, 2011). It has been in production use at Powerset since late 2007. Other production users of HBase include WorldLingo, Streamy.com and OpenPlaces, as well as groups at Yahoo! and Adobe (White, 2009).

HBase Concepts

In this section, we provide a quick overview of the core HBase concepts, based totally on (White, 2009):

- The HBase data model: The applications store data into labelled tables made up of rows and columns. The table cells, which are formed by the intersection of row and column coordinates, are versioned. By default, their version is a timestamp automatically assigned by HBase at the time of cell insertion. A cell's content is an uninterpreted array of bytes.

Table row keys are also byte arrays, so theoretically anything can serve as a row key, from strings to binary representations of long, or even serialized data structures. Table rows are sorted by row key, the table's primary key. By default, the sort is byte-ordered. Table access is via the table's primary key. Row columns are grouped into column families. All column family members have a common prefix, so, for example, the columns 'temperature: air' and 'temperature:dew_point' are both members of the temperature column family, whereas 'station: identifier' belongs to the station column family. The column family prefix must be composed of printable characters. The qualifying tail can be made up of any arbitrary number of bytes (White, 2009).

A table's column families must be specified up front, as part of the table schema definition, but new column family members can be added on demand. For example, a new column 'station: address' can be suggested by a client as part of an update, and its value persists as long as the station column family is already in existence on the targetted table. Physically, all column family members are stored together on the file system. So, although earlier we described HBase as a column-oriented store, it would be more accurate to describe it as a column-family-oriented store. Because tunings are applied at the column-family level, and storage specifications are measured there, it is advisable that all column family members have the same general access pattern and size characteristics. To summarize, White sees HBase tables as RDBMS, except that the cells are versioned, the rows are sorted and columns can be added on the fly by the client, as long as the column family already exists (White, 2009).

- HBase regions: Tables are automatically partitioned horizontally by HBase into regions. Each region comprises a subset of a table's rows. A region is defined by its first row, inclusively, and last row, exclusively, plus a randomly generated region identifier. Initially, a table comprises a single region, but, as the size of the region grows and after it crosses a configurable size threshold, it splits at a row boundary into two new regions of approximately equal size. Prior to this first split, all the loading will be borne by the single server hosting the original region. As the table grows, its number of regions grows. Regions are the units that are distributed over an HBase cluster. In this way, a table that is too big for any one server can be carried by a cluster of servers, with each node hosting a subset of all

the table's regions. This is also the means by which the loading on a table becomes distributed. At any one time, the online set of sorted regions represents the table's total content (White, 2009).

HBase Implementations

In all distributed programming, there are clients, slaves and a coordinating master. In HDFS, for example, there are namenodes and datanodes; and in MapReduce, there are jobtrackers and tasktrackers. Likewise, HBase is characterized by an HBase master node handling a cluster of one or more 'regionserver' slaves, see Figure 2.7, taken from (White, 2009).

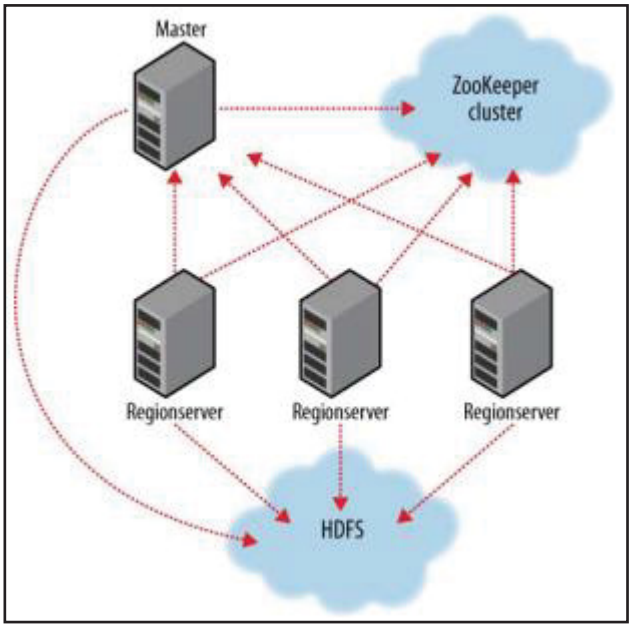


Figure 2.7 HBase cluster members

Based on (White, 2009), we can say that the HBase master is responsible for bootstrapping a virgin install, for assigning regions to registered regionservers and for handling regionserver failure recovery. The master node is lightly loaded. The regionservers carry zero or more regions and field client read/write requests. They also manage region splits, informing the HBase master about the new daughter regions, so that it will manage offlining the parent region and the assignment of the replacement daughters. HBase depends on ZooKeeper, and, by

default, it manages a ZooKeeper instance as the authority on the cluster state. Figure 2.8 offers another view of this scheme. It was taken from (Cryans et al., 2008).

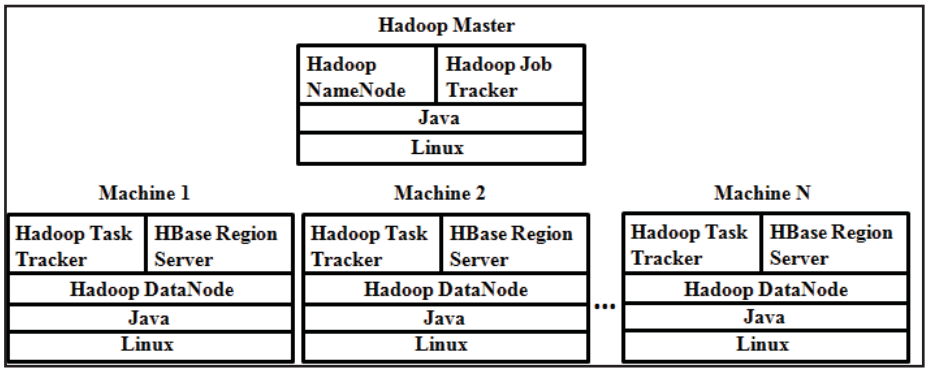


Figure 2.8 HBase infrastructure: master and region servers

The regionserver slave nodes are listed in the HBase *conf/regionservers* file, in the same way that datanodes and tasktrackers are listed in the Hadoop *conf/slaves* file. HBase stores the persistent data via the Hadoop file system API. Since there are multiple implementations of the file system interface, such as Amazon's S3 or the HDFS, HBase can persist data to any of these implementations (White, 2009).

How Does HBase Work

HBase keeps two special catalog tables internally, called ROOT and META. Within these it maintains the current list, state, recent history and location of all regions of the cluster. The ROOT table holds the list of META table regions. The META table holds the list of all user-space regions. Entries in these tables are keyed using the region's start row. Row keys are sorted, so finding the region that is hosting a particular row is a matter of performing a look-up to find the first entry key that is greater than or equal to the requested row key. As regions transition (are split, disabled or enabled, deleted, redeployed by the region load balancer or redeployed due to a regionserver crash), the catalog tables are updated, so that the state of all regions on the cluster is kept current (White, 2009).

The client connecting to the ZooKeeper cluster is the first to learn the location of the ROOT location, and reads the ROOT table to learn the location of the META region, the scope of which covers that of the requested row. The client then performs another look-up of META to determine the hosting user-space region and its location. From then on, the client can interact directly with the hosting regionserver (White, 2009).

In White's opinion, to save having to make three round trip operations per row, the client should cache the information read from ROOT and META. With both the caching locations and the user-space region begin and end rows, the client can identify the hosting regions without having to go back to reading the .META. table. The client continues to use the cached entry while working, until there is a fault. When this happens (i.e. meaning that the region has moved), the client consults META again to learn its location. If the META region has moved, then the ROOT catalog must be consulted again (HBase-webpage, 2011; White, 2009).

At the regionserver level, the writing process is first appended to a commit log and is then added to an in-memory cache. Once this cache is filled, its content is flushed to the file system. When the master notices that a regionserver is no longer reachable, it splits the dead regionserver's commit log by region. On reassignment and before its open for transactions, regions that were on the dead regionserver pick up their file of unpersisted edits that was split and replay them to update their state to just before the failure (White, 2009).

For reading, the region's memory cache is consulted first. If sufficient versions are found to satisfy the query, a result is returned. Otherwise, flush files are consulted in order, from newest to oldest, until sufficient versions are found or until there are no more flush files to consult. A background process compacts the flush files once their number has crossed a threshold, rewriting many files as one, because the fewer files a read consults, the better it will perform. On compaction, versions beyond the configured maximum are deleted and expired cells are cleaned out. A separate process running on the regionserver monitors the sizes of the flushed files that split the region when they exceed the configured maximum (White, 2009).

HBase as a No-SQL Model

HBase uses a No-SQL model with the following characteristics (White, 2009):

- No real indexes: Because the rows are stored sequentially, as are the columns within each row, there is no index bloating and the insert performance is independent of table size;
- Automatic partitioning: As the tables grow, they are automatically split into regions and distributed across all the available nodes;
- Linear and automatic scaling with new nodes: Add a node, point it towards the existing cluster and run the regionserver. The regions automatically rebalance and the load is spread evenly;
- Commodity hardware: Clusters are built at a cost of $1,000 to $5,000 per node. RDBMS have a voracious appetite for input/output hardware, which is the most costly type of hardware;
- Fault tolerance: A large number of nodes means that each is relatively insignificant, and there is no need to worry about individual node downtime;
- Batch processing: MapReduce integration allows fully parallel, distributed jobs against the data, with locality awareness.

2.3 Database Migration Theory

This section aims to synthesize the state of the art in database migration activities, presenting various proposals published to identify the fundamental concepts, approaches and techniques available.

2.3.1 Definition

As we have seen in the RDBMS concept section, a database has different components such as: schema, data, queries and application programs like stored procedures. Authors have been

faced with migrating from one database technology to another for some time. According to Maatuk, database migration is a process in which all the database components of the source database are converted to their equivalent in a target database environment/technology. Typically, the migration process focuses first on the database schema translation and then on the data migration itself (Google Cloud Architecture Center (1), 2020; Google Cloud Architecture Center (2), 2020; A. Maatuk, Ali, & Rossiter, 2008). After the database has been converted, the queries and application programs that acess it need to be adjusted to use the target database technology (as demonstrated in Figure 2.9).

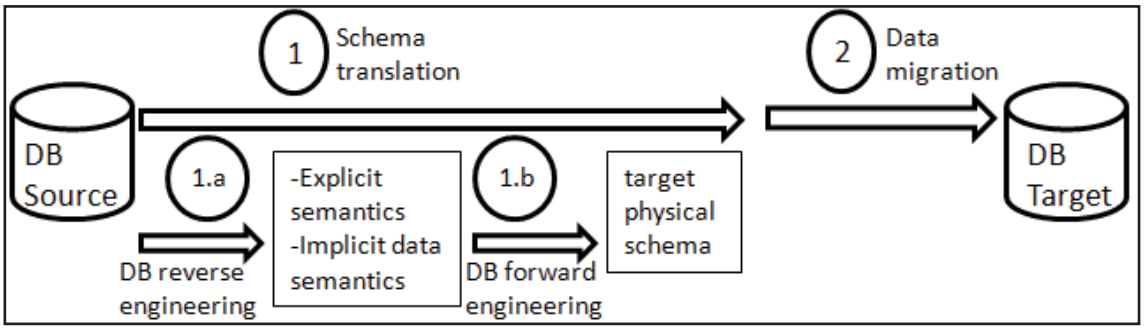


Figure 2.9 Data migration steps

The first component addressed by a migration is the schema translation (step 1 of Figure 2.9), whose goal is the set of mapping rules that needs to be applied. This task consists of two steps: In the first step, the goal is to recover the conceptual schema, for example, work on the entity relationship model. This step also is called the *database reverse engineering* step (step 1.a of Figure 2.9). The resulting model obtained is important because it expresses explicit and implicit data semantics of the DB source schema. The explicit semantics involve relation, attributes, keys and data dependencies. The implicit data semantics are necessary to extract extra semantics that are not expressed explicitly, such as relationships. The second step is called *database forward engineering* step (step 1.b of Figure 2.9) where it takes the results of step one to generate the target physical schema.

The second component addressed by a migration is the data migration itself (step 2 of Figure 2.9) which is a process of converting data from the source database into the target database.

Since the source data of an RDBMS is stored as tuples, they need to be converted into other objects, depending on the technology used by the target database.

Migration activities are always a concern because there are many applications created in older paradigms/technologies that need to be converted to newer technologies. This is globally called, in the literature, the migration database problem.

2.3.2 Database Migration Approaches

According to Maatuk and Fong, database migration can be done using one of three approaches (Fong & Wong, 2004; A. Maatuk et al., 2008; A. Maatuk, Ali, & Rossiter, 2011; Rocha, Vale, Cirilo, Barbosa, & Mourão, 2015):

- Objects like database front-end or mapping tools approach: This approach request deals with schema translation, because data are stored in a relational model but are handled through some object interfaces, such as an object oriented interface or XML interface. Data may be required to be processed in some particular way, according to the object that would be used. Normally, the persistence is handled by the relational part and the object can represent several tuples in several tables, same as the joins required for queries. This can cause a semantic gap problem called OR impedance mismatch. There are two solutions to avoid this problem. First, developers can write huge amounts of code to map objects in programs into tuples in a relational model, which might be time-consuming to write and execute. The second solution is by using OR mapping middleware, which is a software layer that links programming language concepts to data stored in relational databases through ODBC or JDBC drivers;
- Database integration or gateways tools approach: This approach request deals with schema translation too. In this approach there are two databases, the source database (relational) and the target conceptual database system (objects set that simulates the target). A connection between the relational database and this target is created. The applications built on top of the new database access both relational and objects giving an impression that all the data are stored in one database. This approach uses a special type of software called gateways, which support connectivity between databases and

do not involve the user in SQL and relational schema. Hence, queries and operations are converted into SQL and the results are translated into target objects. Most commercial databases provide flexibility on gateways construction among heterogeneous databases.

As reported by (A. Maatuk et al., 2008), the difference between gateways and mapping tools is that with gateways, objects are persistently stored in the new target database system, whereas in the mapping, objects are created and handled in the normal way but are stored in a relational database. However, in both approaches old data, stored in a relational database, is retained.

- Database Migration: The third approach is migrating a relational database into a target database, this means both schema and data are completely migrated into a target database. The first database, the source, is a relational database, and the second one, the target, represents the result of the database migration process. Sometimes, this process needs the help of an intermediate conceptual representation, although it is possible to accomplish this task without the intermediate. Usually, the input source schema is enriched semantically and translated into a target schema. Also, relations and attributes are translated into equivalent targets and some elements, such as relationships or foreign keys or data dependencies, may be replaced by another equivalent domain in the target. Data stored in the source database is converted to the target database. Due to heterogeneity between the concepts and structures of source and target data models, the migration process faces several challenges, for instance, data of one relation may be converted into a structure or references set rather than into one corresponding type.

2.3.3 Migration Translation Techniques

Once a migration approach has been chosen, techniques must be clarified. According to Maatuk and Fuxman, the database migration techniques are (Fuxman et al., 2006; Ha & Shichkina, 2022; A. Maatuk et al., 2008; A. Maatuk et al., 2011):

- Source-to-Target technique: This technique translates a physical source code into an equivalent target. However, all the operations are done without a semantic enrichment

process. This results in an incomplete design, which means some data semantics are ignored.

- Source-to-Conceptual-to-Target technique: Given that the above technique has problems with semantics, because they are not clearly expressed (including relationships), in this technique the schema is translated from logical into conceptual, then the conceptual schema is translated into the target. The source schema is enriched by recovering the domain semantics and making them explicit. This technique results in a well-designed target database. Achieving a conceptual schema from a logical one (relational database schema) requires a thorough analysis of the schema, data and queries.

2.3.4 Overview of Database Migration State of the Art

Database migration publications follow the database evolution itself. Publications about this topic start in the late 60's with the use of two popular database management systems at the time: 1) the hierarchical database; and 2) the network database. IBM designed the IMS database with Rockwell and Caterpillar starting in 1966 for the Apollo program. This database used a hierarchical model and required the schema to be compiled. In a network database model, the data schema is viewed as a graph composed of object types, such as nodes and relationships like arcs. The network database model's original inventor was Charles Bachman. It was so popular that it was developed into a standard specification published in 1968 by the CODASYL Consortium. This database model was conceived as a flexible way of representing objects and their relationships. In 1969, IBM developed their own commercial network database called IDMS. Around the same time, the relational database model was invented and proposed to the market in 1970 by Edgar F. Codd. This database model did not require compiling its schema and was based on first-order predicate logic. The purpose of the relational database model is to provide a declarative method for specifying data and queries. The resulting data access model describes data structures for storing the data and ensuring that the access procedures to retrieve data from queries are satisfied.

In 1985, the object-oriented database management system (OO-DBMS) was invented with the goal to combine the relational database capabilities with the object-oriented programming language capabilities. This proposed database technology allowed for object-oriented programmers to store objects, replicate and modify them within an object-oriented database. Since this database integrates the object-oriented programming language concepts, the programmer can easily access objects from the database to the computer memory without any migration treatments. In contrast, using the relational model with an object-oriented language always requires that a migration of the data formats be done between the database formats and the objects in memory. This technology is still not very popular today.

Database migration publications start to appear in 1989, when database performance issues and migration used between different database technologies were debated. The first paper addressing the migration issues from a network database model to a relational database model (Fong & Chris, 1994) was published this same year. In 1997, a paper discussing a database migration that converted a relational database model to an object-oriented database model was also published (Fong, 1997).

Despite that XML database management systems were initially proposed and used since 1994, it was not until 2001 when the first of two publications that discussed migration to this type of model appears, the other was in 2004. Both discuss issues of migration from a relational model to an XML database model and also address the migration from a relational model to an XML for Internet computing. (Fong, Pang, & Bloor, 2001), (Fong & Wong, 2004). Figure 2.10 shows the main milestones in the history of migration topics over the years.

2.3.5 Examples of No-SQL Migration Attempts from Relational DB

In this section, an example of how a company has converted to No-SQL technology as part of the solution to their current SGBD processing problems (Lam, 2011) is presented. This example applies to the social networking, media and entertainment domain but could also touch many other business domains.

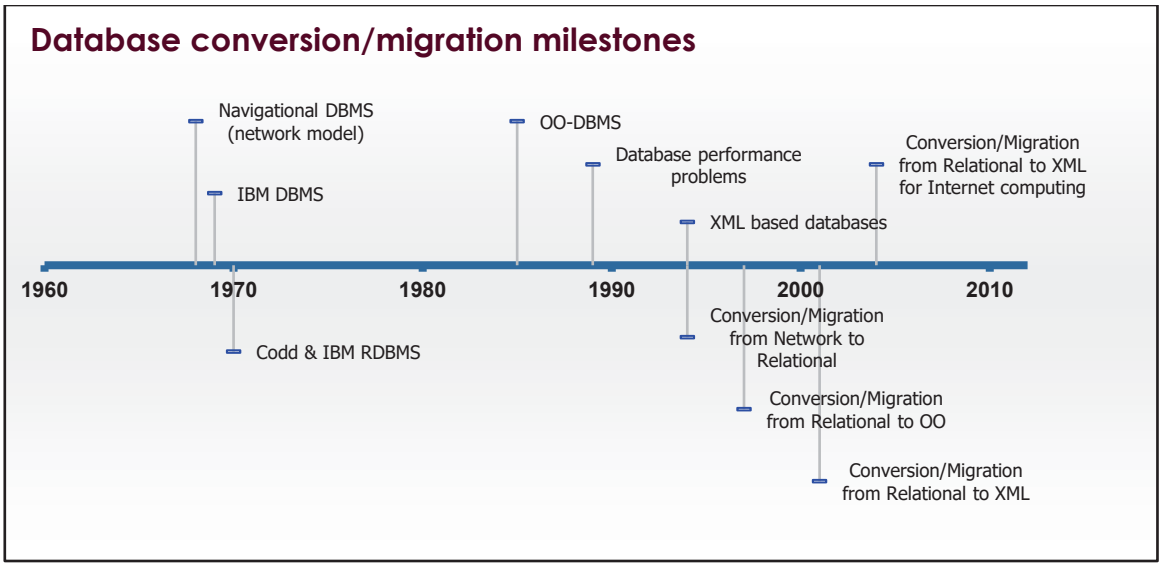


Figure 2.10 Migration database milestones

The Facebook case study

The information that Facebook handles every day is huge and it needs to scale constantly. Since hardware and software are resources that are key to their operations, the company started to analyse the reliablility, ease of use, scalability and maintainability of its many RDBMS databases. Initially, data processing at Facebook was performed entirely on Oracle RDBMS technology, a popular RDBMS. As the data and the site's use began to grow, Facebook investigated whether or not there was an open-source technology that they could use to eliminate their growing licensing costs. As part of this investigation, it deployed a relatively small Hadoop proof of concept and began testing some of their core datasets with it. In White's opinion, Hadoop was attractive to Facebook because Yahoo! was using it successfully for its batch processing needs and also because they were hearing about the scalability of the MapReduce programming model, used by Hadoop, for distributed No-SQL DB as popularized by Google's BigTable technology (White, 2009).

According to (Hadoop-webpage, 2011; HBase-webpage, 2011), Facebook is running the third largest Hadoop cluster in the world currently. This cluster has 12 PB of storage, and more than 10 TB of data loaded into it daily. It operates on 8800 cores and about 9 TB of memory, and reaches its capacity many times during the day. The ability of the cluster to scale out rapidly in

response to Facebook's growth is an important and strategic advantage. Facebook also has modified Hadoop to suit its changing needs since it is an open-source software. Consequently, Facebook has contributed often to the open source community, both in the form of contributions to some core components of Hadoop, such as HBase and Hive. According to White, Facebook uses No-SQL technology in at least four different but interrelated ways (White, 2009):

- Use Facebook data to produce daily and hourly summaries, which can be classified into 3 groups:
  - Summaries that help engineering and non-engineering functional teams take product decisions;
  - Summaries that produce metrics about the advertising campaigns conducted by Facebook;
  - and summaries that obtain information about consumer preferences, such as people or applications the user may like.
- Extract information from historical data that could assist in the decision-making process of different product groups and executive teams;
- As the default information container for the log files;
- To look up the log information, filtering by pre-defined attributes, in order to do very specific tasks, such as help users against spam bots.

The New Facebook Messages

As presented by Aiyer, between 2008 and 2009, Facebook faced a lot of issues with a new project called "The New Facebook Messages" that was designed to host Messages, Chats, Emails and SMS in a single application. Before this project, each Facebook application had their own dedicated hardware/software infrastructure which included mostly the MySQL RDBMS technology as well as a No-SQL DB called Cassandra which was developped

internally by Facebook (Aiyer et al., 2012). For the New Facebook Messages service, a number of target objectives were set:

- An excelent read and write performance;
- The possibility to scale out at the horizontal level easily;
- An automated fault recovery service;
- A strong data consistency data model and service;
- The use of HDFS and MapReduce.

These requirements lead the Facebook team towards choosing the HBase No-SQL database technology, which completely fulfilled all these requirements. By the time the project was started in 2009, Facebook estimated the new application would need, at the time of release, the capacity to store over 6 billion messages/day (both person-to-person message and chat messages) which represents nearly 75 billion read/write operations per day, with estimated peaks around 1.5 million operations per second. In addition, Facebook expected percentages of read/write operations at a rate of 55% read and 45% write. These estimates resulted in a total size between 2 PB to 6 PB when considering the data replication, and an expected growth size estimated at a rate of 250 TB per month.

The problem was not that MySQL could not scale. It is totally possible for MySQL to scale but the problem begins when you are scaling an RDBMS to this massive size. Updating an index would take a long time, some Facebook statistics could not be properly updated and queries for users would perform in a poor way. With the Facebook Messaging project, it is normal that the data sharding option becomes a difficult problem to solve using an RDBMS. As an example, just finding the right sharding approach would be a real challenge since the data needed to be replicated across all of the database servers in production. Also, the sharding algorithm should allow for the easy addition of new servers without any lengthy set up as Facebook needed growth on demand in this project.

The project started in December 2009 and the first version was ready for testing in November 2010. The entire application was finished by July 2011. The migration project from RDBMS to HBase had to migrate more than 1 billion accounts from the legacy messages. By the time

the project was finished, Facebook reported two key issues with the migration process: 1) they had a hard time identifying the target schema in HBase and changed the schema twice; and 2) they failed to identify how to migrate certains aspect of the RDBMS, such as metadata, and how to benefit from the notion of column families in HBase (Al Mahruqi, 2020).

2.4 Conclusion

This chapter presented three different literature reviews: 1) the RDBMS technology; 2) the No-SQL database technology; and 3) the different approaches for database migration.

In the first review, an introduction to RDBMS technology was presented, followed by the introduction of different database concepts, such as ACID properties, the CAP theorem and the current limitations facing this technology.

This initial review was followed by an overview of the No-SQL database technology which provided an overview of the advantages and challenges this new technology offers. Pertinent to the research, a section was dedicated to the introduction of the Hadoop project and its No-SQL database technology.

The third and final section of the literature review included a discussion of the different techniques used for database migration as well as the Facebook case study and the database migration challenges they encountered while using only their experience to conduct this complex migration process.

The discussion so far highlights the lack of formal guidance in this specific area. This creates an opportunity to research how to facilitate these types of migrations that are sure to become more popular as companies start using No-SQL database technologies. In other words, migrations from RDBMS to No-SQL are conducted using only the experience of the software engineers. Can migration rules be uncovered and be made available to help? This topic is addressed in the next chapter with the presentation of an experimental approach to developing these migration rules.

CHAPTER 3

EXPERIMENT – MIGRATION BASED ON HEURISTICS

We have already presented that the main goal of this research project (TRACK 1) is to uncover guidelines to potentially help to improve the migration process from RDBMS to No-SQL databases for software engineers that conduct this migration for the first time. The intended result of this research is to offer software engineers, which may have expertise in RDBMS but not in No-SQL migration, a more formal migration step by step process to guide them. To achieve this goal, it is important to establish a baseline to allow a comparison between the results of migrating with and without help.

This chapter presents the design of an experiment that will establish a baseline to provide a valid comparison between the results of a migration processes, based only on experience (this thesis – TRACK 1), and future research (TRACK 2) that would use these guidelines. This chapter content publishes as a first research paper (Gomez, Ouanouki, Ravanello, April, & Abran, 2015) which was awarded best paper of the conference "*Cloud Computing 2015: The Sixth International Conference on Cloud Computing, GRIDs, and Virtualization*".

When new technologies and paradigm appear, such as the No-SQL databases, most organizations look for help in the literature or use only their experience to attempt this type of database migration process. At the beginning of this research (in 2014) and conducting this experiment, little to no previous work/research had been published in this specific area. Indeed, at the time of conducting the experiments, there has only been preliminary research published focusing on the migration of certain elements such as database tables limited to a few types of database relationships.

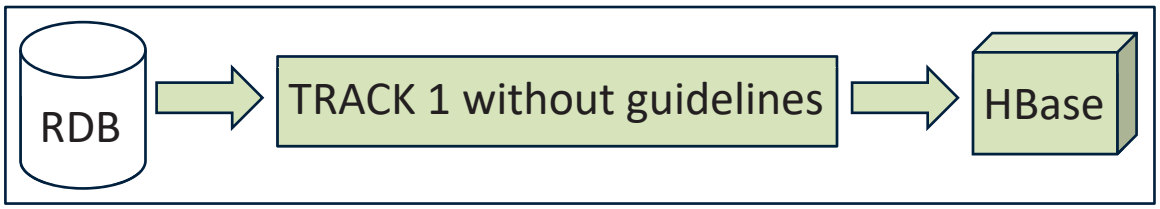


Figure 3.1 Chapter 3 Objective

3.1 Experimental Design

The experimental design of Track 1 of this research is based on an experiment where participants will conduct a database migration using only heuristics. The next section describes the experiment participants as well as the data collection procedure.

3.1.1 Experiment Participants and Data Collection Procedure

This subsection follows the experimental recommendations and structure of (Easterbrook, Singer, Storey, & Damian, 2008), (Marcos, 2005), and (Zelkowitz, Wallace, & Binkley, 2003); moreover, the experiment was designed using the point of view of a typical developer. Using this point of view, the participants were asked to state their experience level and they were classified according to: 1) their academic background; 2) working field; 3) number of years of work experience with relational databases; and 4) the number of years of work experience with any No-SQL database.

The word "experience" was related to the following IT domains: programming, relational database use as a programmer or relational database use as a database administrator. Additionally, the classification was summarized according to different choices. The academic background choices are: Graduate with PhD, Graduate with Master, Graduate, and Undergraduate Student. The working field options are: Industry, Academic, and Research Center.

The number of years of work experience with a relational database environment have the following options: No Experience, Low Experience (less than a year), Average Experience (2 to 5 years), and Advanced Experience (more than 5 years). The number of years of work experience related to any No-SQL database had the same options as above.

The goal of these questions was to obtain a fine grain classification for the participants according to their experience. This would allow us to know the combinations (pair) "relational-No-SQL" of experience where the migration guidelines could be most useful. Figure 3.2, for instance, highlights the pair (Relational Database Low- No-SQL Database Medium), meaning that a participant had "low" experience with a relational database environment and "medium"

experience with No-SQL databases.

Relational Database	No-SQL Database
Low	Low
Medium	Medium
High	High

Figure 3.2 Participant's classification

Twenty individuals participated in the experiment: fourteen participants worked in the industry and three participants worked in the academic sector, one participant came from both sectors (industry and academic) and the last two participants decided do not submit any documentation. All participants were provided with a clear and well established understanding of the purpose of the experimental workshop. The material used in the execution of the experiment was (refer to the annexes of this thesis for example of the database migration results):

1. A document including the call for participants (date, time, place and activities that took place in the workshop), which was an invitation sent by email, followed-up with phone call reminders two weeks before the workshop;
2. The participant's instructions guide;
3. A relational database schema (Blue document). A graphical representation of the database schema that they must migrate to the No-SQL database. This schema is reused from the research of (Singh, 2010) and contains seven data tables: four large tables (City, Department, Doctor and Hospital) and three junction tables (DoctorDeparment, HospitalCity and HospitalDepartment,). The data tables City, Department, Doctor and Hospital have "Id, Name" structure, with "Id" as primary keys. Table "Doctor" contains "Id, Name, Age, Sex and BorIn". The latter field is the Id of the city where the doctor was born. The junction tables allow for expressing the "many-to-many" relationship indicated by each junction table's title (as depicted in Figure. 3.3). It is

important to note that the participants were offered the opportunity to choose between several sub-schemas from the main schema. For instance, one participant could choose only to migrate the sub-schema composed by the entities Hospital – HospitalDepartment – Deparment or the sub-schema Doctor – DoctorDepartment – Deparment. Alternatively, the participant could select the entire schema for his migration attempt (see Figure. 3.3).

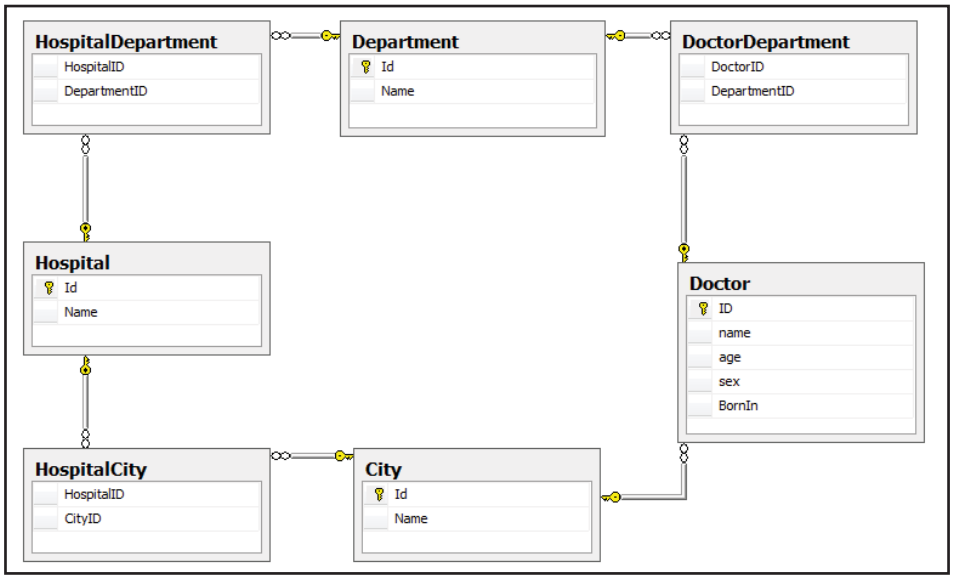


Figure 3.3 Relational schema given to the participants

4. The No-SQL solution (Green document). This document is a blank sheet of paper where the participant will draw the target schema proposed for their migration. To do this they only use their previous knowledge about RDBMS and No-SQL database technologies;

5. The participants training document (White document). It is a document that summarizes some key knowledge presented in the training session provided to each participant, including relational database and No-SQL explanations;

6. The drafts documents (Yellow documents). Blank paper that can be used to draw anything the participant needs to draw.

7. The experiment final survey form. This survey was designed following the

research work devised by (Kasunic, 2005) and (Lethbridge, 1998). It is composed of nine questions, where the first four were oriented to "experience classification", as explained earlier. The fifth question was related to the migration process aimed at understanding how the participant started his migration effort (e.g. what was the first step of his process). The sixth question is aimed at understanding the effort needed to achieve this process (i.e. without any guidelines). This question was rated from 1 to 5, where 1 indicates that the process was easy to achieve without too much effort, a value of 3 indicates that it required a large amount of effort to achieve it and a value of 5 means that no matter how much effort was put in, it was not possible to do the migration during the experiment. The seventh question is designed to assess the level of confusion experienced during the migration process, e.g., no idea where to start or what the next step was. The question was rated from 1 to 5: always confused, very often confused, sometimes confused, rarely confused, and never confused. The eighth question is a matrix for evaluating the percentage that covers the designed solution with regards to the relational aspects mentioned earlier (Table, Constraint, PK, and FK). Finally, the ninth question asked for the participant's opinion with respect to whether he/she thinks that receiving some guidelines could improve the process. This question was rated 1 to 5 with the levels: strongly agree, agree, undecided, disagree, and strongly disagree. It is one of the two measurement instruments used in the experiment. The second one was the schema designed on the green document.

3.2 Experiment Results

As previously mentioned, there are no previously published experiments or available data that support the process or decisions made when migrating an existing database from RDBMS to No-SQL databases at the time of conducting this experiment. Generally speaking, these migrations have been conducted using a heuristic approach, i.e., with the developers experience or the developer's educated guesses and their common sense.

The experiment was conducted on july 24 and 31, 2013 and it consisted of two parts. First, a 30 minute training session provided an explanation of the technological context, including a short tutorial about RDBMS and the No-SQL technology. All the participants received this training as well as the Track 1 experiment documentation listed in the previous section. Subsequently, the participants conducted the experiment, eventually completing the green sheet (i.e. the No-SQL schema resulting from the migration). Finally they expressed their opinions by filling out the survey.

Table 3.1 reports on the different educational levels of the participants. It reports a few undergraduate students (6%) and 94% (5% with PhD, 50% with masters plus 39% of graduates) of graduate students that showed an interest in participating in the experiment.

Table 3.1 Educational Level of the Participants

Educational Level	
Classification	***Response in percentage***
Graduate with Phd	5%
Graduate with Master	50%
Graduate	39%
Undergraduate	6%

Table 3.2 Work Area of the Participants

Area of work	
Classification	***Response in percentage***
Industry	83%
Academic	17%
Research Center	0%

Table 3.2 shows a large participation from the industry sector (83%). Also, it can be observed, in Table 3.3 that a large number of participants have experience in RDBMS technology with 45% that have more than 5 years of experience. These results, together with those of Table 3.2

(83% of participants in industry sector) support the importance of improving the migration process.

Table 3.3 Level of Experience in DB Domain

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

In contrast, Table 3.3 also shows that 94% of the participants have no knowledge of No-SQL database technology. This is aligns with the fact that these technologies are not well understood in general. This finding is also aligned with the objectives of this research. The results shown in Table 3.3 indicate that a set of guidelines could be an interesting tool for this audience.

With regards to the first action to do at the beginning of the migration process, Figure 3.4 provides the different paths chosen by the participants. Considering 83% of participants are in the industry sector (Table 3.2) and 45% with have more than 5 years of experience (Table 3.3), there was a large proportion of 61% (resulting from 33% plus 28%) of the participants that chose to start this migration process using the data “tables” element (see Figure 3.4). Starting with tables is totally preferred in comparison with relationships. However, it is important to point out that 61% understood the importance of its composition. The 33% represents the option related to the goal of obtaining a one-to-one correspondence between the original database and the target database. This will be helpful in case the result will be evaluated at the “Coverage Evaluation” level. It means, at the process end this decision will result in 100% cover in the relational aspect “tables”.

On the other hand, 28% of participants chose to mix some tables to obtain one; they were not interested in obtaining a 100% “Coverage Evaluation” for the “tables”. Their primary objective was the information itself and not how this information could be represented.

The difficulty of completing the migration process is reported by Figure 3.5. As can be seen,

the initial perception is that this database migration process is difficult (nearly 78% resulting from 39% plus 39%). This result is reinforced considering that Table 3.3 also shows the low level of experience in No-SQL databases technologies. The participants reported that this process demands a considerable amount of effort, likely because No-SQL databases are new to them.

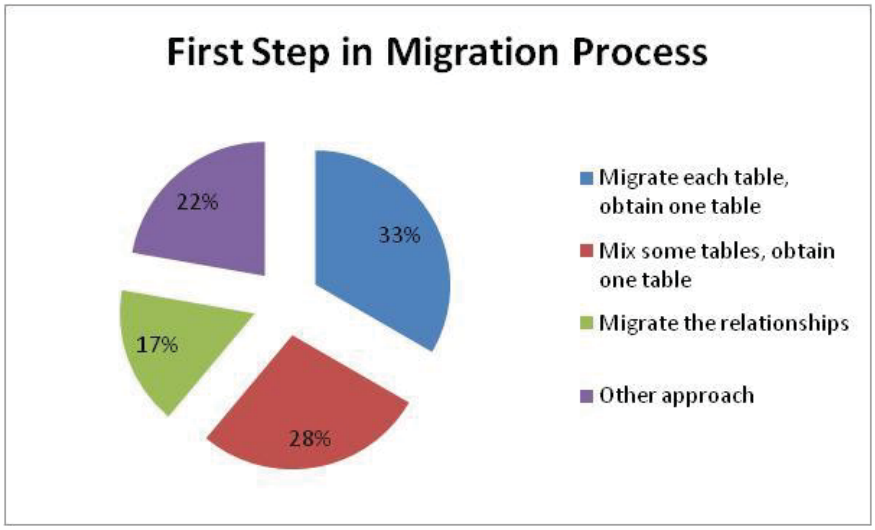


Figure 3.4 First step in the migration process

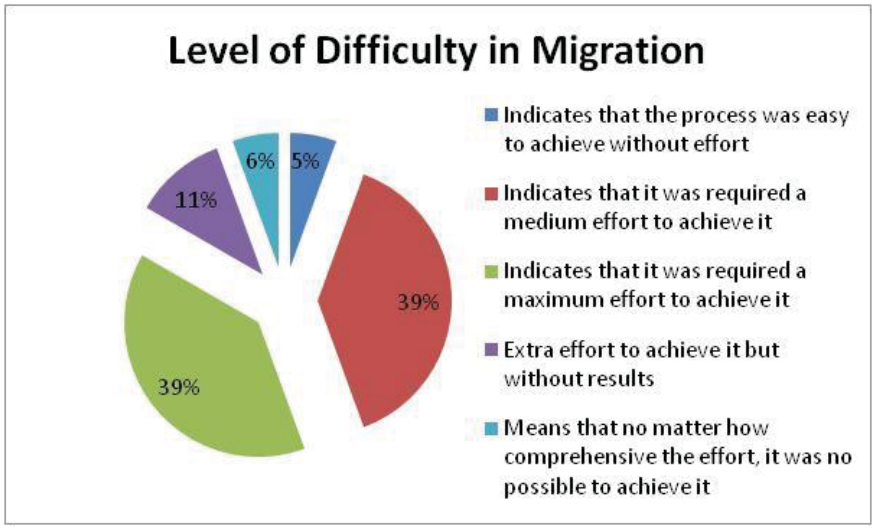


Figure 3.5 Level of difficulty in the migration process

The above argument is further reinforced by Figure 3.6, which demonstrates that the majority

of the participants (44%) felt sometimes confused, i.e., not knowing how to go about it.

Figure 3.6 provides the results of the opinion of the participants with respect to the question about being provided a set of guidelines: 28% strongly agreed with their usefulness and 44% agreed with the relevance of this kind of tool in the migration process. That reinforces and demonstrates the importance of the present research, as the majority of the participants (44%) had felt sometimes confused.

With respect to different database aspects, only five were studied in the experiment: tables, constraints, Primary Keys (PK), Foreign Keys (FK) and others (including all the aspects not specified in a clear way such as fields, types of relationships, views, indexes, procedures and triggers).

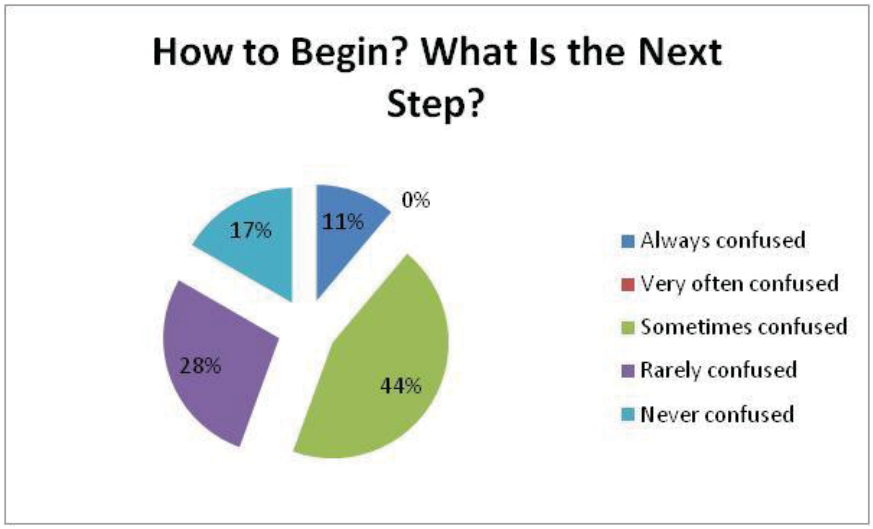


Figure 3.6 How to begin the process?

Table 3.4 provides the information discussed above regarding the relational aspect covered against the percentage of coverage in terms of 0%, 25%, 50%, 75% and 100%. For instance, 50% of the participants think that their solution covers 100% of the relational aspect “tables” . In contrast, 22% think that their solution did not cover this aspect at all (0%).

Moreover, Table 3.4 shows that 28% of the participants think that their solution covers 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 3.4 reports that 41% of the participants think that their solution covers 100% of the relational aspect “primary keys”. However, 29% think that their solution did not cover this aspect (0%).

Table 3.4 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%

Table 3.4 shows that 39% of the participants think their solution covers 100% of the relational aspect “foreign keys”. Conversely, 28% think that their solution did not cover this aspect (0%).

Other relational database improvement aspects like fields, store procedures or triggers were combined in the relational aspect “others” and the results reveals that 94% of the participants showed no interest in these aspects (see Table 3.4, last row).

3.3 Conclusion

This chapter presented the experiment artefacts and participants profile as well as the results of their database migration activity. The data obtained in the experiment (see annexes) show the many personal approaches that were taken to try to do the database migration. Despite the fact that this database migration is possible using an heuristic approach based only on a software engineer’s experience, it was demonstrated that not all of them with their current expertise in RDBMS had enough expertise in the No-SQL domain to carry out the migration fully.

A surprising outcome of the experiment was that nearly all participants tried to migrate the relational aspect “table” first, but they did not pay attention to other relational aspects like

“relationships” or “fields”. To explain this empirical finding, the background of the participants was considered and the results support that their decisions were driven by their large RDBMS experience.

The results of this experiment suggest that each individual migration approach is to be studied in detail to extract migration rules that could be helpful if they were explicitely presented to the participants of TRACK 2 experiment to be conducted in the future. Another conclusion of this experiment is that it is reasonable to conclude that those without familiarity of the RDB domain experienced even more difficulties than those with some RDB experience.

It was also observed that the participants spent a considerable amount of time consulting the reference documentation, and they also reported that the first obstacle was to figure out how to start the migration process (e.g. what is the first step).

The next chapter explores the detailed results of this database migration experiment to try to uncover migration rules.

CHAPTER 4

PROPOSED GUIDELINES FOR MIGRATION

This chapter identifies, explores, uncovers, and describes the proposed guidelines as a potential solution to improve the migration process from RDB to No-SQL databases. Section 4.1 establishes a vocabulary about what exactly should be understood by guidelines, how important this should be for the migration and why it is important in terms of the standardization of the migration processes. Section 4.2 presents the guidelines themselves and describes the procedure to carry out the migration, and finally, Section 4.3 presents a summary of the chapter focusing on the results and the way these findings could be used to improve the migration process.

4.1 What exactly do "guidelines to migrate" mean

In this chapter, the phrase "*guidelines to migrate*" is a way to describe an appropriate, repeatable, standardized, and documented way to conduct a migration process from RDB to No-SQL DBs. It includes but is not limited to a set of processes, tools, and techniques. It could be thought of as an adaptable template to conduct the above-mentioned migration, a way to classify the current RDB, selecting all its relational aspects (see section 2.1) and analyze the way the No-SQL DB (in this case HBase) could handle each of these relational aspects, wherever possible. This will speed up the database migration process, even for development teams that are first-timers on migration projects. The final objective will always be to deliver the target No-SQL DB more quickly.

Among the desired qualities of the proposed solution, the *guidelines to migrate* should have the following characteristics:

- Adjustable to the size and complexity of the project, in other words, they should be flexible and scalable enough to be able to be used in very specific migration cases;

- Easy to understand and to be applied by a software engineer that is attempting to conduct this migration for the first time;
- Subject to the process of repeated refining, always providing an updated solution.

4.2 Guidelines to migrate by relational aspect

The migration attempt of RDB to No-SQL database, specifically the column oriented one as in HBase had different design properties that should have been taken into account in the target HBase had different design properties that should have been taken into account in the target database schema by the participants. A roughly quick migration without regard to the specificities of HBase has resulted in a poor design and, as a consequence, in a solution that does not align well with the properties of a well migrated target database schema.

To study each of the participants proposed solution (see annexes), we will use the same simple example that was shown in chapter 3, i.e. the *Hospital-Doctor* RDB schema that was given to the participants in the experiment, with information about hospitals, doctors, departments and cities. Also, the outcomes of the experiment explained in chapter 3, in which nearly all participants, based on their large RDB experience, started with the assumption that they should migrate “table” by “table” from RDB to HBase, but did not pay attention to other relational aspects like “relationships” or “fields” will be taken into account.

The strategy used by some of the participants in the experiment was not completely wrong. Indeed, this section will use that outcome as an input but in addition will put into practice a methodology called DDI, which is an acronym for “De-normalization, Duplication, and Intelligent Keys”. This methodology was created by (Salmen, Malyuta, Fetters, & Norbert, 2009) and has also been widely used by (Lars, 2011). DDI is recommended since the source database schema and the target database schema are so different that a migration process should consider first the appropriate way to design the target database schema and, at the same time, profit from all the advantages that HBase could offer as a No-SQL database.

Given that HBase does not provide the sophisticated query processing and optimization capabilities of RDB schema, DDI recommends setting as much data as possibly available in a

single row of the target schema. In order to accomplish this goal DDI provides:

De-normalization (D): With the goal of retrieving data via fewer searches from the schema. The *"de-normalization"* concept comes from the RDB world and the idea behind it is to have all the information about an entity and the entities related to it in the same table.

Duplication (D): The duplication task is related to de-normalization and both have the same purpose, which is optimizing the solution for fast reads without any further processing. The intention is to duplicate the information in more than one table with the consequence that, at read time, no further aggregation is required.

Intelligent Keys: HBase physically orders the rows using the row keys. A row can be easily accessed using the row key value. One way to take advantage of this kind of storage is to analyze whether the key should be designed as a composition of the attributes that are most often used as search criteria. This was brought from the RDB world where an index could be designed as a combination of multiple columns.

Combining the DDI principles with the experimental results described in chapter 3 and the literature review done for this research, a series of guidelines were designed and it is strongly recommended to take them into account during the process of migrating an existing RDB schema to a No-SQL database schema, specifically HBase.

The developed guidelines will be applied to all the relational aspects contained on the *"Hospital-Doctor"* RDB schema. In other words, all the aspects will be listed focusing on the guidelines that create that list. Please note that some HBase concepts are not present in the RDB world and it could be hard to explain them using such a simple RDB example. If this is the case, another appropriate example will be used.

Guideline No. 1: *Identify each relational aspect in the source RDB schema and analyze if it can be implemented in HBase and how it could be implemented.*

The idea of this guideline is to make a list of all the aspects that should be migrated, considering there is a possibility that not all aspects on the list will be migrated either because that aspect is no longer needed in HBase or that HBase does not support it.

Figure 4.1 shows the simple RDB schema used for the experimentation. Most of the relational aspects will be covered but, as stated earlier, the chosen example does not cover all of them. (Please see section 2.1 for a detailed list and theory about it).

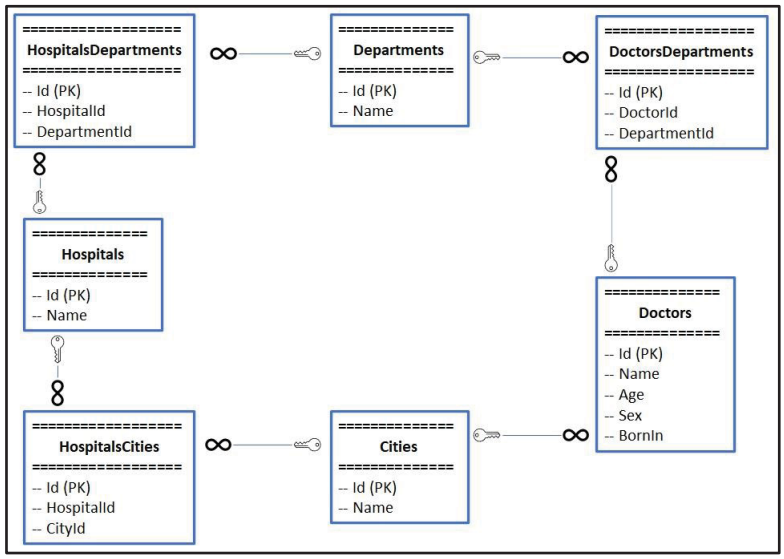
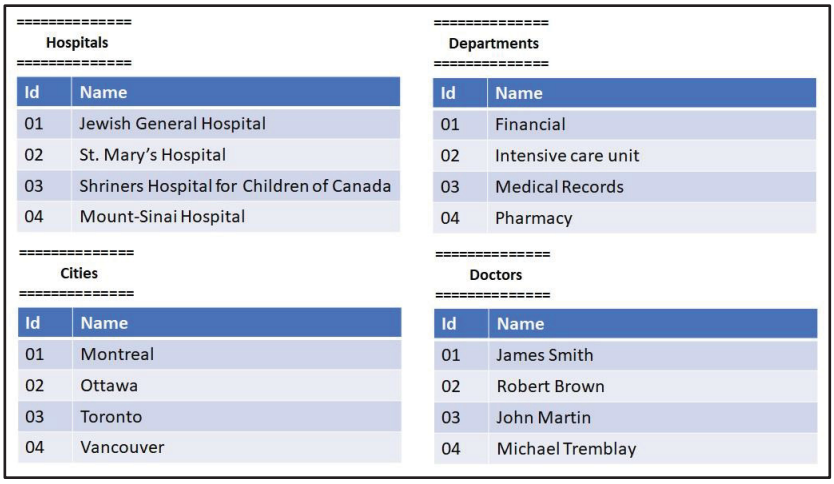


Figure 4.1 RDB Schema from Chapter 3 experiment

Relational Aspect "Tables": In Figure 4.2, the *Hospital-Doctor* database shows four tables: Hospital, Deparments, Cities and Doctors. Each table contains information related to hospitals, departments, cities and doctors accordingly. This relational aspect will be covered if we have at least one HBase table. In HBase, a table is a collections of rows. Please note that no matter how many tables the RDB schema has, it does not mean we will necessarily have the same number of tables in HBase.

Relational Aspect "Columns/Attributes": In HBase, a "column" is a collection of key value pairs and a "column family" is a collection of columns. In an RDB schema the columns are designed as shown in Figure 4.3. The column family concept is not present on an RDB schema. Therefore the following is the columns list from the example in Figure 4.1. It lists 21 columns (see Figures 4.4 and 4.5). The migrated solution in HBase could have more or less tables, but in any case, it will implement the columns concept and also the column family concept.



==========
Hospitals
==========

Id	Name
01	Jewish General Hospital
02	St. Mary's Hospital
03	Shriners Hospital for Children of Canada
04	Mount-Sinai Hospital

==========
Departments
==========

Id	Name
01	Financial
02	Intensive care unit
03	Medical Records
04	Pharmacy

==========
Cities
==========

Id	Name
01	Montreal
02	Ottawa
03	Toronto
04	Vancouver

==========
Doctors
==========

Id	Name
01	James Smith
02	Robert Brown
03	John Martin
04	Michael Tremblay

Figure 4.2 Relational aspect "tables" for the given schema

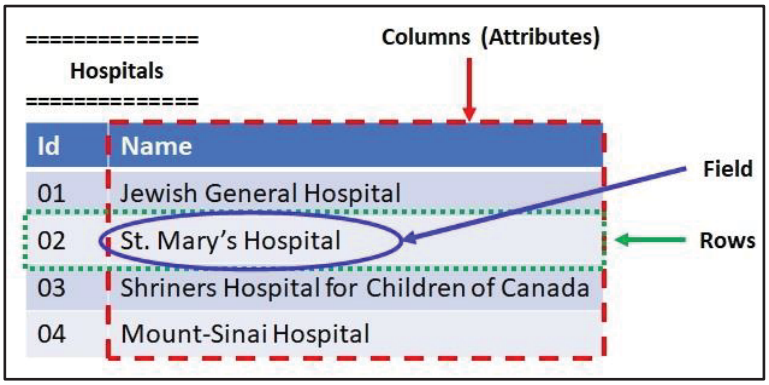


Figure 4.3 In RDB Schema intersection row-column

From Hospitals	From Departments	From Doctors	From Cities
Id (PK)	Id (PK)	Id (PK)	Id (PK)
HospitalId	Name	Name	Name
DepartmentId		Age	
		Sex	
		BornIn	

Figure 4.4 Column's list from the Figure 4.1

From HospitalsDepartments	From DoctorsDepartments	From HospitalsCities
Id (PK)	Id (PK)	Id (PK)
HospitalId	DoctorId	HospitalId
DepartmentId	DepartmentId	CityId

Figure 4.5 Relationships and Columns from Figure 4.1

Guideline No. 2: *Create tables with columns families based on similar information.*

Before applying this guideline, it is important to know how HBase works. All data for a given column family goes into a single store on HDFS (Hadoop Distributed File System, see details in section 2.2.5). In that store, it might have one or multiple HFiles. Columns in a column family are all stored together on disk (inside the HFile), and this way of storage has significant implications for table/columns design. For instance, as suggested by (Dimiduk & Khurana, 2013; Ouanouki, April, Abran, Gomez, & Desharnais, 2017), the columns in one family are stored separately from the columns in another family and it should be a good idea that information with similar access patterns be designed to stay within the same column family.

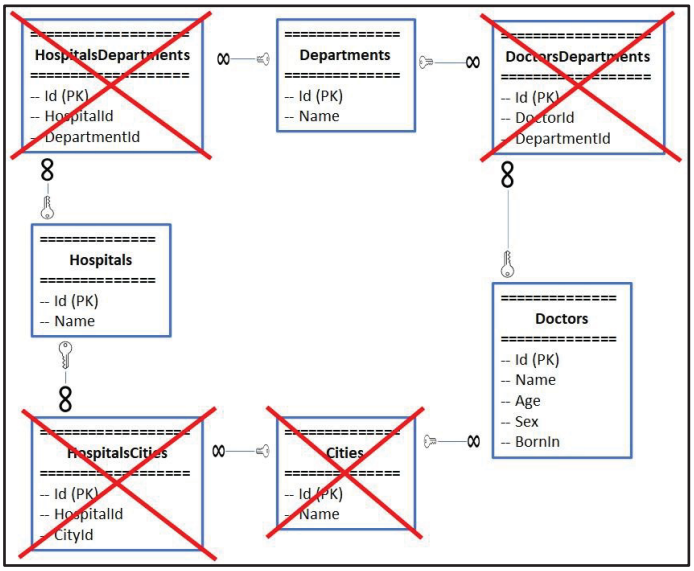


Figure 4.6 Table schema after guideline No. 2

That will save time at read operations. Similarly, information with different access patterns must be stored in different column families and if there is information that is not often queried, it must be assigned to a separate column family.

After guideline No. 2 is applied, the access pattern analyzed for this example suggests one of the best ways could be to have only 3 tables, *doctor*, *hospital*, and *department*. The other tables will be removed (see Figures 4.6 and 4.7).

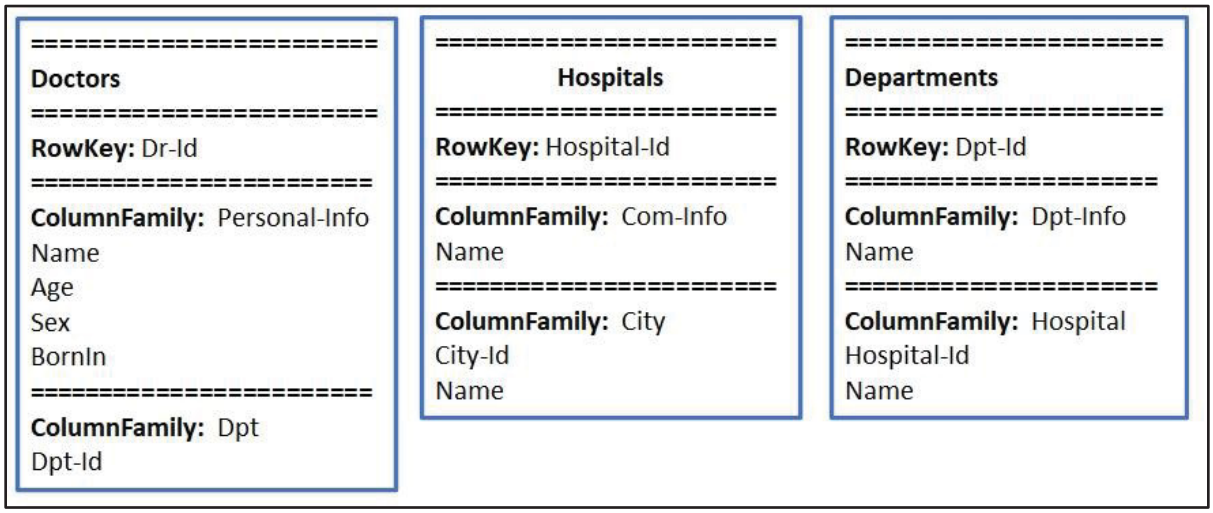


Figure 4.7 Tables with columns families based on similar information

Relational Aspect "Rows": As shown in Figure 4.3, the second row, *"02 St. Mary's Hospital"*, is a way to organize the information inside the table *"Hospital"*. Similarly, HBase rows consist of a row key and one or more column qualifiers and values which are further grouped into sets called columns families. A row key is a unique identifier for the table row. Consequently, here the concept of "row" is different in RDB and in HBase and this difference must be understood. Figure 4.8 shows a possible example of the table *"Doctors"* after guideline No. 2 was applied.

==========
Doctors
==========

RowKey: Doctor_Id	Perso: Name	Perso: Age	Perso: Sex	Perso: BornIn	Dpt: Dpt_Id
01	James Smith	43	M	MON	02
02	Robert Brown	39	M	TOR	03
03	John Martin	34	M	VC	02
04	Michael Tremblay	51	M	MON	01

Is possible to have versions in the future

Figure 4.8 Table Doctors with example of information

Guideline No. 3: *Decide if table row design will apply "Tall-Narrow" or "Flat-Wide" approach.*

At this point in the migration process (Dimiduk & Khurana, 2013; Lars, 2011) recommend evaluating the tables obtained in order to consider if those tables should be re-designed based on *"Tall-Narrow"* or *"Flat-Wide"* approaches.

HBase's performance is directly linked to the row key design and how that row key accesses the information. From guideline No. 2 it was learned that HBase will split up data by column family. A *"Tall-Narrow"* approach is a design with more rows and less family columns/columns, whereas the *"Flat-Wide"* approach will create tables with less rows and more family columns/columns. Therefore, with less attributes per unique row, a *"Tall-Narrow"* approach would need to have a more complex row key (it can be referred as a composite key) giving adjacency of similar elements and allowing for *"scans"* by logical group of entries.

On the other hand, a *"Flat-Wide"* approach would have much more information in the entry itself, since only it will have a logical unique attribute as a row key, and it will *"get"* the entry through the row key, and the entry would have sufficient information to process the user request.

Unfortunately, the *Hospital-Doctor* RDB schema is too small to be affected by the advantages or disadvantages between these design approaches. However, and only for the purposes of

better understanding, we will consider Figure 4.8 more closely. First of all, creating a table following a *"Flat-Wide"* approach is possible thanks to the HBase ability to easily add columns, to the scale of millions, at run time (see section 2.2.8). If the design intention is to remove the *Hospital* table, and for the sake of keeping the data, the information in *Hospital* must be added to the *Doctors* table as one family column: *Hospital*, where each added column will represent a specific hospital where the doctor works (meaning work-1, work-2, etc., see Figure 4.9). Adding more hospitals for a specific doctor will not increase the number of rows, it will increase the number of columns, so this table will become a *wide* table. This approach is recommended if the schema will not store a huge amount of information per row (millions of columns). The advantage is to have all entity information in one row. Figure 4.10 shows the results of solving the same problem using the "Tall-Narrow" approach.

As stated earlier, the *"Tall-Narrow"* approach would need to have a more complex row key. For this specific case the row key will be composed of the id itself and the hospital name, in a lexicographic order. Adding more hospitals for a specific doctor will not increase the columns, it will increase the rows, so this table will become a *tall* table. All the information gathered up to now suggests that, given the row key has the highest cardinality, it would be advisable that the most needed parts of the user query go to the row key.

==========
Doctors
==========

RowKey: Doctor_Id	Perso: Name	Perso: Age	Perso: Sex	Perso: BornIn	Dpt: Dpt_Id	Hospital: WorkIn-1	Hospital: WorkIn-2	Hospital: ...	Hospital: WorkIn-N
01	James Smith	43	M	MON	02	St. Mary's	Jewish General		
02	Robert Brown	39	M	TOR	03	Jewish General			
03	John Martin	34	M	VC	02	St. Mary's			
04	Michael Trembl ay	51	M	MON	01	Mount-Sinai	Shriners		

Figure 4.9 Table Doctors created with "Flat-Wide" approach

Despite the fact that the table in Figure 4.8 has a simple row key, the *"Tall-Narrow"* approach will be used in later sections for the *Hospital-Doctor* RDB schema.

==========
Doctors
==========

RowKey: Doctor_Id	Perso: Name	Perso: Age	Perso: Sex	Perso: BornIn	Dpt: Dpt_Id
01#Jewish General	James Smith	43	M	MON	02
01#St. Mary's	James Smith	43	M	MON	02
02#Jewish General	Robert Brown	39	M	TOR	03
03#St. Mary's	John Martin	34	M	VC	02
04#Mount-Sinai	Michael Tremblay	51	M	MON	01
04#Shriners	Michael Tremblay	51	M	MON	01

Figure 4.10 Table Doctors created with "Tall-Narrow" approach

Relational Aspect "Fields": As you can see in Figure 4.3, a field in a RDB schema is the smallest unit of a RDB table which holds the value of a specific attribute in the form of a tuple (row, column), while a cell in HBase is the smallest unit of a HBase table which holds a piece of data in the form of a tuple (row, column, version). The difference between the two is the former has a unique and specific value while the latter, from the same tuple (row, column), could have several versions of the value differentiated by timestamp. This relational aspect will be covered, in an easy way, in the HBase target from the *Hospital-Doctor* RDB schema.

Relational Aspect "Constraints": The following list will show how the HBase solution will generally handle the different constraints types:

Not Null Constraint: HBase does not have to worry about this constraint since it cannot be implemented, the null value does not take up any space in storage in HBase. Moreover, the *Hospital-Doctor* RDB schema has no Not-Null constraint.

Default and Check Constraints: This features specifies either default value for a column when none is provided or ensures that all values in a column satisfy certain conditions. In any case, HBase uses the java package "org.apache.hadoop.hbase.constraint" in order to

accomplish this task. This means they must be coded specifically from the HBase shell interface. The *Hospital-Doctor* RDB schema has neither default nor check constraints.

Primary Key Constraint: While in RDB the primary key is the column (or set of columns) whose values uniquely identify the row, in HBase that function is accomplished by the row key. For instance, the primary key (PK) of the table *Hospital* is the Id, which means two *Hospitals* cannot have the same Id (see Figure 4.1).

Guideline No. 4: *Design the row key according the business rules and/or access patterns.*

In HBase tables, the row key design is a critical step in order to access the information in an efficient way. Unfortunately the "multiple indexes" feature from a RDB schema does not exist in HBase, which only has the ability to provide a single index, the row key, which is closely linked to the performance of read operations. There are two different ways to design a row key:

- Using a unique value as a differentiator, which is used with the *"Flat-Wide"* approach.
- Using a composite value as a differentiator, which is used with the *"Tall-Narrow"* approach.

In turn, the composite value for the row key could be a combination of categorical data, which means a concatenation of several attributes already presented in the RDB schema.

Also, it could be a combination of categorical data and time-series data, where the timestamp is rounded down to the nearest scale and could be a day or an hour or another point depending on the application business rules, and finally concatenated from the rounding process to the row key as a prefix/suffix. Some key points should be taken into account using this kind of composite row key. Firstly, do not forget that the "timestamp is rounded", so the remainder of the information must be stored as a column in the table to avoid losing data. Secondly, it is important to be sure to store data items together whose timestamps (rounded) belong to a given period of time. In that way, the query response time will be increased.

The composite value for the row key could also contain spatial data, e.g. latitude and longitude coordinates. This kind of composite row key will need the help of an external algorithm that provides the right translation from the given pair, latitude and longitude.

The results of using this concept to migrate the *Hospital-Doctor* RDB schema are shown in Figure 4.11. The highlighted areas were updated.

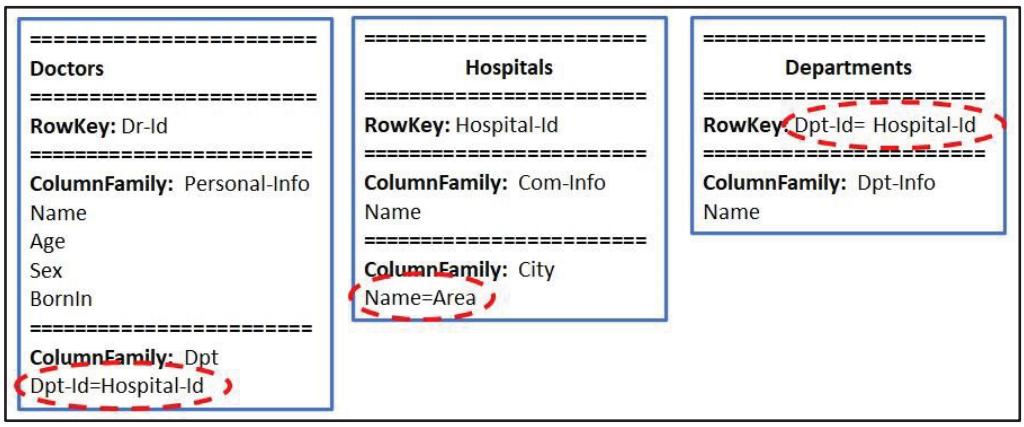


Figure 4.11 Tables after applying guideslines No.2 and No.3

An example of the information that could be stored in the tables is shown in Figure 4.12.

Doctors

RowKey: Doctor_Id	Personal: Name	Personal: Age	Personal: Sex	Personal: BornIn	Dpt: Dpt_Id=Hospital_Id
01	James Smith	43	M	MON	02=01
02	Robert Brown	39	M	TOR	03=04
03	John Martin	34	M	VC	02=03
04	Michael Tremblay	51	M	MON	01=01

Hospitals

RowKey: Doctor_Id	Personal:Name	City: Name=Area
01	Jewish General Hospital	MON-Côte-Des-Neiges
02	St. Mary's Hospital	MON-Côte-Des-Neiges
03	Shriners Hospital for Children of Canada	MON-Notre-Dame-de-Grâce
04	Mount-Sinai Hospital	MON-Côte Saint-Luc

Figure 4.12 Example of information in tables Doctors and Hospitals

==========
Departments
==========

RowKey: Dpt_Id	Dpt-Info: Name
01=01	Financial
02=01	Intensive care unit
02=03	Intensive care unit
03=04	Medical Records
04=02	Pharmacy

Figure 4.13 Table Departments

Finally, concerning Relationships and Referential Integrity Constraint (Foreign Keys): In an RDB, the relationships uniquely identifies a row in any other table. A specific column value must be a reference to existing rows in another table. This relationship could be specified using either the primary key in the reference table or some other unique constraint.

Guideline No. 5: *De-normalize the relationships.*

Normalization is a principle on RDB schemas that involves dividing a large amount of tables into smaller and less redundant tables, and defining relationships between them that can be accessed by invoking join queries. Unfortunately, HBase does not support joins, instead it uses the DDI approach in order to handle the lack of relationship capabilities.

One-to-One Relationships: As suggested by (Chongxin, 2010), there is no intensive work for the migration of a one-to-one relationship because this has already been done in RDB schema using the foreign key information inside a column in HBase. The reason is the one-to-one relationships are the least frequent relationship type in RDB world.

One-to-many Relationships: There are several possible ways to migrate an RDB one-to-many relationship to a HBase schema. The most widely used strategy to migrate to HBase is to create one HTable, with two column families: one column family to store the first RDB table columns and a second column family to store the second RDB table columns. Again, this is just one among several different ways to migrate this kind of relationship to HBase.

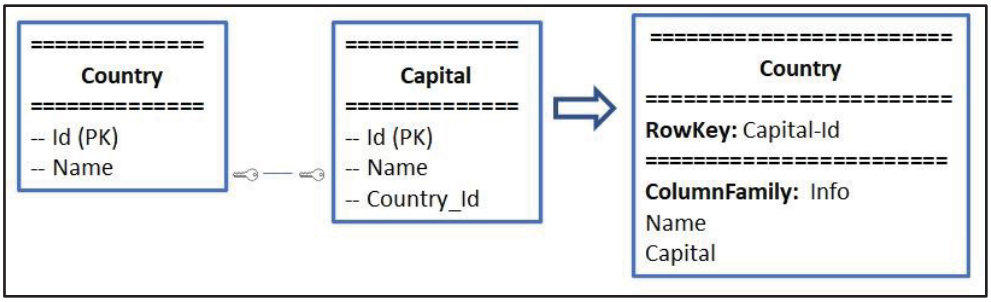
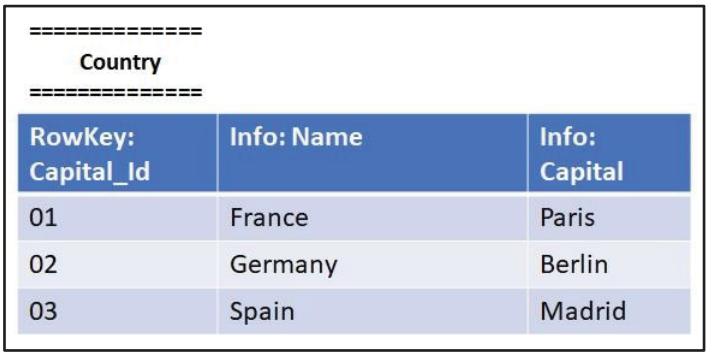


Figure 4.14 One-to-One Relationships on RDB schema to HBase



RowKey: Capital_Id	Info: Name	Info: Capital
01	France	Paris
02	Germany	Berlin
03	Spain	Madrid

Figure 4.15 Migrated tables

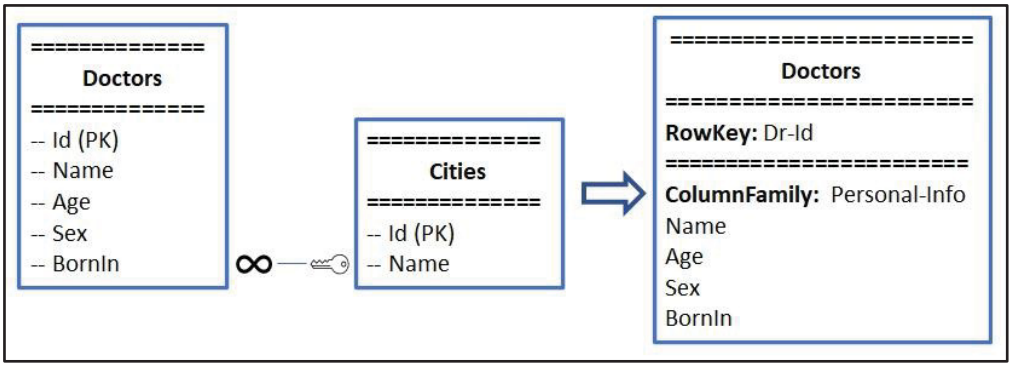


Figure 4.16 One-to-Many Relationships on RDB schema

Many-to-many Relationships: In a many-to-many relationship there is a third table maintains the relationship and keeps the foreign keys for both tables, which means both sides of the relationship are the "many" side. For that reason, migration could be implemented with the same approach as for the "many" side of one-to-many relationship. In other words, new column families should be created in both tables to capture row keys. The third table which is used to

maintain the relationship will be removed, because its information has been expressed on both sides of the relationship.

==========
Doctors
==========

RowKey: Doctor_Id	Personal: Name	Personal: Age	Personal: Sex	Personal: BornIn
01	James Smith	43	M	MON
02	Robert Brown	39	M	TOR
03	John Martin	34	M	VC
04	Michael Tremblay	51	M	MON

Figure 4.17 Example of information in the migrated table

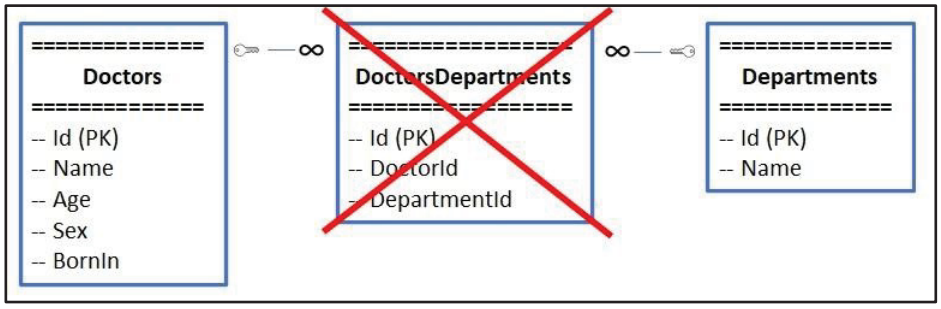


Figure 4.18 Many-to-Many Relationships on RDB schema

==========
Doctors
==========

RowKey: Doctor_Id	Personal: Name	Personal: Age	Personal: Sex	Personal: BornIn	Dpt: Dpt_Id=Hospital_Id
01	James Smith	43	M	MON	02=01
02	Robert Brown	39	M	TOR	03=04
03	John Martin	34	M	VC	02=03
04	Michael Tremblay	51	M	MON	01=01

Figure 4.19 Example of information in the migrated table using DDI

Guideline No. 6:*Merge tables as a way to reduce foreign keys.*

In the RDB world a main table could be used independently in the schema, whereas an attached table is used depending on the referenced object. This guideline states that information on attached tables can be merged into a single row of the main table based on the foreign key and business rules.

Secondary Index Constraint: This constraint cannot be implemented in HBase since it has no native support for secondary indexes. However, there are some specific market solutions that address this problem.

Unique Constraint: In RDB, the unique constraint ensures that all values in a specific column are different. In HBase there are no indexes. The row key, the column families, and the column qualifiers are all stored in sort order (the java comparable method for byte arrays).

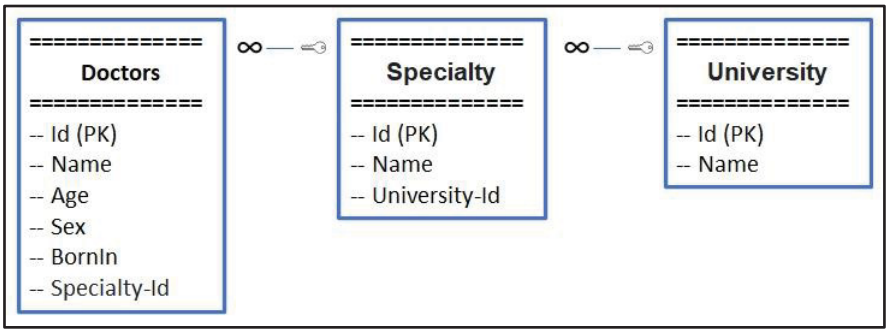


Figure 4.20 Main and attached tables on RDB schema

Doctors

RowKey: Doctor_Id	Personal: Name	Personal: Age	Personal: Sex	Personal: BornIn	Dpt: Dpt_Id=Hospital_Id	Educ: Specialty	Educ: University
01	James Smith	43	M	MON	02=01	Cardiology	Toronto
02	Robert Brown	39	M	TOR	03=04		
03	John Martin	34	M	VC	02=03	Neurosurgery	McGill
04	Michael Tremblay	51	M	MON	01=01		

Figure 4.21 Example of information in the merged tables

Guideline No. 7: *Identify if secondary or unique indexes are needed. If they are, apply the "inverted table"approach.*

An "inverted table" approach is to create a second table where the row key follows the pattern (value, row key) in contrast with the main table where the row key is (row key, value). In others words, the main table will have a row key (could be designed, or not, as DDI) and the second table will "invert" the attribute as the row key and then create a column entry for each row key of the main table. As an example, to include the concept of "managers" where some doctors report to another doctor (main table), we would create a second table (emulating the secondary index) where the row key will be the doctor's name. So for each record in the main table, there would be a corresponding entry in the second table (index table).

If Dr. Tremblay reported to Dr. Smith, and Dr. Brown reported to Dr. Smith, there would be a single row in the *Doctor*'s table, with a row key for Dr. Tremblay and then two column entries containing the row key for Dr. Tremblay's record, and a row key for Dr. Brown's record (see Figures 4.22). Now, to see all of the doctors managed by Dr. Smith, it will be as simple as going to the secondary index table and finding Dr. Smith's row and it will contain the row key for all doctors that report to him. This approach will work only for simple tasks. If you want to try something more complicated there are libraries dedicated to this kind of work (Culvert).

==========
Doctors
==========

RowKey: Doctor_Id	Personal: Name	Personal: Age	Personal: Sex	Manager_Of: C1	Manager_Of: C2
01	James Smith	43	M	02	04
02	Robert Brown	39	M		
03	John Martin	34	M		
04	Michael Tremblay	51	M		

Figure 4.22 Main table (Doctors) in HBase schema

==========
Doctors_Index
==========

RowKey: Doctor_Id	Personal: Id	Manager: C1	Manager: C2
James Smith	01	02	04
Robert Brown	02		
John Martin	03		
Michael Tremblay	04		

Figure 4.23 Second table (index table) in HBase schema

4.3 The guidelines extraction process explained

This section provides an explanation about all the steps conducted with the aim to extract the guidelines from different possible sources, such as, the experiment results analysis presented in chapter three and the literature review explored in chapter two. All those combined, produced a collective knowledge which allowed this research to identify several guidelines that will be explained in detail in the next subsection. To better understand the extraction of these guidelines, it is important to describe the steps performed in the process:

- The first step was to prepare the experiment. As explained in subsection 3.1.1, item 3, the idea was to give the participants, due to time constraints, the possibility to choose a subset of the schema shown in figure 3.3, in order to finalize the experiment as soon as possible. This *"divide and conquer approach"* was expressed in the blue form (synthetic RDB schema) as an information table with the title *working group*. Also, this same approach was used in the survey design, specifically, in question number eight, which took all the concepts of RDB and, for the purposes of this research, they were renamed as *relational aspects*. At the end, this reasoning led to the development of guideline one *"Identify each relational aspects in the source RDB schema and analyze if they can be implemented on HBase and how it could be implemented"*.
- The second step was to apply the experiment in the date and time agreed (see appendix I).

- The third step was the verification and validation of the data provided by the participants in the experiment presented in chapter three. The goal was to ensure all participants filled out and submit all the given forms in a proper way. In case someone took the decision to not submit any documentation, this step also made sure the process was done in the intended formal manner and the documents were also labeled as *not submitted*.
- The fourth step was to collect all the heuristic behind the knowledge process used to achieve the given task in the experiment. Each experiment result sheet was analyzed, it means, the blue (synthetic RDB schema), the green (No-SQL solution) and the yellow one (draft sheets). The analysis started by an organisation by colors, hence, all the participants blue sheet was brought together. Accordingly, the same was done with the green and the yellow. The objective was highlighted common steps conducted during the experiment by the participants, and this step was the base for guidelines number two "*create tables with columns families based on correlated information"* , guideline number four *"design the row key according the business rules and/or access patterns"* and guideline number five *"de-normalize the relationships"*.
- The fifth step was to take a second look and analyze again the proposed solutions on the green sheet (No-SQL solution) and the process expressed on the yellow one (draft sheets) and find any link between the solution shown in the results and the literature review explored in chapter two. As expected, this step was the base, by far, of more technical guidelines, such as guideline number three *"decide if table's row design will apply « tall-narrow » or « flat-wide » approach"*, the guideline six *"merge tables to reduce foreign keys"* and finally the guideline number seven *"Identify if secondary or unique indexes are needed. If so, apply "inverted table" approach"*.

4.4 How the 7 proposed migration steps compare to the current state of the art of RDBMS to No-SQL migration

To further validate our findings, another literature review was conducted on the topic of RDBMS to No-SQL to see if any other publication was made from 2014 to 2022 which had

discovered or improved on these proposed 7 steps. Figure 4.24 confirms that this research work still proposes a more complete set of migration steps than other publications. The only other research from Serrano et al., with 4 of the 7 steps, comes close to our findings of 2014. This is encouraging as our research results are still to be challenged by other researchers.

Migration process literature review	Column family based on correlated information.	Design the Row Key by access patterns.	Denormalize relationships.	Merge tables reduce foreign keys.	Identify each relational aspects.	Row design: tall-narrow/flat-wide.	"inverted table" for 2nd index.
A Comprehensive Spark-Based Layer for Converting RDB to NoSQL - Abdel-Fattah, Wael & Abdelgaber, 2022	✘	✓	✓	✘	✘	✘	✘
Transformation of Schema from RDB to NoSQL DB - Alotaibi & Pardede, 2019	✓	✘	✓	✘	✘	✘	✘
Transf data with metamorfose framework - Kuszera, Peres & Didonet, 2019	✓	✓	✓	✘	✘	✘	✘
Data conversion from RDB to Hbase - Chen & Lee, 2017	✓	✘	✓	✘	✘	✘	✘
Towards an SQL-to-HBase Transf Method - Serrano, Han & Stroulia, 2015	✓	✓	✓	✓	✘	✘	✘
Rules to migrate from RDB to No-SQL DB - Gomez 2014-2017	✓	✓	✓	✓	✓	✓	✓

Figure 4.24 Migration guidelines in the 2014-2022 litterature

A final validation of the resulting guidelines was conducted to understand the "*Level of Coverage of HBase in RDB Aspects*". Table 4.1 shows that this was greatly improved through the use of the seven steps as compared with the results obtained with the experimentation described earlier (see Table 3.4). In Table 4.1, the "*comments*" explain why the percentage was assigned.

We think that the seven steps will offer a much better coverage of the RDB aspects when they will be used in the future.

Table 4.1 Level of Coverage of HBase in RDB Aspects

Relational aspect covered	Percentage of coverage	
	%	*Comments*
Table	100%	
Columns/Attributes	100%	
Rows	100%	
Fields	100%	
Constraints: Not Null	0%	Not supported
Constraints: Default Value	100%	Using java package "org.apache.hadoop.hbase.constraint"
Constraints: Check Constraints	100%	Using java package "org.apache.hadoop.hbase.constraint"
Constraints: Primary Key	100%	Using DDI and access pattern analisys.
Constraints: Referential Integrity		It must create the code to support referential integrity.
Relationships		
--One-to-One Relationships	100%	
--One-to-many Relationships	100%	
--Many-to-many Relationships	100%	
Constraints: Secondary Index	50%	The 50% represents that is possible to emulate them using the "inverted table" approach, but there is still a lot of extra work.
Constraints: Unique Constraint	75%	Only the Row Key offers a unique constraint. Is not possible to have another one.

4.5 Future research

Concerning future research, Figure 4.25 shows the distinction between the overall research objectives and the scope of this research, it means, the TRACK 1 represented by the red border in the figure. The TRACK 2 research will take place in a forthcoming research, conducting another experiment to evaluate how the use of these guidelines would help in migration tasks.

The proposed experimental approach to the overall research is to use the same RDB as the origin and follow the two different experimental tracks to arrive at a target database (a migrated database), and thus, allow a valid comparison between the two resulting databases. The present research addressed only the TRACK 1, which was conducted without the use of migration guidelines, it means, based only on experience (heuristics), and the TRACK 2 will be conducted with the use of the migration guidelines presented in chapter 4. At the end of the overall research, will be possible to compare the two HBase databases obtained.

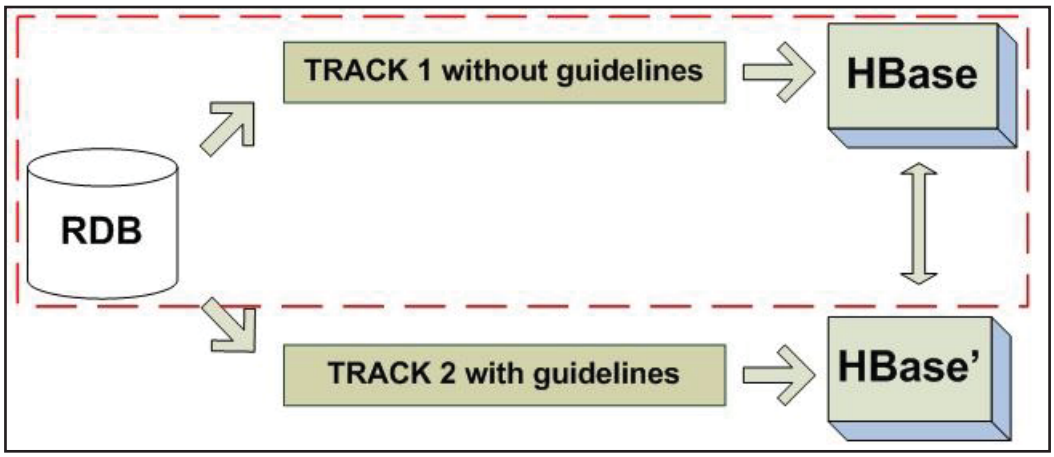


Figure 4.25 The two TRACKs of the research migration from RDBMS to No-SQL

4.5.1 Validation of the proposed guidelines

In this sub-section, it will be explored a way to validate the guidelines proposed in the previous section. The general idea is to check if the use of the guidelines improves the migration process from RDB to No-SQL databases (specifically HBase). Here, the concept *"improve"* will be used in the sense to bring into a more desirable or excellent condition the current migration process from RDB applications to No-SQL database applications.

Figure 4.25 shows, graphically, the whole experimental research objective, please take into account the present research scope only cover the TRACK 1 highlighted by the red dots mark. In order to conduct such validation, it would be necessary, design the validation plan that will outline the objective and the strategy that will be used to endorse the given guidelines. The first step of the validation plan will include an analysis and selection of a suitable RDB

candidate to be migrated. This RDB should be sufficiently complex in terms of relational aspects and data quantity and it will be used the same RDB application to follow the two experimental tracks showed in the figure 4.25: the first, without the use of the guidelines, and the other one with the use of the guidelines.

The second step of the validation plan will be the creation of a relational aspects list containing all the desired relational aspect that would be migrated. Examples of such relational aspects would be, among others:

- Tables
- Columns
- Rows
- Fields
- Constraints:

  Not Null
  Default Value
  Check Constraints
  Primary Key
- Relationships:

  One-to-One Relationships
  One-to-many Relationships
  Many-to-many Relationships

The third step of the validation plan will include a recommendation about how to compare the two resulting DBs showed in figure 4.25: HBase (without guidelines) and HBase' (with guidelines). Many authors have studied different ways to compare database application (Gomez et al., 2015; Goyal, Swaminathan, Pande, & Attar, 2016; Serrano et al., 2015) but the recommended evaluation way in this sub-section implies the coverage evaluation of the two resulting database applications. It means, compare HBase (without guidelines) with RDB source and, also, compare HBase' (with guidelines) with RDB source. As a result, each DB target: HBase (without guidelines) and HBase' (with guidelines) will have a list of the level of coverage of each relational database aspect. Table 4.1 shows a similar list.

The fourth step of the validation plan will be the analysis of each result: HBase (without guidelines) with RDB source and, also, compare HBase' (with guidelines). Here, the evaluation criteria could be the average of percentage of coverage.

Despite the proposed validation plan is based on the comparison of different coverage evaluation, also it could be recommended different parameters, specifications, and acceptance criteria could be recommended, such as content evaluation, data coverage, structure coverage or content verification.

4.6 Why this research is still relevant today?

Nowadays, the present research is still relevant, mostly due the RDB is the database dominant model in the market. Indeed, migration of large RDB to cloud computing databases, specifically No-SQL DB, still poses several significant problems, e.g., the RDB model constraints, the lack of migration experience, the fact that No-SQL DB implies new ways of solutions, the possibility that not all data can be migrated and the shortage of proper teaching about No-SQL DB in academia, among others. (Kumar, Kumar, & Namdeo, 2021; A. M. Maatuk, Abdelaziz, & Ali, 2020; Raouf, Abo-Alian, & Badr, 2021).

Present-day migrations are mostly based on heuristic approach, it means, the entire process depends on whereby experiences from some migration team members. The applications based on RDB should be migrated in a way that could maintain a modern and reliable migration process, it means one that not depends only on experienced team members to know which step should be the next in the process.

The importance of this research lies in the need to provide a first attempt to transform part of the used heuristics into standards, starting with the given guidelines from previous sections. They were extracted based on a combination of heuristics and deep review of preceding attempts to face similar migrations.

The given guidelines are open to interpretation, it means, they are not intended to be applied in a specific order, however, they depict a path to conduct the migration and their application could have an impact in terms of reduce the migration times, for instance, if it takes a lot of

time to decide how to address the multiple relationship issues in RDB, it would takes longer for the RDB to be migrated.

In the chapter two, a deep literature review was conducted to show how different researchers addressed similar problems. The key point found in these researches was that all of them treated each issue in an isolated way. The present research treats each issue as part of the whole rather than a separate entity, it means, each issue found should be a different aspect of the same migration process, and treat all the whole as the beginning of a possible standardization process.

Apply a clear and transparent migration process, will help the software engineers, with no experience in No-SQL technology, in each step and will provide results in a way that could be easily replicated by others. Additionally, it will provide the right solution strategy to face each particular issue during the migration and it that way the migration time could be reduced.

4.7 Conclusion

This chapter has identified, explored, uncovered, and described a series of seven (7) steps that could be used as guidelines for the migration process from RDB to a column-store No-SQL database, specifically the HBase DB. Starting from the demographic experiment results, it was shown that most software engineers, with no experience in No-SQL technology, need a guidance about what steps to conduct during the migration process from RDB to No-SQL DB. Taking the experiment's results as a base, and using the literature review analyzed in chapter two, it was possible to uncover and extract seven guidelines. Also, to help in the understanding of each guideline, a simple RDB example was used to explain how each of these relational aspects should be migrated, and, how the concept of relational aspects was covered by each guideline.

Another point addressed in this chapter was the relation between the literature review in chapter two, the experiment shown in chapter three and the guidelines proposed in this chapter. Finally, the future of this research was examined, in conjunction with a way to validate the proposed guidelines.

CONCLUSION

The problem of migrating an existing RDB towards a column-store No-SQL database, specifically HBase, was studied. As presented in chapter 3, a heuristic approach is used for conducting an RDB to No-SQL migration based on the experience of the participants. This 2013 case-study has helped in identifying, exploring, uncovering, and describing seven (7) migration steps that could act as a set of guidelines to help future software engineers in their first attempt at conducting the migration from RDB to column-store No-SQL database.

These guidelines were identified after analyzing the detailed responses of the participants as well as the database migration literature. The participants had industrial and academic backgrounds, some were undergraduates, others graduates with a Master or PhD degree, but most had strong relational database experience and almost no experience with No-SQL databases (which reflects the current state of affairs in the industry). The experiment aimed at asking the participants to migrate a simple database schema from RDB to a column-store No-SQL database (HBase) without the use of any guidelines.

The participants filled out a questionnaire after the experiment which was used to better understand their migration approach and the difficulties encountered. This information was used to ensure that specific RDB aspects be well covered during a migration. Finally the seven (7) steps published have shown that they offer a better coverage of the RDB aspects and are still current in 2022.

Main contribution and outcomes

A list of seven (7) migration steps were identified, explored, uncovered, and described.

The assistance provided by these guidelines is still to be tested but, in our opinion, offer much better coverage of RDB aspects than any other proposed approach, even in 2022. The research results related to the guideline formulation and the experimentation were published in:

- Toward building RDB to HBase conversion rules

R. Ouanouki, Abraham Gomez A. April, A. Abran & J. M. Desharnais

Journal of Big Data volume 4, Article number: 10 (2017)
doi:10.1186/s40537-017-0071-x

- Building an Experiment Baseline in Migration Process from SQL Databases to Column Oriented No-SQL Databases
Abraham Gomez, R Ouanouki, A. April, A. Abran
Journal of Information Technology & Software Engineering (2014)
doi: 10.4172/2165-7866.1000137

- Gomez, Abraham, Ouanouki, R., Ravanello, A., April, A. and Abran, A.
Experimental Validation as Support in the Migration from SQL Databases to NoSQL Databases
Conference on Cloud Computing, GRIDs, and Virtualization. (2015).
doi: 978-1-61208-388-9

Limitations

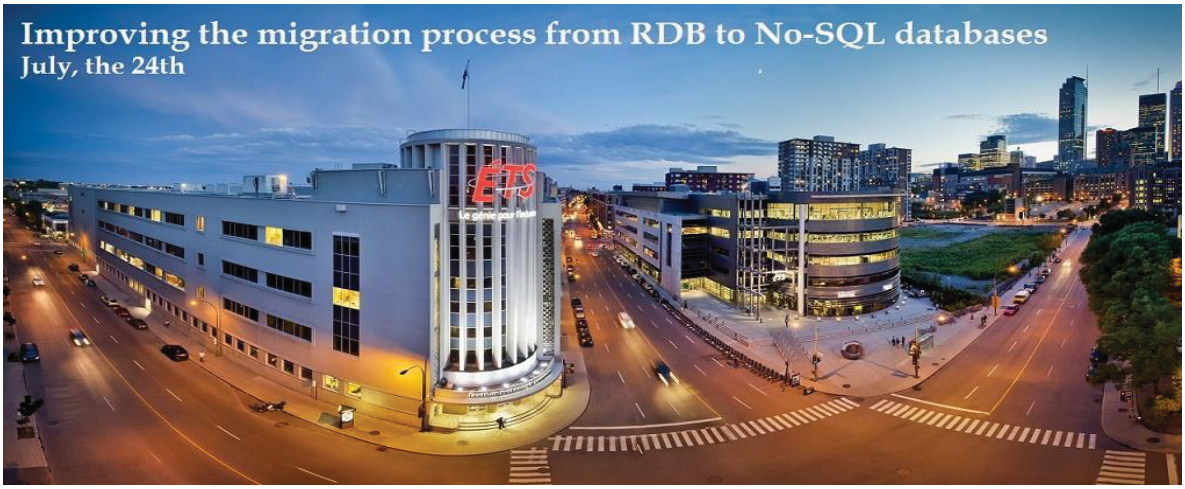
The findings of this study have some limitations:

- The experiment sample size (twenty participants) may make it difficult to determine if a particular outcome of the experiment can be extrapolated in a valid way for all populations of developers and whether the result can be generalized.
- The lack of prior contributions about specific guidelines make it difficult to compare the guidelines provided with any prior study;
- As a consequence of the preceding limitation, more experimentation is required (in TRACK 2) in order to validate and potentielly improve the proposed guidelines, evaluate the performance of the guidelines, add new ones or analyze if they are still valid for complex relational database schemas;
- The limitation regarding the domain. In other words, the guidelines were designed to cover a particular kind of No-SQL databases, the column-oriented database, and a particular example of these databases, HBase. So, if the HBase technology evolves, and this evolution

affects some of the guidelines, these should be updated. Likewise, more research would be needed to evaluate if the guidelines can be generalized for other kinds of No-SQL databases.

APPENDIX I

EXPERIMENT 1 – CALL FOR PARTICIPATION



Call for Participation

The École de Technologie Supérieure (ÉTS) invites you to participate in the project "Rules for Converting a Relational Database to a No-SQL Database for Cloud Computing". The objective is to verify the guidelines developed by researchers to improve the migration process from relational applications to No-SQL applications.

The session will be held at ÉTS, Montreal, on July 24th and 31st, 2013 and it will be 90 minutes in length.

The session will comprise three defined parts:

- Training in relational and No-SQL database aspects.
- The experiment itself with the support of the appropriate material.
- Finally, it will have a survey, in order to evaluate the experience.

We especially encourage those participants with experience in programming and databases design and administration; their participation will contribute to enhance the migration process to applications in this new complex environment.

Participants with previous experience will collaborate with the formalization of the migration process contributing to the final solution.

Participants without previous experience will obtain valuable information about how to conduct a SQL to No-SQL migrations process.

APPENDIX II

EXPERIMENT 1 –GENERAL INSTRUCTIONS

Instructions to follow during the session:

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

Please do not communicate with other participants during the session.

The experiment will have four parts: Introduction, training session, experiment, and survey.

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

2. The training session: Listen carefully to 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 relational DB, No-SQL database and an example of the migration process (at schema's level).

3. 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 designing your response schema on the green sheet, please make sure that your “participant code” is written on all the documents used in the experiment.

Also, return all the documents used and not used in the experimentation to the organizer in the yellow envelope.

4. 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.

APPENDIX III

EXPERIMENT 1 – 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 a 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)	

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

5. Before starting the migration process, what was your first step?

Question	Answer
Did you try migrating 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 it 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 not 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 Relational aspect covered	0%	25%	50%	75%	100%
Table					
Constraint					
PK					
FK					
Other:					
Other:					

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

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

APPENDIX IV

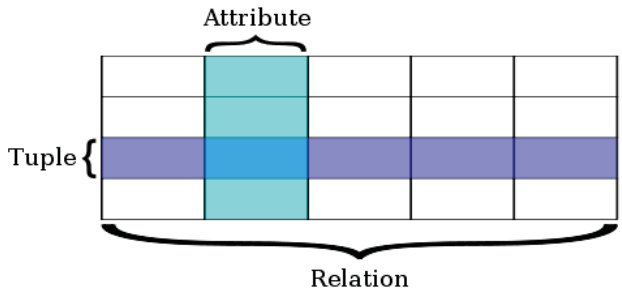
EXPERIMENT 1 –TRAINING DOCUMENT

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 organized 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 relations 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 is a figure of this model:



Relational Aspects

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 which uniquely identifies it (for instance, in a table of information about students, the student ID). Another common occurrence, especially in regards to N:M cardinality is the composite key. A composite key is a key made up of two or more attributes within a table that (together) uniquely identify a record.

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

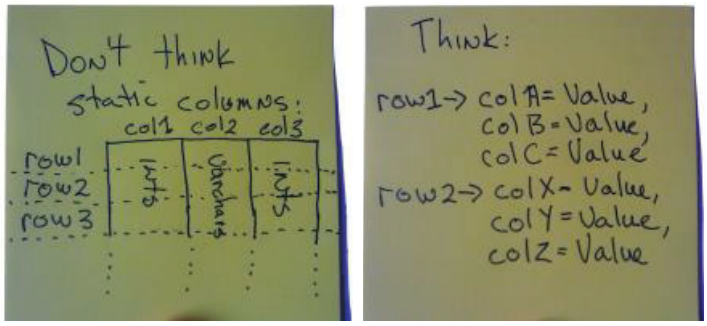
The relational databases have row-oriented storage (they are organized by rows):

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 column family have the same prefix. For example, the columns info:name and info:age are both members of the info column family. The colon character (:) delimits the column family. All the columns of the same family are recorded in the same region server.



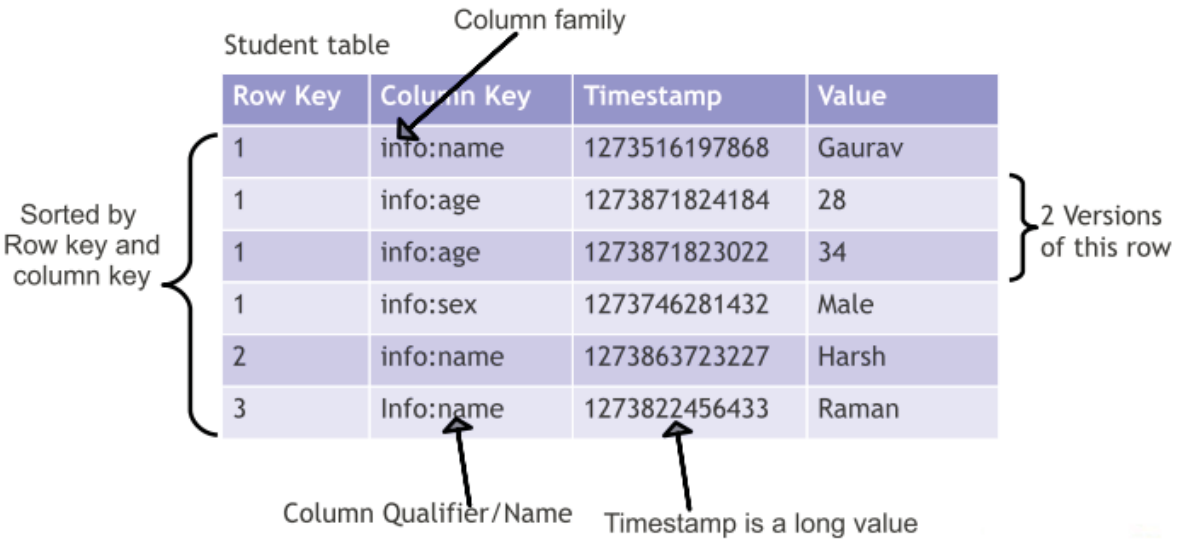
How does HBase work?

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 hosts the tables; performs the reads, manage the buffers writes. Also the clients can talk directly to them for reads/writes.

HBase schema design

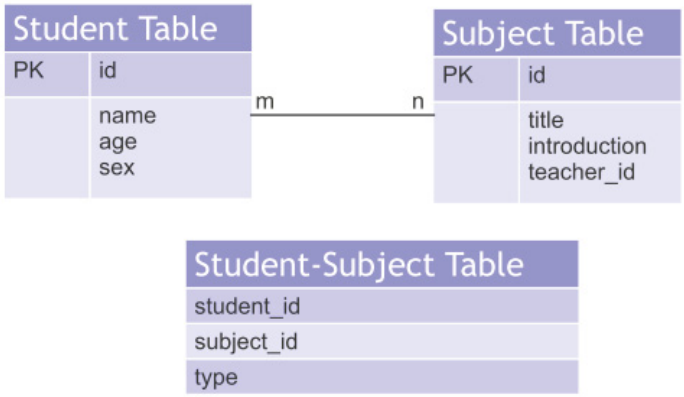
HBase is a big sorted map and to obtain a cell value, you have to enter the Row Key+ Column Key + timestamp.



Migration Use Cases

Relational and No-SQL are two different worlds, to obtain a good enough migration you must implement at least the idea of denormalize and duplicate data and build a good row 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"

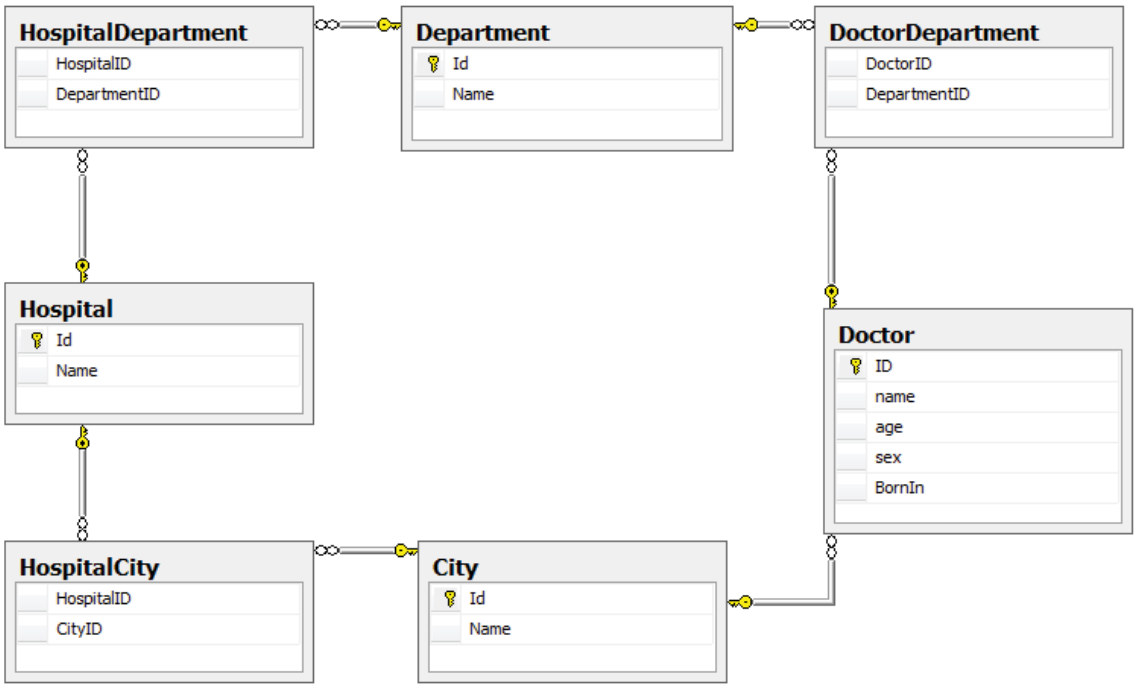
Subject table

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

APPENDIX V

EXPERIMENT 1 –BLUE DOCUMENT

Participants Code : _____ **Synthetic relational schema**



Working groups	
Id	**Tables**
A	Hospital – HospitalDepartment – Deparment
B	Doctor – DoctorDepartment – Deparment
C	Hospital – HospitalCity – City
D	Doctor – City
E	All Tables

APPENDIX VI

EXPERIMENT 1 –GREEN DOCUMENT TEMPLATE

Participants Code : _____ **No-SQL solution**

Please apply the document “White Training Document” and for each working group that you will consider in the “blue synthetic relational schema” write:

Relational Schema that you will migrate

Working Group:

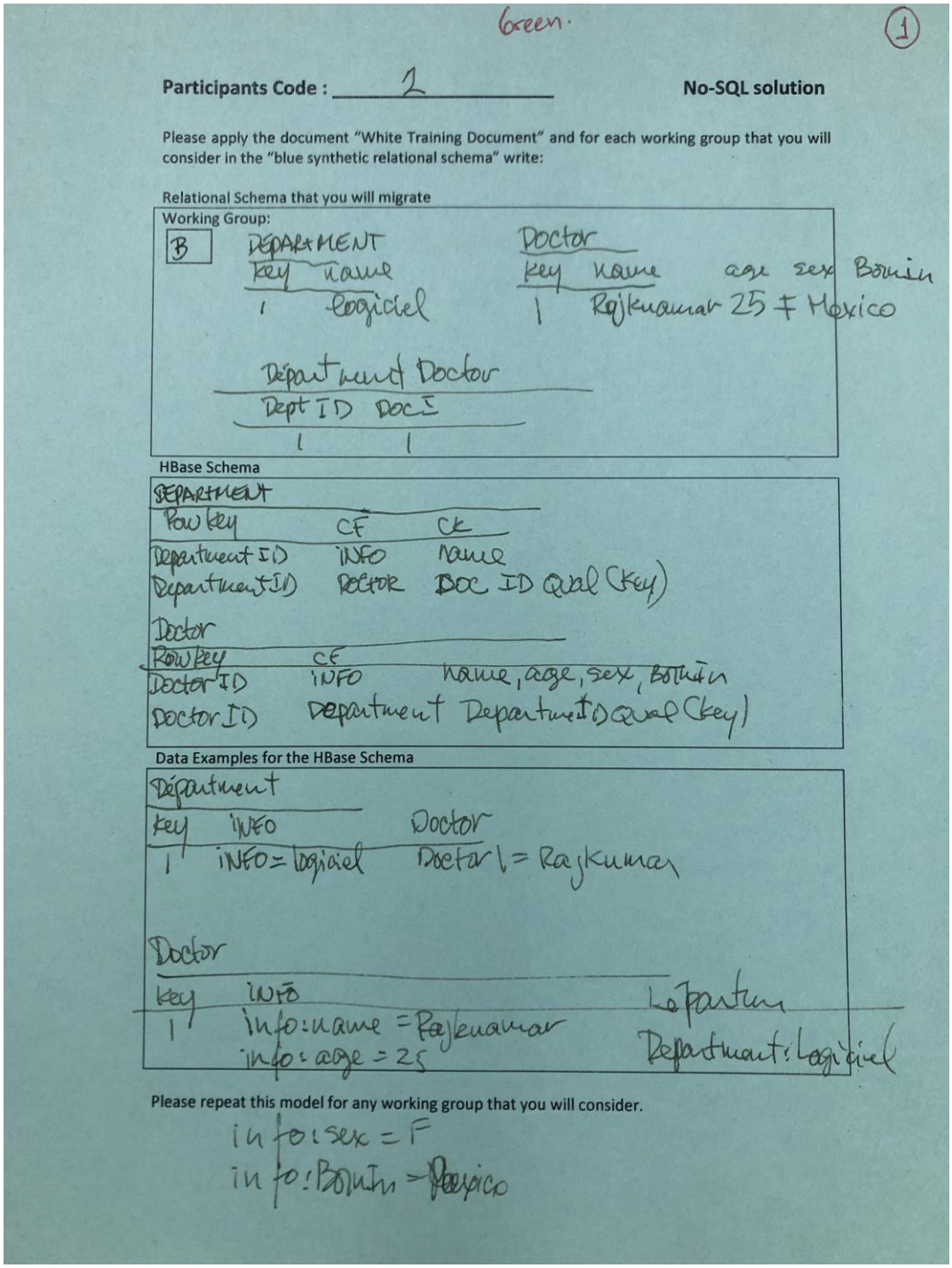
HBase Schema

Data Examples for the HBase Schema

Please repeat this model for any working group that you will consider

APPENDIX VII

EXPERIMENT 1 –GREEN DOCUMENT SAMPLES



green. (1)

Participants Code : 2 **No-SQL solution**

Please apply the document "White Training Document" and for each working group that you will consider in the "blue synthetic relational schema" write:

Relational Schema that you will migrate

Working Group: B

DEPARTMENT
Key name
1 Logiciel

Doctor
Key name age sex Bornin
1 Rajkumar 25 F Mexico

Department Doctor
Dept ID Doc ID
1 1

HBase Schema

DEPARTMENT
RowKey CF CK
DepartmentID INFO name
DepartmentID Doctor Doc ID Qual (key)

Doctor
RowKey CF
DoctorID INFO name, age, sex, Bornin
DoctorID Department DepartmentID Qual (key)

Data Examples for the HBase Schema

Department
Key INFO Doctor
1 INFO = logiciel Doctor 1 = Rajkumar

Doctor
Key INFO
1 info:name = Rajkumar
info:age = 25
info:sex = F
info:Bornin = Mexico

Department:Logiciel

Please repeat this model for any working group that you will consider.

(2)

Participants Code : 2 **No-SQL solution**

Relational Schema that you will migrate

Working Group:

9

Doctor TABLE

Key	Name	age	sex	Boston
1	Rajkumar	28	M	Montreal

City TABLE

Key	Name
1	Rajkumar

HBase Schema

voir p. jaune

Data Examples for the HBase Schema

voir p. jaune

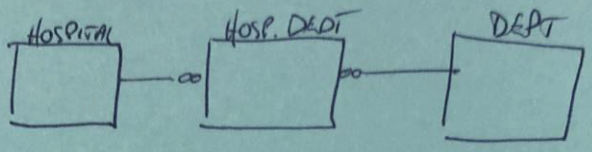
Green. (1)

Participants Code : 3 **No-SQL solution**

Relational Schema that you will migrate

Working Group:

A



HBase Schema

ROW KEY	COL FAM
HOSPITAL_ID	HOSP_NAME
HOSPITAL_ID	~~HOSPITAL~~ DEPT_NAME

Data Examples for the HBase Schema

KEY	~~HOSPT~~ NAME	DEPT_NAME
1	STE- JUSTINE	RADIOGRAPHIE
2	CHUM	PHYSIOT...

(2)

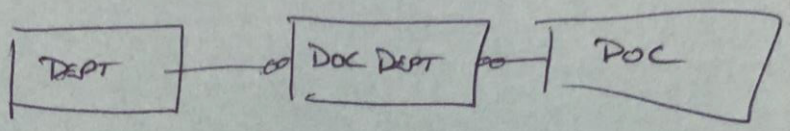
Participants Code : 3 **No-SQL solution**

Please apply the document "White Training Document" and for each working group that you will consider in the "blue synthetic relational schema" write:

Relational Schema that you will migrate

Working Group:

B



HBase Schema

Row key	Col Fam.
Doc ID	DEPT_ID DEPT_NAME

Data Examples for the HBase Schema

KEY	DEPT ID	DEPT NAME
1	21	RADIOLOGIE
1	31	PHYSIO...

Please repeat this model for any working group that you will consider.

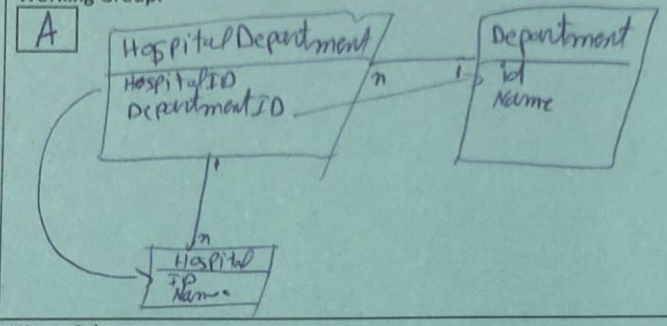
Green. (1)

Participants Code : 4 **No-SQL solution**

Relational Schema that you will migrate

Working Group:

A



HBase Schema

Row key	family name	Colum key
Hospital-ID	~~Info~~ Department	~~Name~~ DepartmentID
Department-ID	~~Info~~ Info	~~HospitalID~~ Name
Hospital-ID	~~Info~~ Info	Name

Data Examples for the HBase Schema

Hospital Row key	~~Family name~~	Table
1	Department: A	Hospital Department
1	Info: name	Department
1	Info: name	Hospital

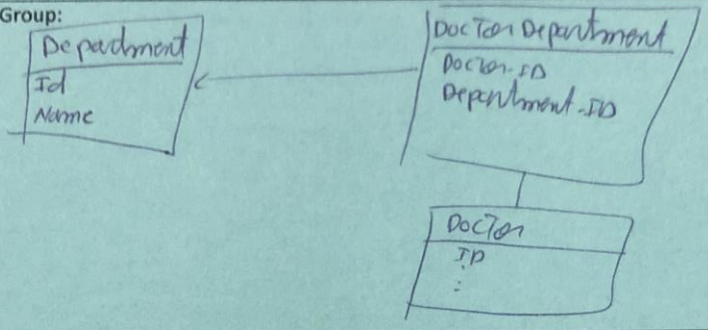
(2)

Participants Code : 4 **No-SQL solution**

Please apply the document "White Training Document" and for each working group that you will consider in the "blue synthetic relational schema" write:

Relational Schema that you will migrate

Working Group: B



HBase Schema

Department Table

Row Key	Column family	Column Key
Department_ID	Info	name
Department_ID	Doctor	DoctorID

Doctor Table

Row Key	Column family	Column Key
DoctorID	Info	name, age, sex, RoomIn
DoctorID	Department	DepartmentID

Data Examples for the HBase Schema

Department Table

~~Row~~ Key	Info	Doctor
1	Info: ~~R~~dward	Doctor: pierre.

Doctor Table.

Key	Info	~~Doctor~~ Department
1	pierre, 30, M, Paris	1

Please repeat this model for any working group that you will consider.

Green. (1)

Participants Code : 5 **No-SQL solution**

Please apply the document "White Training Document" and for each working group that you will consider in the "blue synthetic relational schema" write:

Relational Schema that you will migrate

Working Group: A

Hospital Table

Row Key	Column family	Column Key
Hospital Id	info	Name
Hospital Id	Hdepartments	Department Id
Hospital Id		

Department Table

Department Id	info	Name
Department Id	Hospital	Hospital Id

HBase Schema

Data Examples for the HBase Schema

Hospital table

	info	Hdepartment
1	info: Name = Hopital Saint-Laurent	Hdepartment: 1 = plasturgie

Department table

	info	
1	info: Name plasturgie	Hospital: 1 = Hopital Saint-Laurent.

Please repeat this model for any working group that you will consider.

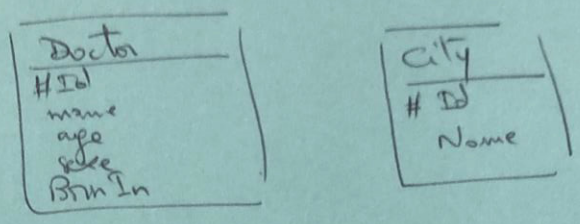
(2)

Participants Code : 5 **No-SQL solution**

Relational Schema that you will migrate

Working Group:

B



HBase Schema

Doctor table

doctorId	info	name, age, sex, bornIn
doctorId	city	city_id

city table

cityId	info	Name
cityId	doctors	doctor_Id

Data Examples for the HBase Schema

docta table:

	info	city
1	info: name = PHILL info: age = 45 info: sex = M info: BirthIn = Nevada	city:1: Dublin

city table:

	info	doctors
1	info: Name = Dublin	doctors:1: PHILL doctors:2: Mat Brien

1

Participants Code : 6 **No-SQL solution**

Please apply the document "White Training Document" and for each working group that you will consider in the "blue synthetic relational schema" write:

Relational Schema that you will migrate

Working Group:

D

HBase Schema

Doctor

Row Key	Col F	Col Key
doctor-id	Info	name, age, sex
City		
city-id	city	name

Data Examples for the HBase Schema

Key	Info	City
1	info: name = Bob info: age = 45 info: sex = M	city: 1
1	city city: name = Valdor	

Please repeat this model for any working group that you will consider.

green. (2)

Participants Code : 6 **No-SQL solution**

Relational Schema that you will migrate

Working Group:
15

HBase Schema

Doctor Table

Row K	Col F	Col Key
doc-id	Info	name, age, sex,
doc-id	dep	dep-id
doc-id	city	city-id

Departement Table

R K	Co F	Col Key
dep-id	Dep	name
dep-id	dep-doc	doc-id

Data Examples for the HBase Schema

dep Table

K	Info	dep
1	info. name: Bob	dep: 1=
	info. age : 45	
	info. sex : M	

dep Table

K	Dep	Doc
1	dep: name = nursing	doc: 1
		doc: 2

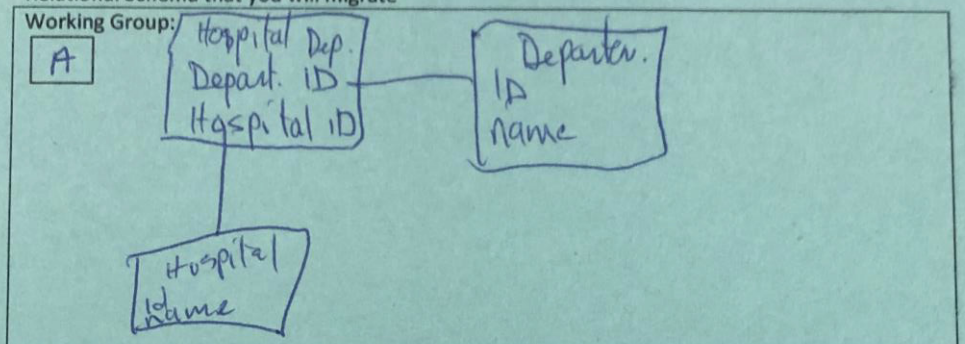
Green. (1)

Participants Code : 7 **No-SQL solution**

Please apply the document "White Training Document" and for each working group that you will consider in the "blue synthetic relational schema" write:

Relational Schema that you will migrate

Working Group: A



HBase Schema

Row Key	Column family	Colums Keys
Hospital id	info	name
Hospital id	Departement	Depart. ID as qualifier
Depart id	info	name
Depart id	Hospital	Hospital ID as qualifier

Data Examples for the HBase Schema

Hospital Table

Key	info	departement
1	info:name = Hospital 1	departement:1 = "Depart 1"

Departement Table

1	info: name = Depart 1	Hospital:1 = "Hospital 1"

Please repeat this model for any working group that you will consider.

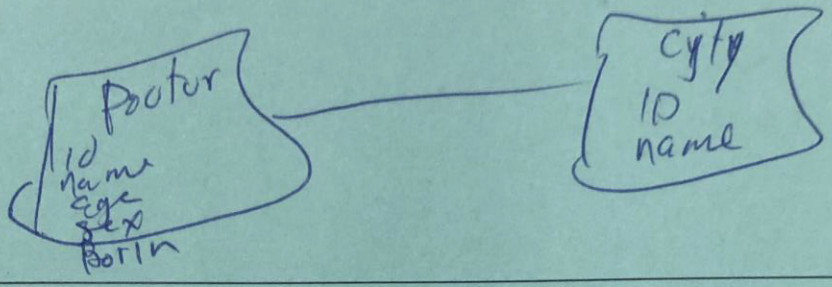
(2)

Participants Code : _____ **No-SQL solution**

Relational Schema that you will migrate

Working Group:

D



HBase Schema

Row Key	Column family	Columns Key
Doctor_ID	info	id, name, age, sex, born
city_ID	info	name

Data Examples for the HBase Schema

1

info: name = ...
info: age = ...
info: sex = ...
info: Born = ...

city = 1

Green. (1)

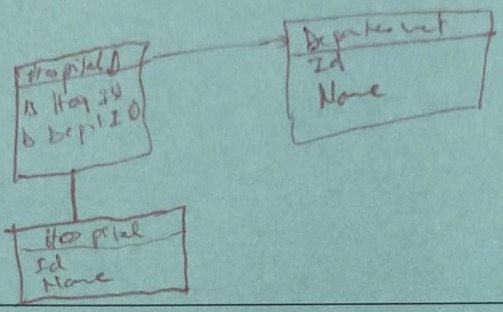
Participants Code : 08 **No-SQL solution**

Please apply the document "White Training Document" and for each working group that you will consider in the "blue synthetic relational schema" write:

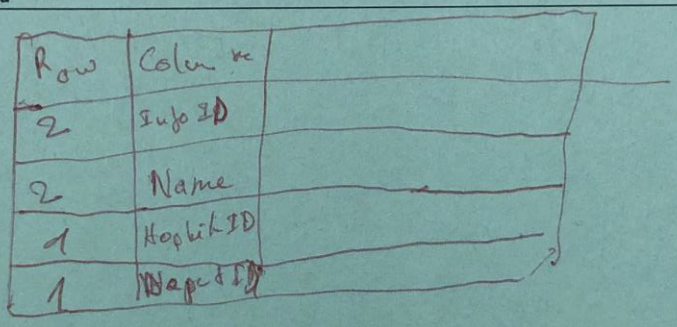
Relational Schema that you will migrate

Working Group:

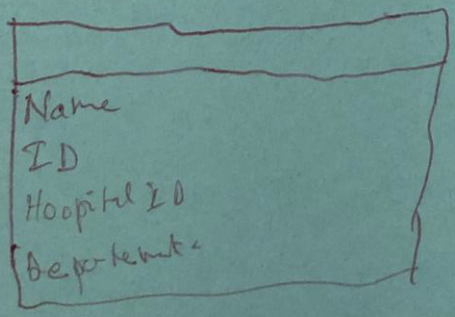
A



HBase Schema



Data Examples for the HBase Schema



Please repeat this model for any working group that you will consider.

Green. (1)

Participants Code : 9 **No-SQL solution**

Please apply the document "White Training Document" and for each working group that you will consider in the "blue synthetic relational schema" write:

Relational Schema that you will migrate

Working Group:

A Hospital - Hospital Department - Department

Hospital

ID	NAME
1	Maisonneuve - Rosemont

Department

ID	NAME
1	ORTHOPEDIE

Hospital Department

1	1

HBase Schema

Hopital

ROW KEY	COLUMN FAMILY	COLUMN KEYS
HospitalID	INFO	NAME
HospitalID	Dept	DepartmentID

Department

ROW KEY	COLUMN FAMILY	COLUMN KEYS
DepartmentFD	INFO	NAME
DepartmentID	Hospital	HospitalID

Data Examples for the HBase Schema

Hospital

key	INFO	Dept
1	INFO:NAME=Maisonneuve Rosemont	Dept:1="Orthopedie

Department

	INFO	Hospital
1	INFO:NAME=Orthopedie	Hospital:1=Maisan Rosem

Please repeat this model for any working group that you will consider.

Green (1)

Participants Code : 10 **No-SQL solution**

Please apply the document "White Training Document" and for each working group that you will consider in the "blue synthetic relational schema" write:

Relational Schema that you will migrate

Working Group:

D

Doctor Table

ID	NAME	AGE	SEX	BORN IN
1	PAUL	45	M	1
2	DIANA	40	F	2

City Table

ID	NAME
1	LAVAL
2	MONTREAL

Where come from Table

Doctor-id
City-id

HBase Schema

Doctor Table
Row key
Doctor-id
Doctor-id

Column Family
INFO
cities

Column key
NAME, SEX
city id AS qualifier key

City Table
Row key
city-id
city-id

Column Family
INFO
wherecomes

Column key
NAME
city id AS qualifier key

Data Examples for the HBase Schema

Doctor Table

key
1

INFO
INFO:NAME = PAUL
INFO: AGE = 45
INFO. SEX = M

cities

City Table

key
1

INFO
INFO:NAME = LAVAL

where comes

Please repeat this model for any working group that you will consider.

Green. (1)

Participants Code : 11 **No-SQL solution**

Please apply the document "White Training Document" and for each working group that you will consider in the "blue synthetic relational schema" write:

Relational Schema that you will migrate

Working Group:

D

Doctor table

Key	name	age	sex	born_in
1	Gaurav	28	male	1001H

City table

id	name
1	HBase

Doctor - City table

doctor_id	city-id	type
1	1	live

HBase Schema

Doctor table

Row Key	Column family	Column Keys
doctor-id	info	name, age, sex, born_in
doctor-id	~~subjects~~ cities	city Id's as qualifier (Key)

city table

Row Key	Column family	Column Keys
city-id	info	name
city-id	~~subjects~~ doctors	Doctor Id's as qualifier (Key)

Data Examples for the HBase Schema

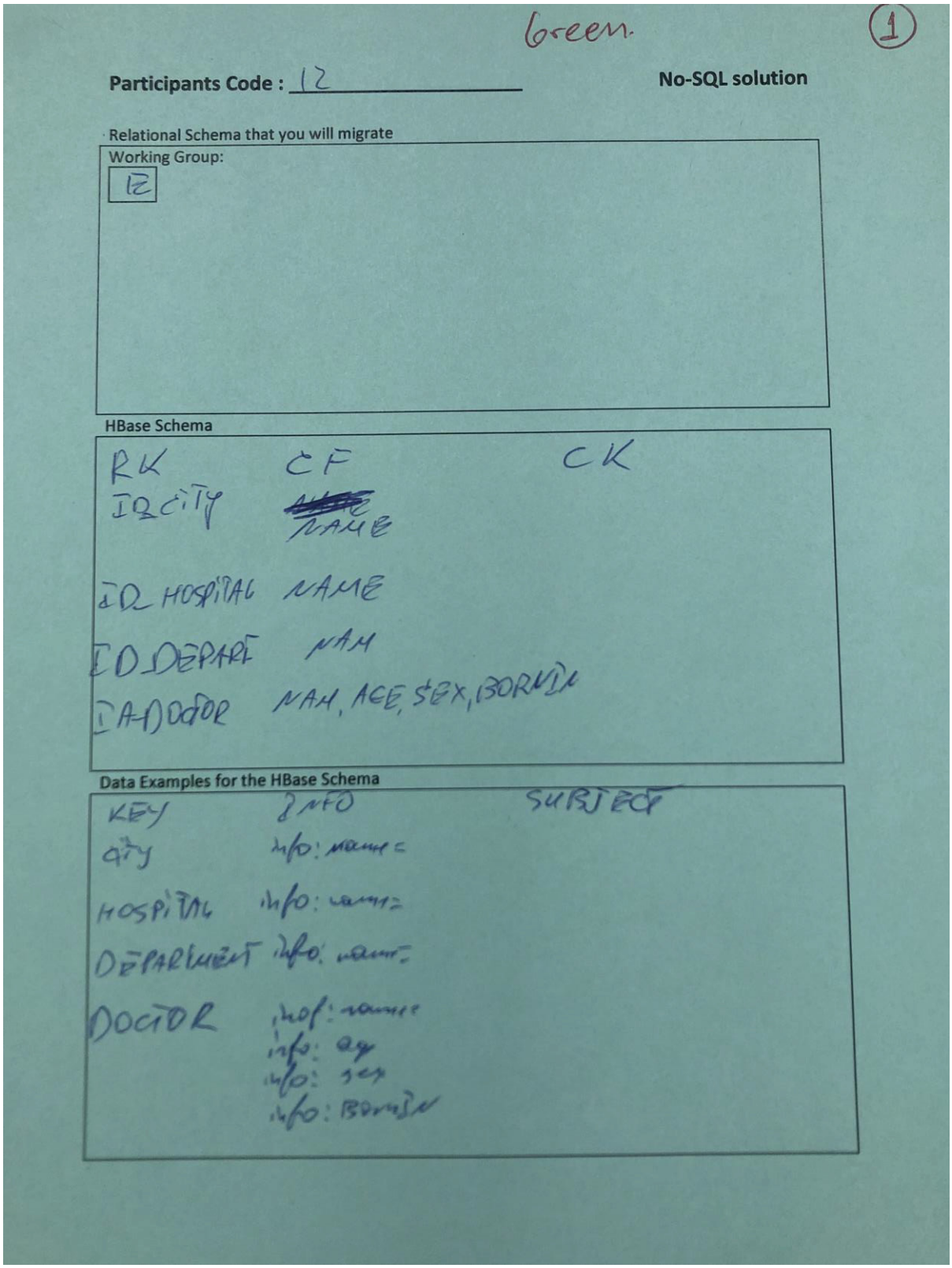
Doctor table

Key	Info	~~Subjects~~ Cities
1	info:name = Gaurav info:age = 28 info:sex = MALE info:born_in = 1001H	cities:1 = "live" cities:2 = "main"

City table

Key	Info	Doctors
1	info:name = HBase ~~info:~~	doctors:1 doctors:2

Please repeat this model for any working group that you will consider.



green. (1)

Participants Code : 12 **No-SQL solution**

Relational Schema that you will migrate

Working Group:

12

HBase Schema

RK CF CK

IDCITY ~~NAME~~ NAME

ID HOSPITAL NAME

ID DEPART NAM

ID DOCTOR NAM, AGE, SEX, BORN

Data Examples for the HBase Schema

KEY INFO SUBJECT

city info: name=

HOSPITAL info: name=

DEPARTMENT info: name=

DOCTOR info: name=
info: age
info: sex
info: Born

Green.

(1)

Participants Code : 13 **No-SQL solution**

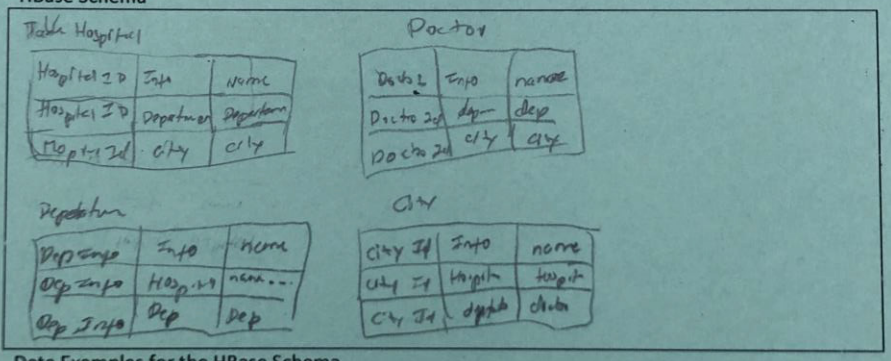
Please apply the document "White Training Document" and for each working group that you will consider in the "blue synthetic relational schema" write:

Relational Schema that you will migrate

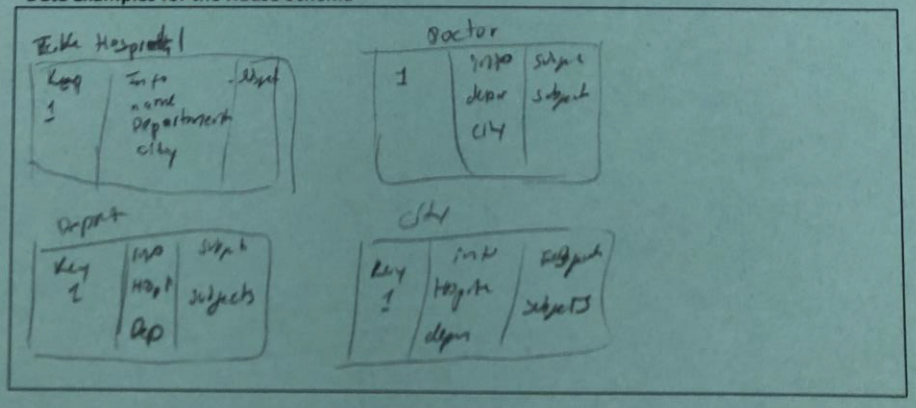
Working Group:

☐

HBase Schema



Data Examples for the HBase Schema



Please repeat this model for any working group that you will consider.

Participants Code : ~~12~~ 14 Green. (1)

No-SQL solution

Please apply the document "White Training Document" and for each working group that you will consider in the "blue synthetic relational schema" write:

Relational Schema that you will migrate

Working Group: D

Doctor Table	
PK	id
	name
	age
	sex
FK	bornIn

City Table	
PK	id
	city

HBase Schema

Row Key	Column Family	Column Keys
doctor_id	info	name, age, sex
doctor_id	Locations	bornIn (Key)

Row Key	Column Family	Column Keys
Location_id	info	city

Data Examples for the HBase Schema

Key	info	Locations
1	info:name = Abraham info:age = 30 info:sex = male	~~info:bornIn = "Montreal"~~ Locations:1 = "Montreal"

Key	info
1	info:city = Montreal

Please repeat this model for any working group that you will consider.

Green

(1)

Participants Code : 15 **No-SQL solution**

Please apply the document "White Training Document" and for each working group that you will consider in the "blue synthetic relational schema" write:

Relational Schema that you will migrate

Working Group:

C Hospital - HospitalCity - City

HBase Schema

Hospital

Rowkey	ColumnFamily	ColumnKeys
HospitalId	info	Name
HospitalId	cities	cityid

City

Rowkey	columnfamily	ColumnKeys
cityId	info	Name
cityId	Hospitals	HospitalId

Data Examples for the HBase Schema

Hospital

Key	Info	cities
1	info:name = Central	cities:1 = Montreal

City

Key	Info	Hospitals
1	info:name = Montreal	Hospitals:1 = Central

Please repeat this model for any working group that you will consider.

(2)

Participants Code : 15 **No-SQL solution**

Relational Schema that you will migrate

Working Group:

D Doctor - city

HBase Schema

Doctor Table

Rowkey	ColumnFamily	ColumnKeys
DoctorId	Info	name, age, sex
DoctorId	Cities	cityId

city Table

Rowkey	ColumnFamily	ColumnKeys
cityId	Info	name
cityId	Doctors	DoctorId

Data Examples for the HBase Schema

Doctor

Key	Info	Cities
1	info: name = Pierre info: age = 40 info: sex = M	cities: 1 = Montreal

City

Key	INFO	Doctors
1	info: name = Montreal	Doctors: 1 = Pierre

(1)

Participants Code : 16 green.

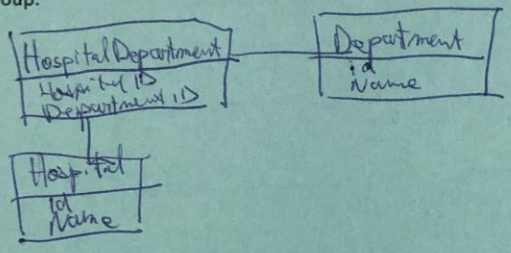
No-SQL solution

Please apply the document "White Training Document" and for each working group that you will consider in the "blue synthetic relational schema" write:

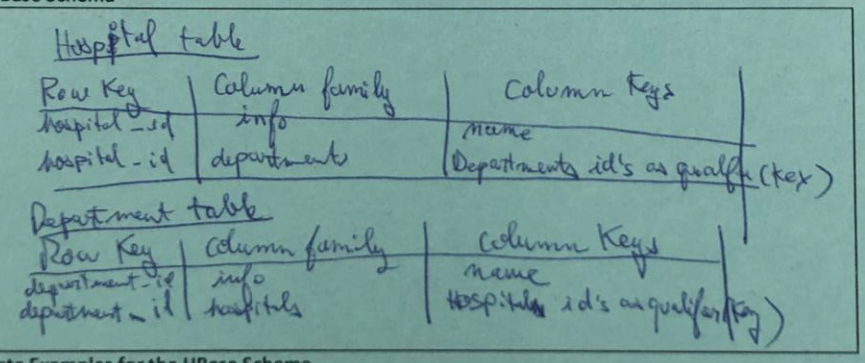
Relational Schema that you will migrate

Working Group:

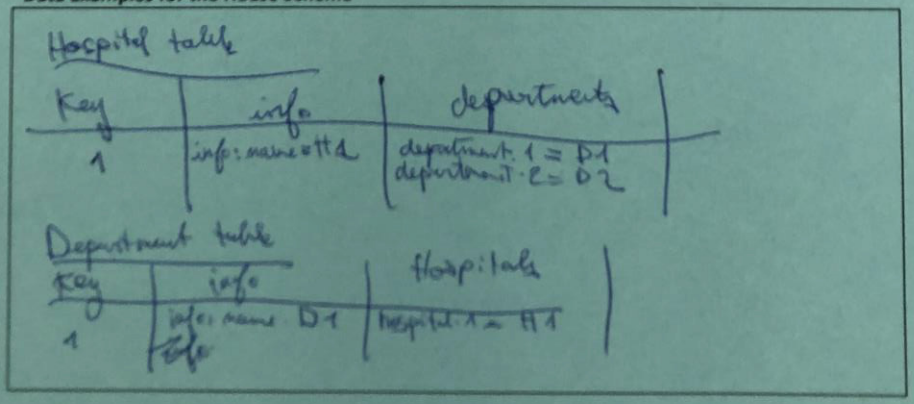
A



HBase Schema



Data Examples for the HBase Schema



Please repeat this model for any working group that you will consider.

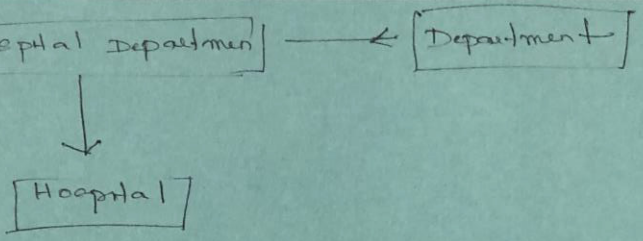
green. (1)

Participants Code : 18 **No-SQL solution**

Relational Schema that you will migrate

Working Group:

A

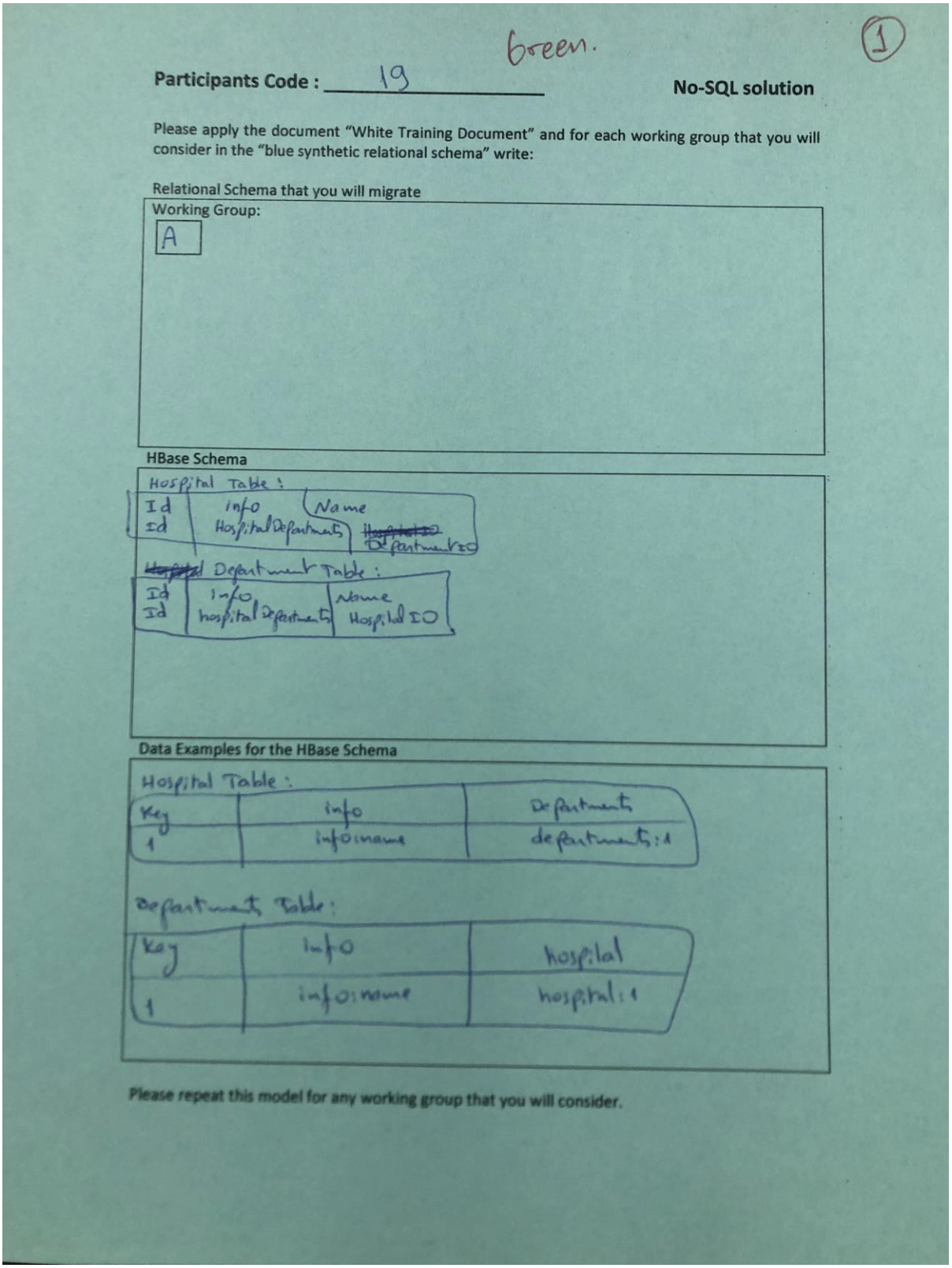


HBase Schema

Row key	Column family	Column key/s
Hospital_id	~~Hospital Department~~	—
Dept_id	~~Department~~	—

Data Examples for the HBase Schema

key	info	Column key/s
Department_Id		Department
1.	info: Dept name info: Dept location	
Hospital_Id		Hospital
1.	info: Hospital name info: Hospital location	



green.

(1)

Participants Code : 19 **No-SQL solution**

Please apply the document "White Training Document" and for each working group that you will consider in the "blue synthetic relational schema" write:

Relational Schema that you will migrate

Working Group:

A

HBase Schema

Hospital Table :

Id	info	Name
Id	Hospital Departments	~~Hospital ID~~ Department ID

~~Hospital~~ Department Table :

Id	Info	Name
Id	hospital Departments	Hospital ID

Data Examples for the HBase Schema

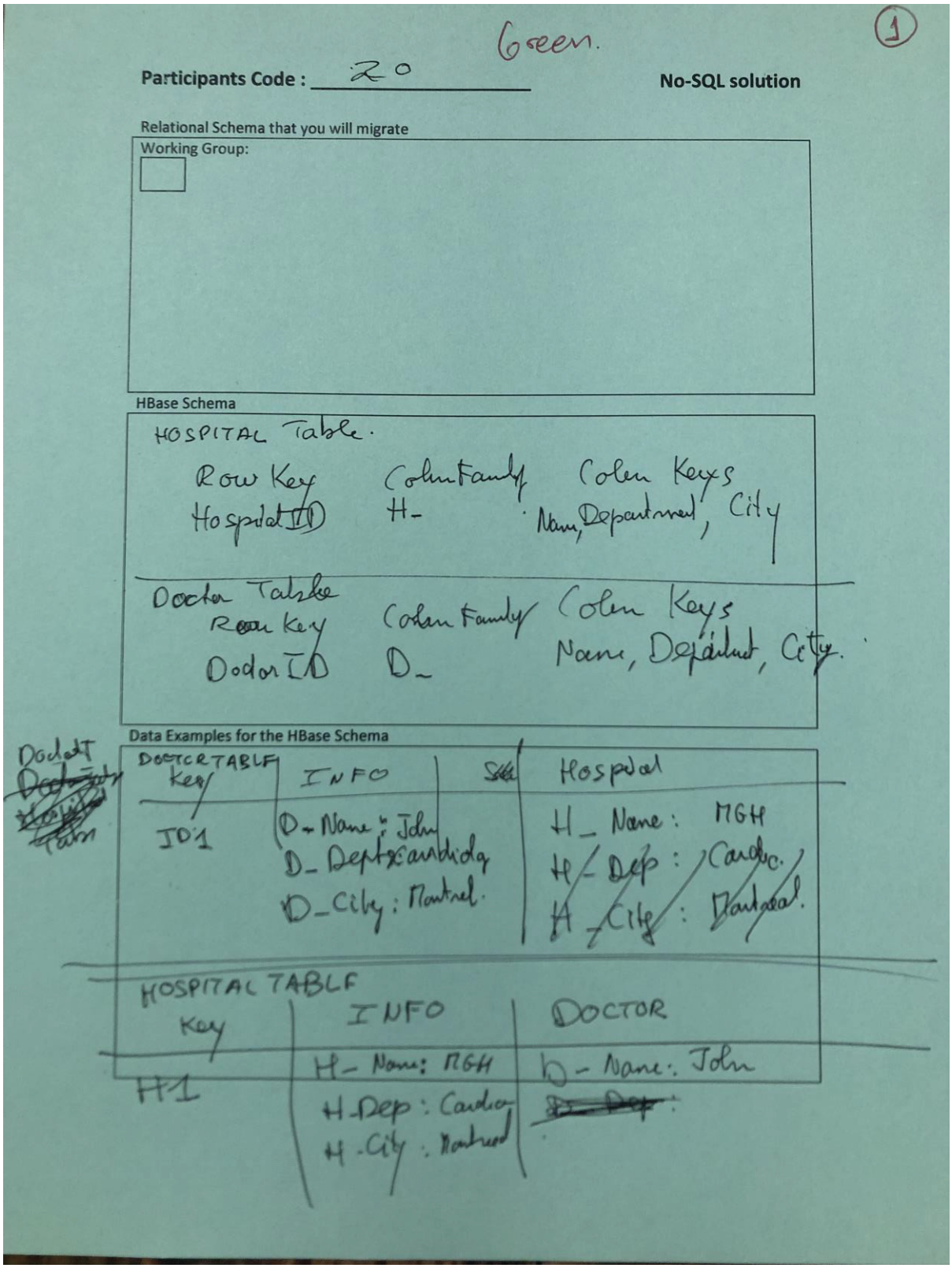
Hospital Table :

Key	info	Departments
1	info:name	departments:1

department Table :

Key	info	hospital
1	info:name	hospital:1

Please repeat this model for any working group that you will consider.



Green.

(1)

Participants Code : R0 **No-SQL solution**

Relational Schema that you will migrate

Working Group:

HBase Schema

HOSPITAL Table.

Row Key	Colun Family	Colen Keys
Hospital ID	H-	Name, Department, City

Doctor Tablke

Row key	Colen Family	Colen Keys
Dodor ID	D_	Name, Depatment, City.

Data Examples for the HBase Schema

DOCTOR TABLE Key	INFO	Hospital
ID1	D- Name : John D_ Dept : Cardiology D_City : Montreal.	H_ Name : MGH H- Dep : Cardio. H_City : Montreal.

HOSPITAL TABLE Key	INFO	DOCTOR
H1	H- Name: MGH H-Dep : Cardio H-City : Montreal	D- Name: John ~~D- Dep :~~

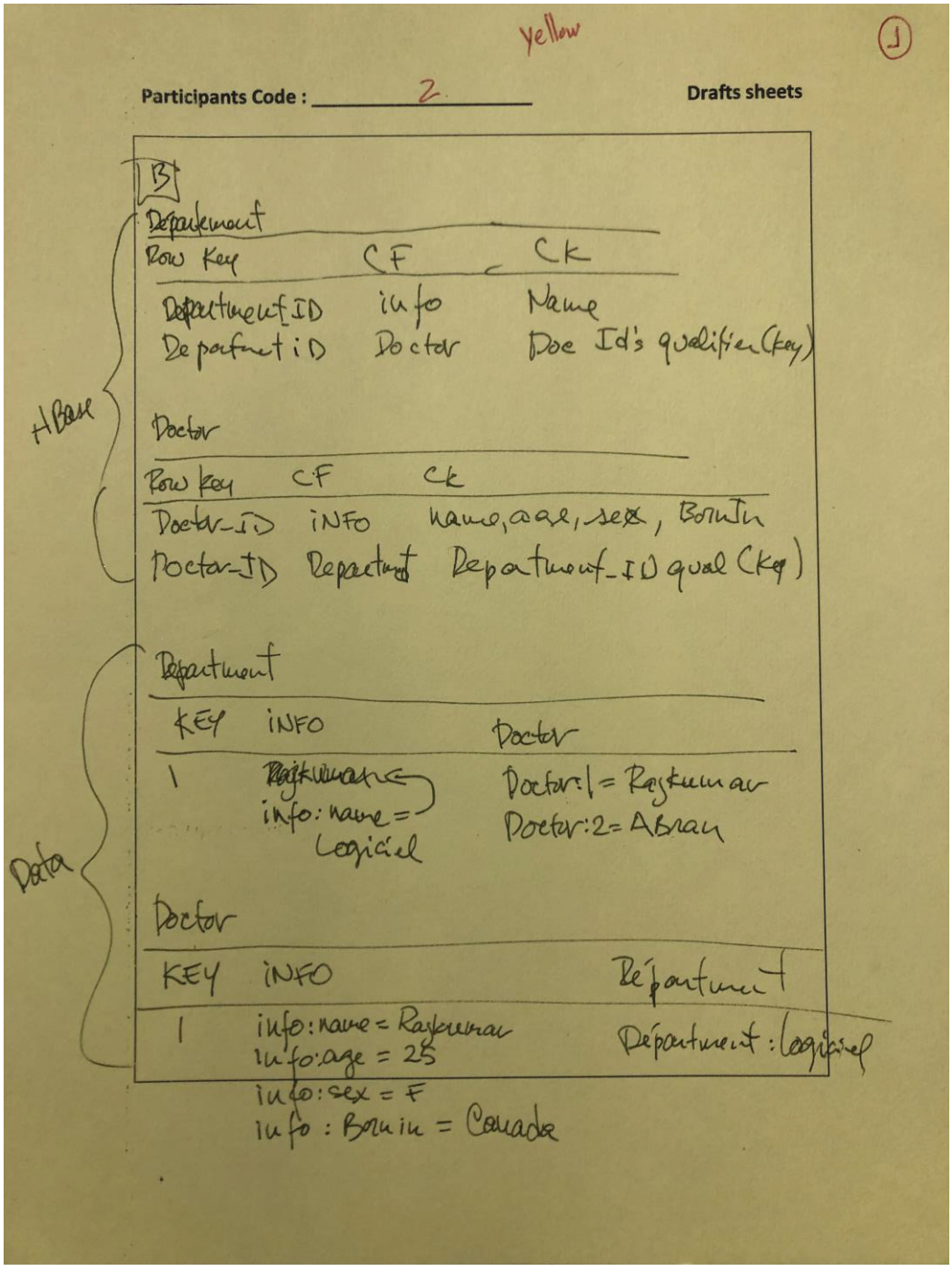
APPENDIX VIII

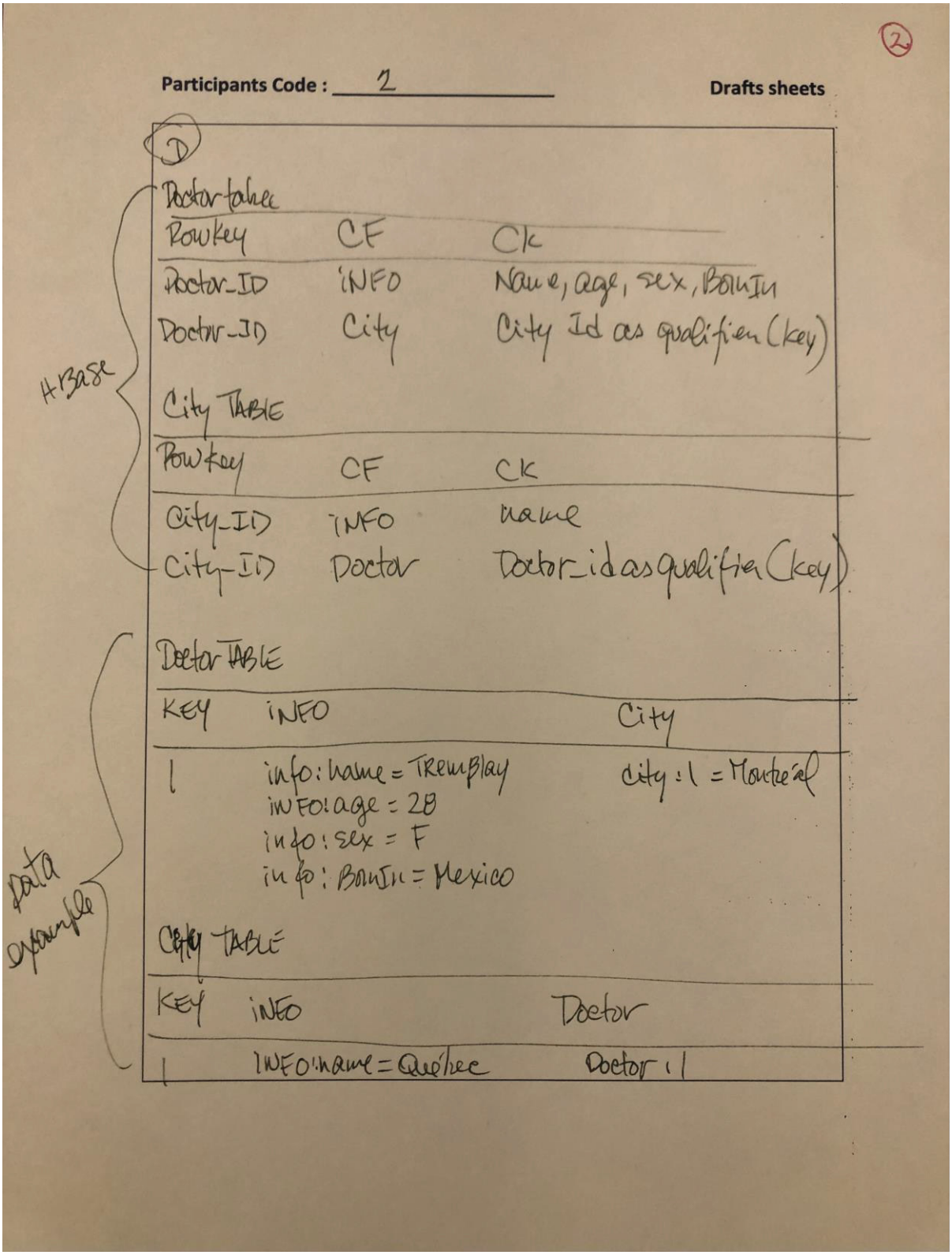
EXPERIMENT 1 –YELLOW DOCUMENT TEMPLATE

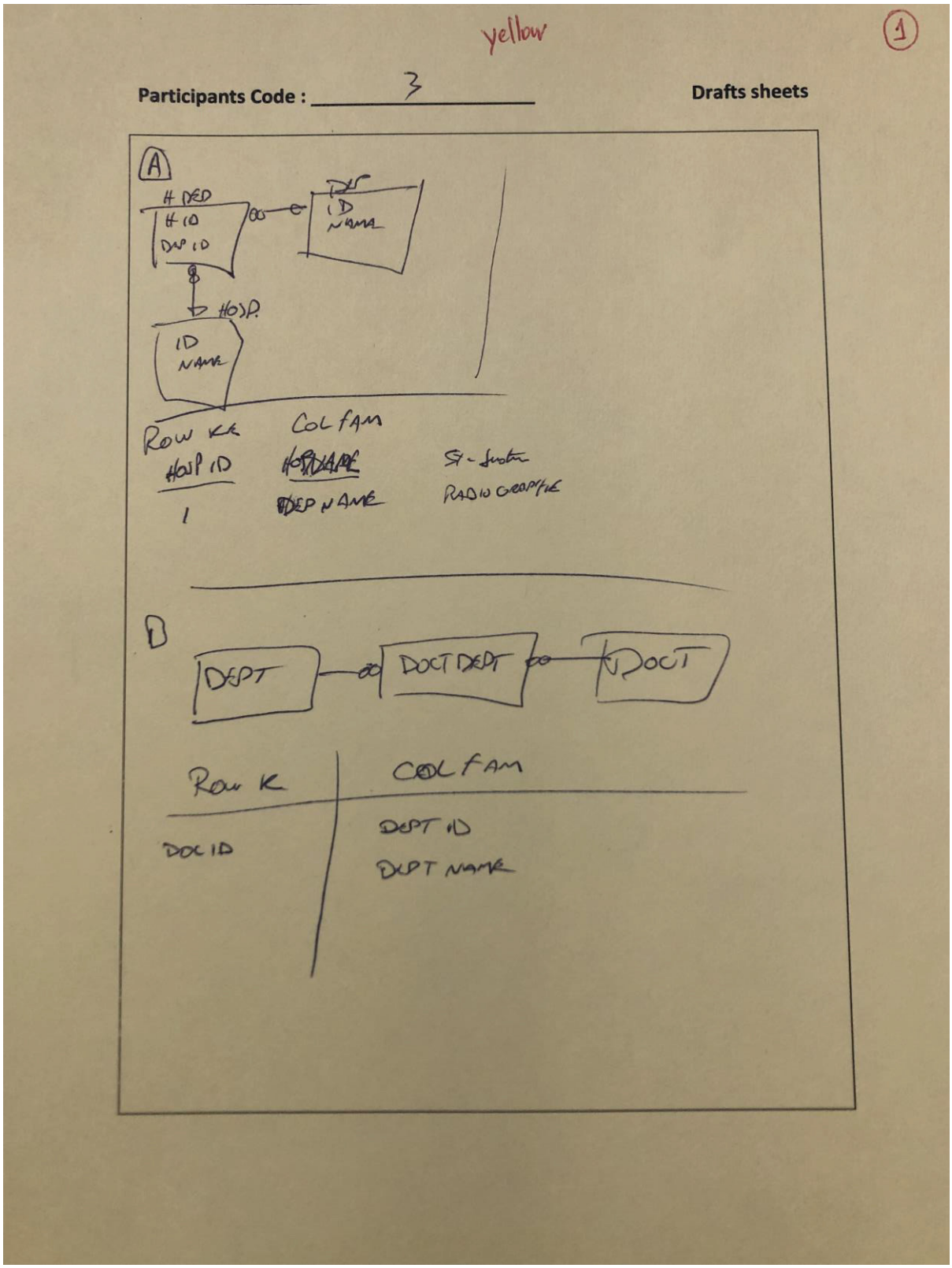
Participants Code : _____ **Drafts sheets**

APPENDIX IX

EXPERIMENT 1 –YELLOW DOCUMENT SAMPLES




Participants Code : 1
Drafts sheets
2
1
Doctor table
Rowkey
CF
Ck
Doctor_ID
INFO
Name, age, sex, BOIRTH
Doctor_ID
City
City Id as qualifier (key)
HBase
City TABLE
Rowkey
CF
Ck
City_ID
INFO
name
City_ID
Doctor
Doctor_id as qualifier (key)
Doctor TABLE
KEY
INFO
City
1
info: name = Tremblay
INFO: age = 28
info: sex = F
info: BOIRTH = Mexico
city: 1 = Montreal
data example
City TABLE
KEY
INFO
Doctor
1
INFO: name = Québec
Doctor: 1


yellow
1
Participants Code : 3
Drafts sheets
A
DEP
H ID
DEP ID
HOSP
ID
NAME
ID
NAME
ROW KE
HOSP ID
1
COL FAM
DEP NAME
ST - Static
RADIO GRAPHIC
B
DEPT
DOCT DEPT
DOCT
ROW K
COL FAM
DOC ID
DEPT ID
DEPT NAME

Yellow ①

Participants Code : 4 **Drafts sheets**

Relational

PK → Row key

attributes → Column key family 1
" " " 2
" " " 3

Department

Row	family	Col ~~entity~~ key
Dep-ID	Info	name

Doctor Department

~~Doctor~~

Department

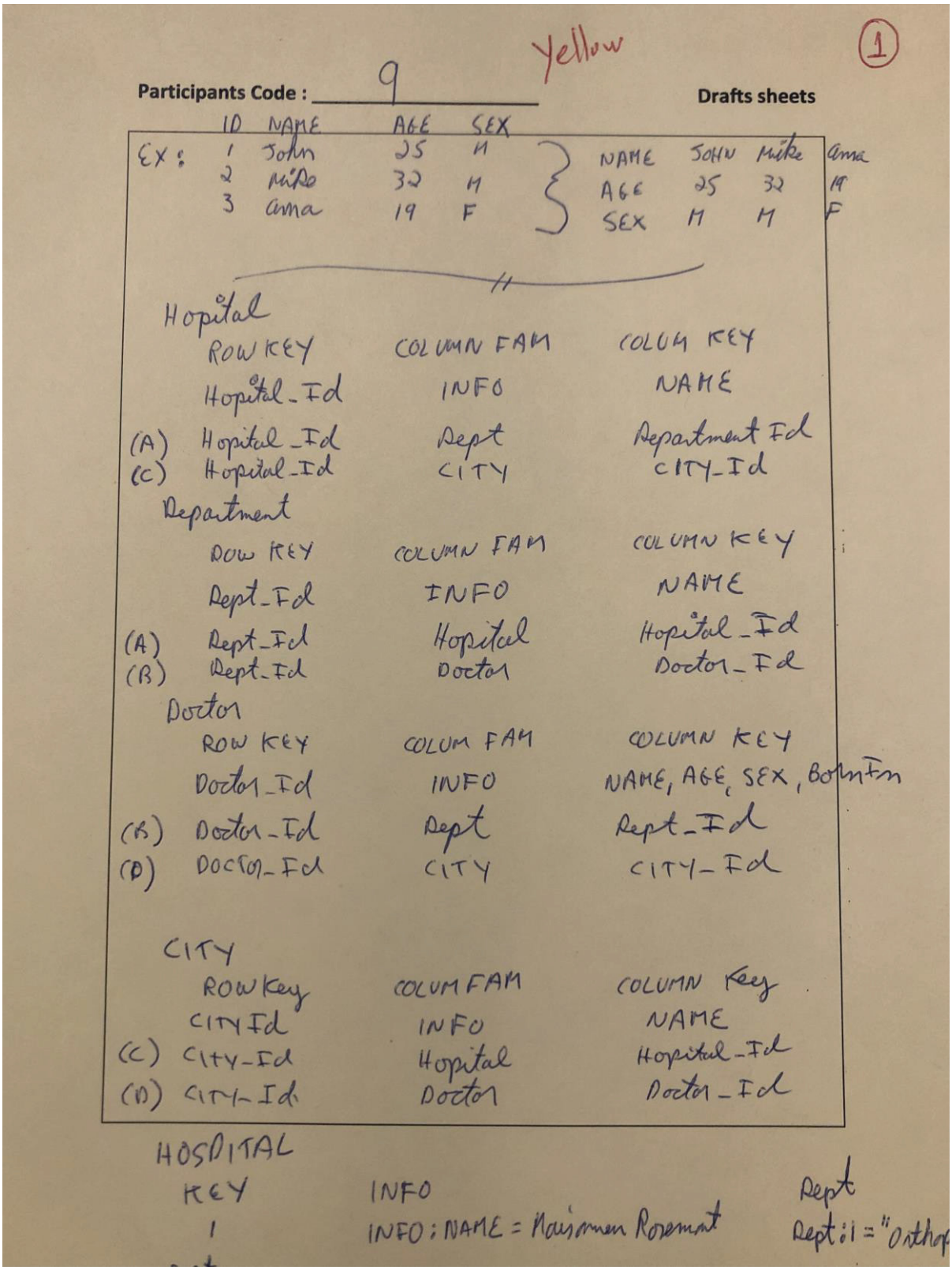
DepID — info: name

DepID — Doctor: ~~name, age~~ DoctorID

Doctor

DoctorID — Info: name, age, Sex, BornIn

DoctorID — Department: DepID



Participants Code : 9 (Yellow) (1)

Drafts sheets

	ID	NAME	AGE	SEX
EX:	1	John	25	M
	2	Mike	32	M
	3	Ama	19	F

NAME	John	Mike	Ama
AGE	25	32	19
SEX	M	M	F

Hopital

	ROW KEY	COLUMN FAM	COLUM KEY
	Hopital_Id	INFO	NAME
(A)	Hopital_Id	Dept	Department Id
(C)	Hopital_Id	CITY	CITY_Id

Department

	ROW KEY	COLUMN FAM	COLUMN KEY
	Dept_Id	INFO	NAME
(A)	Dept_Id	Hopital	Hopital_Id
(B)	Dept_Id	Doctor	Doctor_Id

Doctor

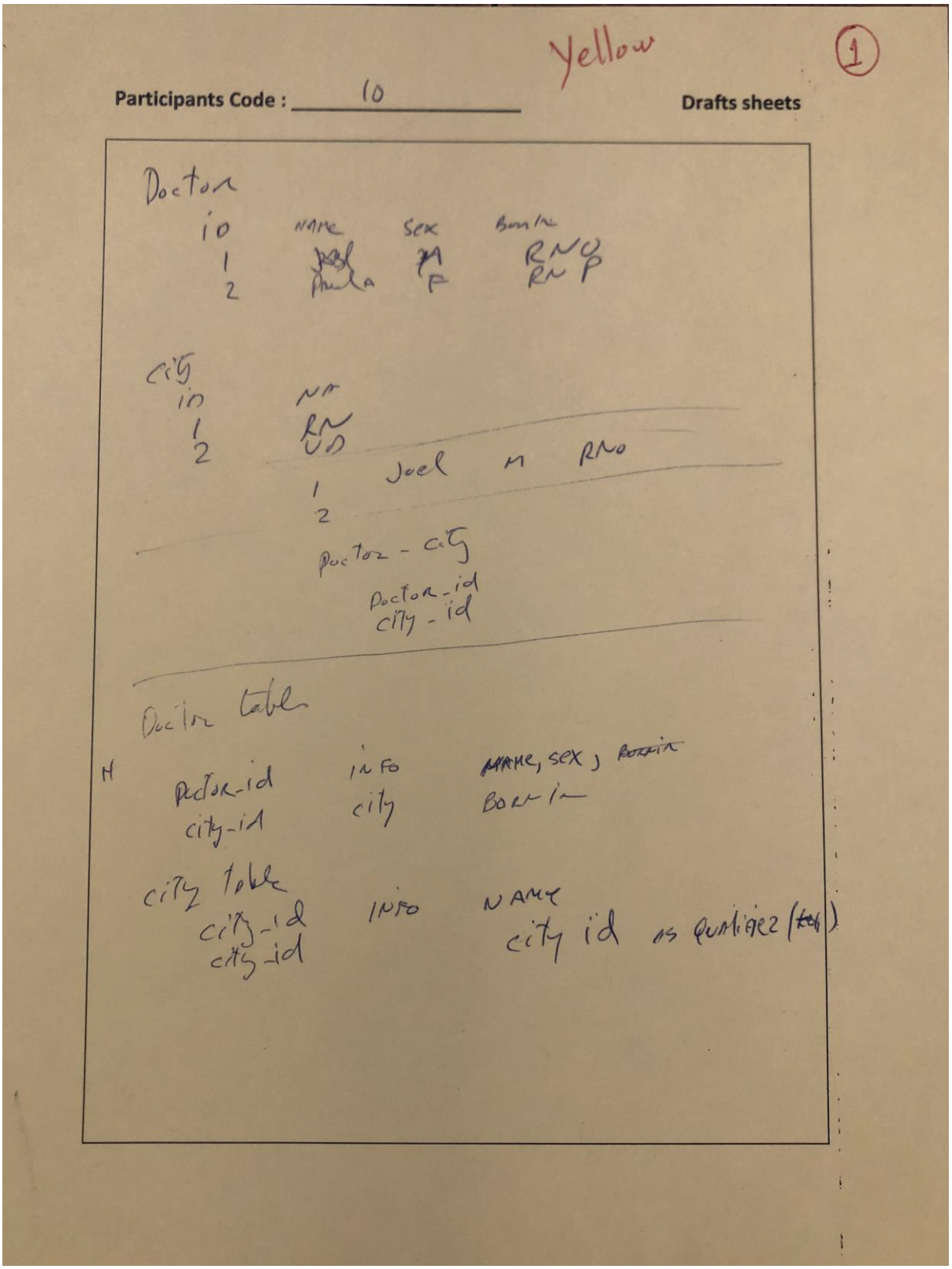
	ROW KEY	COLUM FAM	COLUMN KEY
	Doctor_Id	INFO	NAME, AGE, SEX, BornIn
(B)	Doctor_Id	Dept	Dept_Id
(D)	Doctor_Id	CITY	CITY_Id

CITY

	ROW Key	COLUM FAM	COLUMN Key
	CITY Id	INFO	NAME
(C)	CITY_Id	Hopital	Hopital_Id
(D)	CITY_Id	Doctor	Doctor_Id

HOSPITAL

KEY	INFO	Dept
1	INFO: NAME = Naisimen Rosemat	Dept:1 = "Orthop



Yellow (1)

Participants Code : 10

Drafts sheets

Doctor

id	NAME	SEX	BORN IN
1	[illegible]	M	RNO
2	[illegible]	F	RN P

City

id	NA
1	RN
2	UD

1 Joel M RNo

2

Doctor - City

Doctor_id

City - id

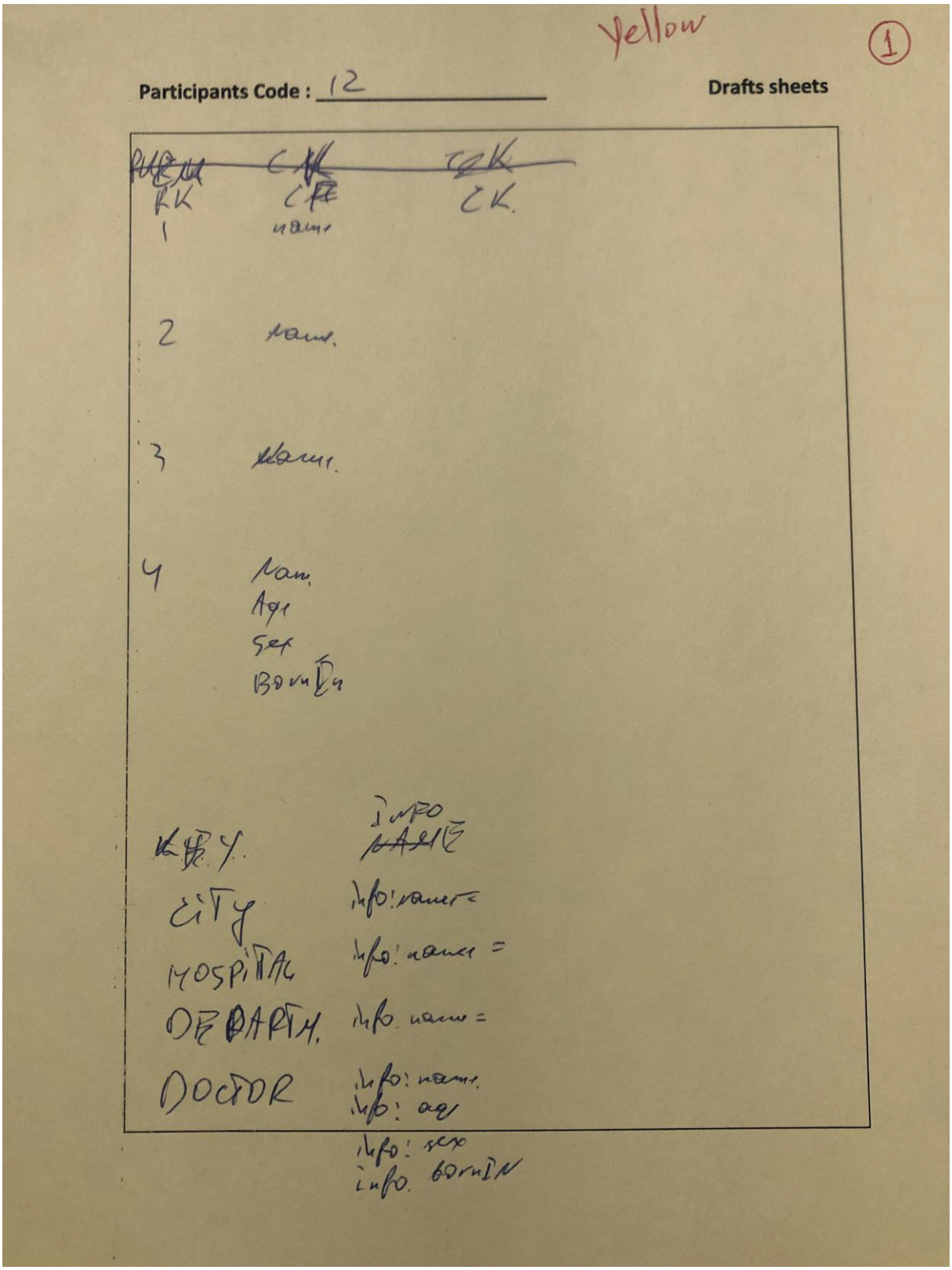
Doctor table

H

Doctor_id	INFO	NAME, SEX, BORN IN
city_id	city	BORN IN

City Table

city_id	INFO	NAME
city_id		city id as Qualified (key)



yellow

1

Participants Code : 12

Drafts sheets

~~RK~~ ~~CK~~ ~~CK~~

RK

CK

CK.

1 name

2 name.

3 name.

4 Name.
Age
Sex
Birthday

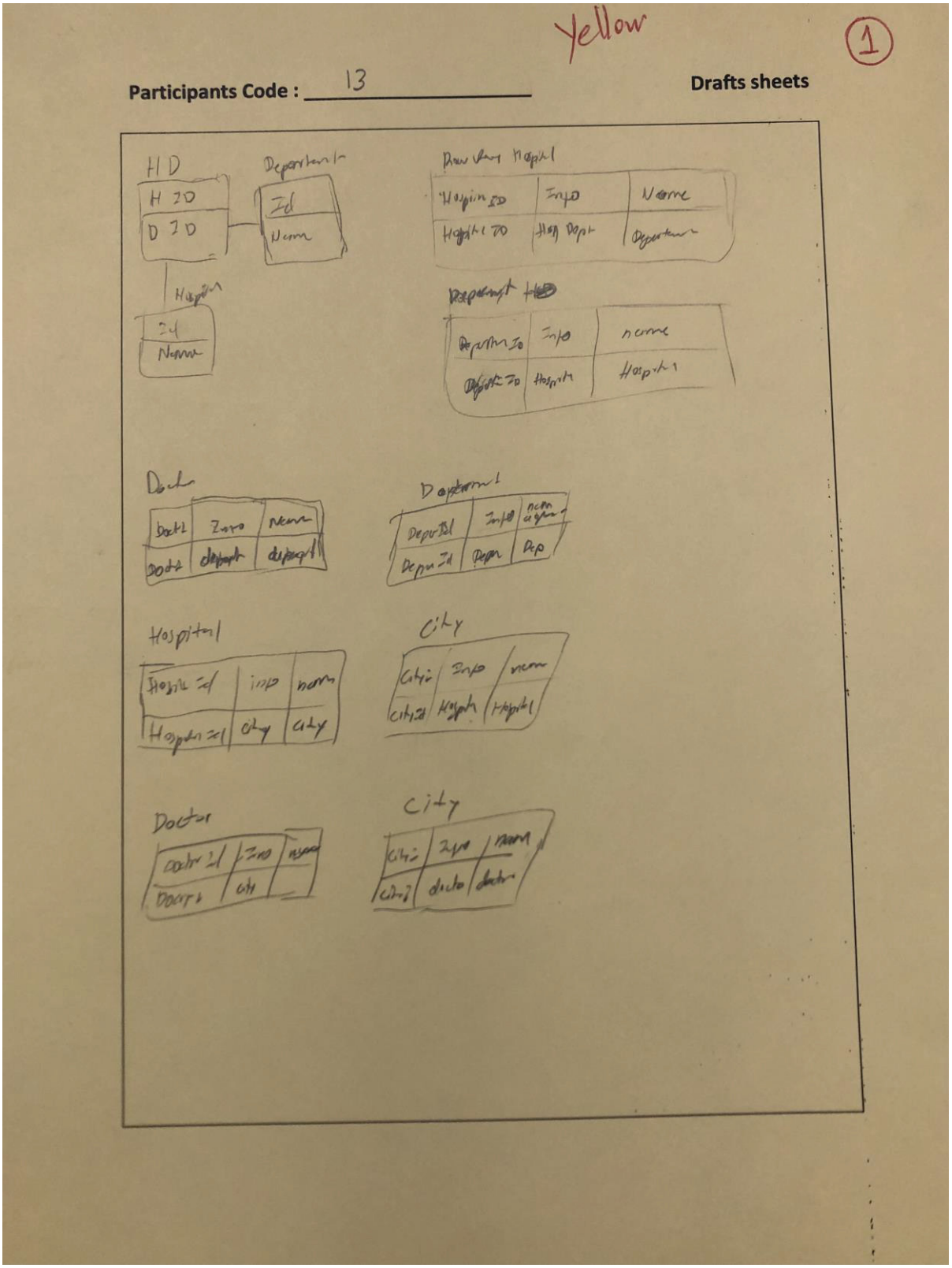
KEY. info table

CITY info: name =

HOSPITAL info: name =

DEPARTM. info. name =

DOCTOR info: name.
info: age
info: sex
info. birthday

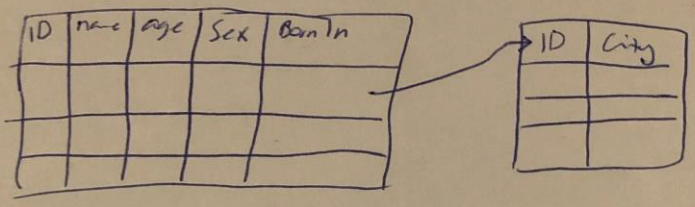

Yellow
1
Participants Code : 13
Drafts sheets
HID
Department
H ID
D ID
Id
Name
Hospital
Id
Name
Hospital
Hospital ID
Info
Name
Department
Department ID
Info
name
Doctor
Doctor
Department
Hospital
City
Doctor
City

Yellow

(1)

Participants Code : 14

Drafts sheets



yellow

(1)

Participants Code : 20

Drafts sheets

Hospital DOCTOR
Depart. CITY
H_Dep.
H CITY

Doct - Dep.

~~INFO:~~

H_INFO : H. Nam
: H. Dep.
H. City

D_INFO : D- Nam.
D - Dep.
D. City

LIST OF REFERENCES

Abadi, D. (2009). Data Management in the Cloud: Limitations and Opportunities. *IEEE Data Engineering Bulletin, 32*(1).

Abdel-Fattah, M., Mohamed, W., & Abdelgaber, S. (2022). A Comprehensive Spark-Based Layer for Converting Relational Databases to NoSQL. *Big Data and Cognitive Computing, 6*, 71. doi: 10.3390/bdcc6030071

Abran, A., Laframboise, L., & Pierre, B. (2003). A Risk Assessment Method and Grid for Software Measurement Programs. *Journal of Systems Management - Institute of Chartered Financial Analysts of India (ICFAI), 3*, 34.

Aiyer, A., Bautin, M., Chen, J., Damania, P., Khemani, P., Muthukkaruppan, K., . . . Vaidya, M. (2012). Storage Infrastructure Behind Facebook Messages Using HBase at Scale. Dans *Data Engineering Bulletin* (Vol. 35, pp. 10). Repéré à http://sites.computer.org/debull/A12june/A12JUN-CD.pdf

Al Mahruqi, R. S. (2020). *Migrating web Applications from SQL to NoSQL Databases* (Queen's University). Repéré à https://qspace.library.queensu.ca/bitstream/handle/1974/27587/Al_Mahruqi_Rahma_Said_202001_Phd.pdf?sequence=3&isAllowed=y

Alotaibi, O., & Pardede, E. (2019). Transformation of Schema from Relational Database (RDB) to NoSQL Databases. *Data, 4*(4), 148. Repéré à https://www.mdpi.com/2306-5729/4/4/148

Amanor-Boadu, J. (2022). Predicting Server Platform Power Delivery Performance through Simulation, Measurement, and Correlation. *IEEE Instrumentation & Measurement Magazine, 25*(5), 53-60. doi: 10.1109/MIM.2022.9832824

Attebury, G., Baranovski, A., Bloom, K., Bockelman, B., Kcira, D., Letts, J., . . . Wuerthwein, F. (2009). Hadoop distributed file system for the Grid. Dans *Nuclear Science Symposium Conference Record (NSS/MIC), 2009 IEEE* (pp. 1056-1061).

Basili, V. R., Selby, R. W., & Hutchens, D. H. (1986). Experimentation In Software Engineering. *IEEE Transactions on Software Engineering, SE-12*(Compendex), 733-743.

Bourque, P., & Cote, V. (1991). An experiment in software sizing with structured analysis metrics. *Journal of Systems and Software, 15*(Copyright 1991, IEE), 159-172. Repéré à http://dx.doi.org/10.1016/0164-1212(91)90053-9

Brewer, E. A. (2000a). *Towards robust distributed systems* présentée à Proceedings of the nineteenth annual ACM symposium on Principles of distributed computing, Portland, Oregon, United States. doi: 10.1145/343477.343502. Repéré à http://delivery.acm.org/10.1145/350000/343502/p7-brewer.pdf?ip=142.137.251.19&CFID=33765333&CFTOKEN=44726574&__acm__=1311346453_b3ab11dc5f5d1fefde9a5d3428c9cb15

Brewer, E. A. (2000b). *Towards robust distributed systems (abstract)* présentée à Proceedings of the nineteenth annual ACM symposium on Principles of distributed computing, Portland, Oregon, USA. doi: 10.1145/343477.343502

Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A., Burrows, M., . . . Gruber, R. E. (2008). Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems, 26*(Compendex). Repéré à http://dx.doi.org/10.1145/1365815.1365816

Chen, J., & Lee, W. (2017). Data conversion from RDB to HBase. Dans *2017 IEEE 8th International Conference on Awareness Science and Technology (iCAST)* (pp. 170-175). doi: 10.1109/ICAwST.2017.8256439

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).

Codd, E. F. (1970). A relational model of data for large shared data banks. *Commun. ACM, 13*(6), 377-387. doi: 10.1145/362384.362685

Codd, E. F. (1971a). *A data base sublanguage founded on the relational calculus* présentée à Proceedings of the 1971 ACM SIGFIDET (now SIGMOD) Workshop on Data Description, Access and Control, San Diego, California. doi: 10.1145/1734714.1734718

Codd, E. F. (1971b). *Normalized data base structure: a brief tutorial* présentée à Proceedings of the 1971 ACM SIGFIDET (now SIGMOD) Workshop on Data Description, Access and Control, San Diego, California. doi: 10.1145/1734714.1734716

Cooper, B. F., Ramakrishnan, R., Srivastava, U., Silberstein, A., Bohannon, P., Jacobsen, H.-A., . . . Yerneni, R. (2008). PNUTS: Yahoo!'s hosted data serving platform. *Proc. VLDB Endow., 1*(2), 1277-1288. doi: 10.1145/1454159.1454167

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. Repéré à http://dx.doi.org/10.1007/978-3-540-89403-2-11

Dean, J., & Ghemawat, S. (2008). MapReduce: Simplified data processing on large clusters. *Communications of the ACM, 51*(Compendex), 107-113. Repéré à http://dx.doi.org/10.1145/1327452.1327492

Desharnais, J. M., Pare, F., Maya, M., & St-Pierre, D. (1997). Implementing a Measurement Program in Software Maintenance - An Experience Report Based on Basili's Approach. Dans. International Function Point Users Group.

Dimiduk, N., & Khurana, A. (2013). HBase in Action.

Easterbrook, S., Singer, J., Storey, M.-A., & Damian, D. (2008). Selecting Empirical Methods for Software Engineering Research Guide to Advanced Empirical Software Engineering. Dans F. Shull, J. Singer & D. Sjøberg (Éds.), *Guide to Advanced Empirical Software Engineering* (pp. 285-311). Springer London. doi: citeulike-article-id:2229410
doi: 10.1007/978-1-84800-044-5_11. Repéré à http://dx.doi.org/10.1007/978-1-84800-044-5_11

Elmasri, R., & Navathe, S. (2016). *Fundamentals of database systems* (7 éd.). Pearson/Addison Wesley.

Erdogmus, H. (2009). Cloud Computing: Does Nirvana Hide behind the Nebula? *Software, IEEE, 26*(2), 4-6.

Fong, J. (1997). Converting relational to object-oriented databases. *SIGMOD Rec., 26*(1), 53-58. doi: 10.1145/248603.248614

Fong, J., & Chris, B. (1994). Data Conversion Rules from Network to Relational Databases. *Information Software Technology, 36*(3), 13.

Fong, J., Pang, F., & Bloor, C. (2001). Converting relational database into XML document. Dans *Database and Expert Systems Applications, 2001. Proceedings. 12th International Workshop on* (pp. 61-65).

Fong, J., & Wong, H. K. (2004). Replicate relational and XML databases for Internet computing. Dans *Information Technology: Coding and Computing, 2004. Proceedings. ITCC 2004. International Conference on* (Vol. 1, pp. 492-496 Vol.491).

Fuxman, A., Hernandez, M. A., Ho, H., Miller, R. J., Papotti, P., & Popa, L. (2006). *Nested mappings: schema mapping reloaded* présentée à Proceedings of the 32nd international conference on Very large data bases, Seoul, Korea.

Gantz, J., Reinsel, D., & Rydning, J. (2018). The Digitization of the World - From Edge to Core.

Georgia-Tech-Research-Institute. (2009). *no:sql east 2009*. Atlanta, Georgia: Conference Center -- Georgia-Tech-Research-Institute. Repéré à https://speakerrate.com/events/230-no-sql-east-2009

Ghemawat, S., Gobioff, H., & Leung, S.-T. (2003). The Google file system. *SIGOPS Oper. Syst. Rev., 37*(5), 29-43. doi: 10.1145/1165389.945450

Giddens, J. (2017). *Concepts for nursing practice*. St. Louis, Missouri : Elsevier, [2017]. Repéré à https://books.google.ca/books?id=lB-KCwAAQBAJ&lpg=PA430&ots=NRuXNCYpvJ&dq=concepts%20for%20nursing%20practice%20giddens%20pdf&pg=PP1#v=onepage&q&f=false

Gilbert, S., & Lynch, N. (2002). Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *SIGACT News, 33*(2), 51-59. doi: 10.1145/564585.564601

Gomez, A., Ouanouki, R., Ravanello, A., April, A., & Abran, A. (2015). Experimental Validation as Support in the Migration from SQL Databases to NoSQL Databases. Dans *CLOUD COMPUTING 2015 : The Sixth International Conference on Cloud Computing, GRIDs, and Virtualization* (pp. 147 to 153). Repéré à http://www.thinkmind.org/index.php?view=article&articleid=cloud_computing_2015_6_40_20057

Google Cloud Architecture Center (1). (2020). Database migration: Concepts and principles (Part 1). Repéré à https://cloud.google.com/architecture/database-migration-concepts-principles-part-1

Google Cloud Architecture Center (2). (2020). Database migration: Concepts and principles (Part 2). Repéré à https://cloud.google.com/architecture/database-migration-concepts-principles-part-2

Goyal, A., Swaminathan, A., Pande, R., & Attar, V. (2016). Cross platform (RDBMS to NoSQL) database validation tool using bloom filter. Dans *2016 International Conference on Recent Trends in Information Technology (ICRTIT)* (pp. 1-5). doi: 10.1109/ICRTIT.2016.7569537

Ha, M., & Shichkina, Y. (2022). Translating a Distributed Relational Database to a Document Database. *Data Science and Engineering, 7*(2), 136-155. doi: 10.1007/s41019-022-00181-9. Repéré à https://doi.org/10.1007/s41019-022-00181-9

Hadoop-webpage. (2011). The Apache Hadoop project. Repéré à http://hadoop.apache.org/

HBase-webpage. (2011). The Apache HBase. Repéré à http://hbase.apache.org/

Hive-webpage. (2011). The Apache Hive. Repéré à http://hive.apache.org/

Ippoliti, E. (2015). *Heuristic Reasoning* (Vol. 16). Springer. doi: 10.1007/978-3-319-09159-4

Kasunic, M. (2005). *Designing an Effective Survey*. doi: citeulike-article-id:3936216. Repéré à #

Kleppmann, M. (2017). *Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems*. O'Reilly Media. Repéré à https://books.google.ca/books?id=BM7woQEACAAJ

Kong, C., Gao, M., Qian, W., Zhou, M., Gong, X., Zhang, R., & Zhou, A. (2015). ACID Encountering the CAP Theorem: Two Bank Case Studies. Dans *2015 12th Web Information System and Application Conference (WISA)* (pp. 235-240). doi: 10.1109/wisa.2015.63

Kossmann, D., Kraska, T., & Loesing, S. (2010). *An evaluation of alternative architectures for transaction processing in the cloud* présentée à Proceedings of the 2010 international conference on Management of data, Indianapolis, Indiana, USA. doi: 10.1145/1807167.1807231

Koto, A., Kono, K., & Yamada, H. (2014). A Guideline for Selecting Live Migration Policies and Implementations in Clouds. Dans *2014 IEEE 6th International Conference on Cloud Computing Technology and Science* (pp. 226-233). doi: 10.1109/CloudCom.2014.36

Kumar, A., Kumar, M. S., & Namdeo, V. (2021). A Regression-based Hybrid Machine Learning Technique to Enhance the Database Migration in Cloud Environment. Dans *2021 International Conference on Computing, Communication, and Intelligent Systems (ICCCIS)* (pp. 149-155). doi: 10.1109/ICCCIS51004.2021.9397123

Kuszera, E. M., Peres, L. M., & Fabro, M. D. D. (2019). *Toward RDB to NoSQL: transforming data with metamorfose framework* présentée à Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing, Limassol, Cyprus. doi: 10.1145/3297280.3299734. Repéré à https://doi.org/10.1145/3297280.3299734

Lakshman, A., & Malik, P. (2010). Cassandra: a decentralized structured storage system. *SIGOPS Oper. Syst. Rev., 44*(2), 35-40. doi: 10.1145/1773912.1773922

Lam, C. (2011). *Hadoop in Action*. Stamford, CT: Manning Publications Co.

Lars, G. (2011). *HBase: The Definitive Guide* (1 éd.). O'Reilly Media.

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.

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

Maatuk, A., Ali, M. A., & Rossiter, N. (2011). *Re-engineering relational databases: the way forward* présentée à Proceedings of the 2011 International Conference on Intelligent Semantic Web-Services and Applications, Amman, Jordan. doi: 10.1145/1980822.1980839

Maatuk, A. M., Abdelaziz, T., & Ali, M. A. (2020). Migrating Relational Databases into XML Documents. Dans *2020 21st International Arab Conference on Information Technology (ACIT)* (pp. 1-11). doi: 10.1109/ACIT50332.2020.9299967

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

Microsoft. (2011). MSDN Microsoft. Repéré

Ouanouki, R., April, A., Abran, A., Gomez, A., & Desharnais, J.-M. (2017). *Toward building RDB to HBase conversion rules* (Vol. 4). doi: 10.1186/s40537-017-0071-x

Özsu, M. T., & Valduriez, P. (2011). *Principles of Distributed Database Systems*. Springer New York. Repéré à https://books.google.ca/books?id=TOBaLQMuNV4C

Pearl, J. (1984). *Heuristics: Intelligent search strategies for computer problem solving*. Addison-Wesley Pub. Co., Inc.,Reading, MA.

Pierce, B. A. (2012). *Genetics: A Conceptual Approach*. W. H. Freeman.

Raouf, A. E. A., Abo-Alian, A., & Badr, N. L. (2021). A Predictive Multi-Tenant Database Migration and Replication in the Cloud Environment. *IEEE Access, 9*, 152015-152031. doi: 10.1109/ACCESS.2021.3126582

Rocha, L., Vale, F., Cirilo, E., Barbosa, D., & Mourão, F. (2015). A Framework for Migrating Relational Datasets to NoSQL. *Procedia Computer Science, 51*, 2593-2602. doi: https://doi.org/10.1016/j.procs.2015.05.367. Repéré à https://www.sciencedirect.com/science/article/pii/S1877050915011758

Salmen, D., Malyuta, T., Fetters, R., & Norbert, A. (2009). Cloud Data Structure Diagramming Techniques and Design Patterns.

Sandhu, A. K. (2022). Big data with cloud computing: Discussions and challenges. *Big Data Mining and Analytics, 5*(1), 32-40. doi: 10.26599/BDMA.2021.9020016

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/cloud.2015.21

Shuchih Ernest, C., Kuo-Ming, C., & Yu-Ching, C. (2015). Cloud migration: Planning guidelines and execution framework. Dans *2015 Seventh International Conference on Ubiquitous and Future Networks* (pp. 814-819). doi: 10.1109/icufn.2015.7182656

Singh, P. (2010). Schema Guidelines & Case Studies. Repéré à http://www.paxcel.net/site Web |URL| doi:DOI

Stonebraker, M. (1986). The Case for Shared Nothing. *A quarterly bulletin of the IEEE computer society technical committee on Database Engineering, 9*(1), 6.

Stonebraker, M. (2008). Technical perspective: One size fits all: an idea whose time has come and gone. *Commun. ACM, 51*(12), 76-76. doi: 10.1145/1409360.1409379

Stonebraker, M., & Kepner, J. (2012). Possible Hadoop Trajectories. Repéré le May 2, 2012 à http://cacm.acm.org/blogs/blog-cacm/149074-possible-hadoop-trajectories/fulltext

Stonebraker, M., Madden, S., Abadi, D. J., Harizopoulos, S., Hachem, N., & Helland, P. (2007). *The end of an architectural era: (it's time for a complete rewrite)* présentée à Proceedings of the 33rd international conference on Very large data bases, Vienna, Austria.

Tawfik, M., Al-Zidi, N. M., Alsellami, B., Al-Hejri, A. M., & Nimbhore, S. (2021). Internet of Things-Based Middleware Against Cyber-Attacks on Smart Homes using Software-Defined Networking and Deep Learning. Dans *2021 2nd International Conference on Computational Methods in Science & Technology (ICCMST)* (pp. 7-13). doi: 10.1109/ICCMST54943.2021.00014

Tow, D. (2003). *SQL Tuning*. O'Reilly & Associates, Inc.

Venner, J. (2009). *Pro Hadoop*. Apress. doi: citeulike-article-id:5014567. Repéré à http://www.amazon.ca/exec/obidos/redirect?tag=citeulike09-20&path=ASIN/1430219424

Wagner, C., Hudic, A., Maksuti, S., Tauber, M., & Pallas, F. (2015). Impact of Critical Infrastructure Requirements on Service Migration Guidelines to the Cloud. Dans *2015 3rd International Conference on Future Internet of Things and Cloud* (pp. 1-8). doi: 10.1109/FiCloud.2015.79

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

Zelkowitz, M. V., Wallace, D. R., & Binkley, D. W. (2003). Experimental validation of new software technology. Dans *Lecture notes on empirical software engineering* (pp. 229-263). World Scientific Publishing Co., Inc.